Systems Research Center

Compaq Personal Jukebox File System Specification

ABSTRACT

The Compaq Personal Jukebox, or PJB, is a portable, battery powered device that stores the equivalent of 100 compact discs in compressed (MP3) format. The PJB itself consists of a DSP, DRAM, flash, LCD, buttons, USB interface, and a 2.5" IDE hard drive. This document describes the USB interface and "on-disk" structure. Readers of this document should be able to write their own file system managers for the Personal Jukebox.

Author:	Compaq Corporate Research Compaq Systems Research Center 130 Lytton Avenue Palo Alto, CA 94301
Document Version:	1.0
Edit Number:	59
Last Revised:	04/12/00 5:20 PM
Classification:	Draft

This page is supposed to be blank.
(but now that we've mentioned that, it isn't blank is it?)

Table Of Contents

1. Introduction

1.1 Copyright

This document should have been distributed along with the example source files that implement the algorithms and protocols described here.

1.2 Project Description

The Personal Jukebox, or *PJB*, is a personal audio appliance developed at Compaq's Systems Research Center in Palo Alto, California. Powered by a lightweight rechargeable battery or by an AC adapter, the PJB can hold up to 75 hours of CD-quality audio. It is small enough to fit into your jacket pocket, yet it produces sufficient quality audio to drive your home music system. With a fully charged battery, a PJB will yield about 10 hours of continuous playback.

Inside, the PJB is a fairly powerful special-purpose computer. It contains a Motorola 56309 digital signal processor, a 4.8 GByte hard disk, 12 MB of memory, 1 MB of flash memory, a USB interface, a digital-to-analog converter, and a small LCD display. We currently use MPEG-2 layer-3 encoding technology (MP3) from Fraunhofer IIS (see http://www.iis.fhg.de) to store compressed CD-quality digital audio on the hard disk. This results in a 11:1 size reduction over raw digital audio with little noticeable difference in sound quality (even when you play it over your home stereo). Because the PJB's software is stored in flash ROM and the CPU is a general-purpose DSP, it is easy to upgrade it to use other compression algorithms, or even to use different algorithms for different tracks.

This document is the *filesystem* guide. It contains, among other things, the following information:

- A description of the USB protocol between the PJB and a host computer
- A description of the on-disk structure for storing MP3 data
- A description of the "table of contents" file
- A description of the example source for the GNU Public License PJB library code

1.3 Related Documents

The following documents may be useful:

- USB Specification, version 1.1 (see http://www.usb.org)
- GNU Public License (http://www.gnu.org)

1.4 Revision History

This section contains a list of revisions to this document.

Who	When	What
Lichtenberg	2/1/2000	Created this file.

1.5 Notation Conventions

This section lists special notation that is used in this document.

Notation	Description
`Fixed-Width Text`	Information displayed by the computer, names of files, or directories, data structures, etc.
`Italic Fixed Text`	Information you type into the computer (commands, answers to prompts)
Bold or *Italic* comments	Special notes or caveats

2. USB Protocol Description

2.1 Overview

The Personal Jukebox has a standard USB "B" type connector on one edge. Inside the PJB is a Philips PDIUSBD12 USB peripheral controller, attached to the memory bus of the DSP. Raw throughput to the USB interface averages 400Kbytes/second.

The PJB's USB interface is "vendor-specific." It does not correspond to any of the defined USB device classes. In particular, the PJB's disk does not appear as a USB disk device.

The protocol between the host and the PJB is a simple request-response protocol, with message boundaries. A short USB packet transfer is used to signal the end of a packet.

2.2 USB Vendor and Product ID

Field	Hex	Decimal
Vendor ID	0x49F	1183
Product ID	0x504A	20554

2.3 Windows Global Unique ID

The GUID for the PJB's Windows device driver is:

```
{5be02160-e380-11d2-bbcc-0000f879fc04}
```

Pipes "2" and "3" from the device driver bound to this GUD are the input and output pipes, respectively.

2.4 Endpoint Descriptions

For more information, consult chapter 9 of the USB Specification.

The Philips PDIUSBD12 chip in the PJB supports three endpoints (three pairs of pipes). These endpoints are assigned as follows:

Endpoint	Transfer Size	Transfer Type	Description

Endpoint 0 Pipe 0 (out) Pipe 1 (in)	16	Bulk	Control endpoint, used only for USB control information
Endpoint 1 Pipe 2 (out) Pipe 3 (in)	16	Bulk	Debug endpoint, used only with special firmware in the PJB to retrieve debug messages
Endpoint 3 Pipe 4 (out) Pipe 5 (in)	64	Bulk	Main endpoint. All PJB commands and responses are sent via this endpoint.

2.4.1 Configuration Descriptors

There is only one configuration descriptor, with bConfigurationValue = 1. The configuration descriptor will return the following values:

Field	**Value**
bNumInterfaces	1
iConfigurationValue	1
iConfiguration	0
bmAttributes	0xC0
MaxPower	0

There is only one interface descriptor, with descriptor bInterfaceNumber = 0. The interface descriptor will return the following values:

Field	**Value**
BInterfaceNumber	0
BAlternateSetting	0
BNumEndpoints	4
BInterfaceClass	USB_CLASS_CODE_VENDOR_SPECIFIC (0xFF)
BInterfaceSubClass	USB_SUBCLASS_CODE_VENDOR_SPECIFIC (0xFF)
BInterfaceProtocol	USB_PROTOCOL_CODE_VENDOR_SPECIFIC (0xFF)
IInterface	0

The four endpoints will return the following descriptor values:

Endpoint	**bEndpoint Address**	**bmAttributes**	**wMaxPacketSize**	**bInterval**
EP1 (IN)	0x81	BULK (0x02)	16	0
EP1 (OUT)	0x01	BULK (0x02)	16	0
EP2 (IN)	0x82	BULK (0x02)	64	0
EP2 (OUT)	0x02	BULK (0x02)	64	0

2.4.2 Power Management

The PJB is considered a self-powered device. It does not draw any current from the USB.

2.5 USB-specific protocol header

After the PJB has been configured onto the USB, an application may send messages to the main endpoint.

Each USB packet has a 10-byte header that is sent before the PJB message. This header is used for some rudimentary error checking and to allow some future protocol multiplexing (to share the main endpoint among other applications). The header has the following format:

Field	Size (bytes)	Description
Seal	4	Contains the value {0xFF, 'P', 'J', 'B'}. This is a "seal" that can be used in debugging to verify you are at the beginning of a message.
Length	2	The length of the data that follows this header. The low-order byte is transmitted first.
Checksum	2	A simple checksum of the data field that follows the header. Simply add up the values of all the bytes.
Protocol code	1	Identifies the target process on the PJB. For now, there is only one target, the PJB itself. Fill this value in as zero.
MBZ	1	This field must be zero.

Note: The messages should not be an even multiple of 64 bytes in length. The PJB's firmware and the USB driver on Windows treat USB "short packets" as an end-of-message indication. The *length* field in the header in the table above will contain the real message length (it should not be incremented if the (data+header) portion is an exact multiple of 64 bytes.

Some USB implementations allow you to send zero-length packets. The PJB's firmware will correctly interpret a 0-length packet as an end-of-message indication. The Windows drivers do not let you do this, so multiple-of-64 byte messages are padded with one extra byte to force the short message to occur.

2.6 PJB Terms

The PJB's file system has some unique terms that you may find in this specification and in the sample source code.

Term	Description
Block or *Allocation Block* or *Allocation Unit*	Refers to a 128Kbyte unit of storage on the PJB's disk. Disk space is allocated in allocation units. In terms of playback, an allocation unit is approximately 8 seconds of MP3 at 128kbps.

Click	Refers to a 1Kbyte piece of an allocation unit. Because the transfer size over the PJB's USB driver is limited to 1500 bytes, requests to write allocation units are broken into 128 smaller requests. Each piece of an allocation unit is called a *click*. For the most part, any data structure in the PJB that occupies 1K is called a click.
TOC	Table of contents. This file describes the contents of the PJB and the hierarchy of the Sets, Discs, and Tracks on the disk.
Set	A set of "Disc" entries in the table of contents.
Disc	A set of "Track" entries in the table of contents
Track	A single unit of playback from the user interface's point of view. A CD (Disc) is a collection of tracks, and a collection of Discs is a set.
Rip Unit	A set of contiguous tracks extracted from a CD. The Jukebox Manager encodes contiguous tracks as a single bitstream to eliminate the gaps between tracks that may be audible on live recordings.
UID	Unique Identifier. The header of each PJB allocation block contains enough information to reconstruct the name of the CD that the block came from. This is also stored in the table of contents.
Allocation Map	The allocation map describes the free and allocated space on the PJB's disk. It can be calculated from the TOC file.

2.7 PJB Protocol Message Format

The PJB protocol messages consist of two parts: a *fixed header* and an *item list*. Fixed headers contain fields common to all message types, and item lists contain variable-length fields and other data unique to a message type.

2.7.1 Fixed Header

The fixed header has the following format:

Field	Size (bytes)	Description
Version	1	The version number of the protocol. For this version of the PJB protocol specification, the value is 1.
Class	1	The "Class" that the packet belongs to. This identifies a particular section of the PJB's firmware.
Command	1	The command code within the class. This identifies a particular command or response code. The low-order bit if this code indicates request if 0, or a response if 1.
Format	1	The format of the data that follows the header. This can include nothing, bulk data, an item list, or a special

		"continue" packet used to prevent a long-running command from timing out.
Status	1	The status (error code) from the request. This field is only used in response packets.
XID	4	Transaction ID. The PJB's firmware remembers this field and passes it back in the response packet. You can use this field to associate requests with their responses.
Block	4	For commands that require it, this field contains the block number (also called the allocation unit number) on the PJB's disk.
Click	2	For commands that require it, this field contains the "click" number (sub-block) within the specified block on the disk.

These 15 bytes are at the beginning of every PJB protocol message.

2.7.2 Class and command codes

The following classes are currently defined in the PJB protocol:

Class name	Number	Description
PJB_PCLASS_CONFIG	0	Configuration and status messages
PJB_PCLASS_TOC	1	Table of contents management
PJB_PCLASS_STORAGE	2	Disk storage management
PJB_PLCASS_UPGRADE	3	PJB software upgrade management

Refer to the next chapter for the list of command codes in each class.

2.7.3 Message formats

The format field indicates what follows the fixed header.

Class name	Number	Description
PJB_FMT_NODATA	0	There is no data following the fixed header.
PJB_FMT_BULKDATA	1	The data following the header is "bulk" (unformatted) data. This is typically used to transfer disk sectors to the PJB.
PJB_FMT_ITEMLIST	2	An item list follows the fixed header.
PJB_FMT_CONTINUE	3	There is no data following the fixed header. When the client receives this message from a PJB, it should reset its timeout timer and continue to wait for a response.
PJB_FMT_CONTINUE_ITEM	4	Similar to PJB_FMT_CONTINUE except an item list is also included in the message. This is used only with the manufacturing diagnostics.

2.7.4 Item List Format

Some commands include an *item list*, which can be used to describe variable-length data. Item lists are also convenient because new items can be added without breaking backwards compatibility with old code. Each item in the item list has the following format:

Field	Size (bytes)	Description
ItemCode	2	A number that uniquely identifies the item's type and purpose. The bytes are transmitted in little-endian format (LSB first).
ItemLength	2	The length of the data that follows this descriptor. The bytes are transmitted in little-endian format (LSB first).
ItemData	'n'	The actual item data.

Item lists terminate with an item code of zero.

The ItemCode field also encodes the type of data in the item. The item's type is stored in the upper 4 bits of the item code, as described in the following table:

Type Name	Type#	Description
IT_UINT	0x1000	An unsigned integer. 32 bits on the wire, but machine word size on the CPU. The bytes are transmitted in little-endian format (LSB first).
IT_STRING	0x2000	ASCII string, null terminated. The null byte should be included in the data field and length.
IT_U32	0x3000	An unsigned number, always 32 bits. The bytes are transmitted in little-endian format (LSB first).
IT_U16	0x4000	An unsigned number, always 16 bits. The bytes are transmitted in little-endian format (LSB first).
IT_U8	0x5000	An unsigned number, always 8 bits.
IT_4BREC	0x6000	A 4-byte record. The data is transferred exactly as it appears on the local CPU.
IT_REC	0x7000	A general record type. The data is transferred exactly as it appears on the local CPU.

To form an item code, bit-wise OR the ItemType with a number that uniquely identifies it. For example, 0x2004 is a string field.

2.8 Typical packet exchange

To build a PJB packet, assemble a buffer with the USB-specific protocol header (section 2.5) followed by the fixed PJB protocol header (section 2.7.1) followed by the item list (section 2.7.4) or bulk data (if any).

The pseudocode below illustrates a simple packet exchange with a PJB:

```
Do_command()
{
     Transmit_command_packet();
     Do {
          Receive_response_packet();
          If (timed_out) break;
     } while (response_packet.format != PJB_FMT_CONTINUE);
}
```

2.9 Status Codes

The following status codes are defined:

Name	Number	Description
FS_ERR_NODISC	-1	There is no disk in the PJB to query
FS_ERR_IO	-2	An I/O error occurred on the disk
FS_ERR_ALLOCBLOCK	-3	Allocation block number out of range
FS_ERR_OFFSET	-4	Offset (click number) out of range
FS_ERR_CHECKSUM	-5	Checksum mismatch
FS_ERR_NOTOC	-6	There is no TOC on this PJB
FS_ERR_BUFTOOSMALL	-7	Supplied buffer is too small
FS_ERR_HWERR	-8	Hardware failure
PJB_ERR_COMM	-100	Communications error
PJB_ERR_TIMEOUT	-101	Command timeout
PJB_ERR_NO_RESOURCES	-102	Out of memory, buffers, etc.
PJB_ERR_NODEVICE	-103	(obsolete)
PJB_ERR_DATACORRUPT	-104	Diagnostic error – echoed data is corrupt
PJB_ERR_PROTOCOLERR	-105	Invalid protocol, invalid fields in USB packet
PJB_ERR_NOBIND	-106	(obsolete)
PJB_ERR_INVPARM	-107	Invalid parameter
PJB_ERR_PJBINUSE	-108	(obsolete)

3. PJB Protocol Messages

3.1 Echo Request (MTC_ECHO)

Description:

Echo request. This command causes the PJB to transmit the data back to the sender.

This command also works in flash update mode

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_CONFIG (0)
Command	MTC_ECHO (0)
Format	PJB_FMT_BULKDATA (1)
Status	0
XID	Transaction ID
Block	0
Click	0

Data Field:

Data to be echoed.

3.2 Echo Response (MTR_ECHO)

Description:

Echo response. This is the response to an MTC_ECHO message.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_CONFIG (0)
Command	MTR_ECHO (1)
Format	PJB_FMT_BULKDATA (1)
Status	Status code (0=ok)

XID	Transaction ID
Block	0
Click	0

Data Field:

Data from the original MTC_ECHO message.

3.3 Solicit Request (MTC_SOLICIT)

Description:

Solicit Request. This command is used to probe a PJB for its software and hardware version number and name. It is used in the Jukebox Manager to obtain the name and serial number of a PJB.

This command also works in flash update mode. The PJB's software version will return 0.0.0, so you can use this version number to detect a PJB that is already in flash update mode.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_CONFIG (0)
Command	MTC_SOLICIT (2)
Format	PJB_FMT_NODATA (0)
Status	0
XID	Transaction ID
Block	0
Click	0

Data Field:

None.

3.4 Solicit Response (MTR_SOLICIT)

Description:

Solicit response. This is the response to an MTC_SOLICIT message. It contains information about the connected PJB such as the hardware and software version #, serial number, etc.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_CONFIG (0)
Command	MTR_SOLICIT (3)
Format	PJB_FMT_ITEMLIST (2)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

An item list, with the following item codes

Field	Code (hex)	Type	Description
IT_HWVER	1009	Int	Hardware version (2 for most PJBs on the market today)
IT_SWVER	100A	Int	Software version. The low 8 bits are the software ECO number. The next 8 bits are the software revision, and the next 8 bits are the major version. For example, software version 2.3.4 would be represented as 0x00020304.
IT_FRIENDLYNAME	200B	String	The PJB's "friendly name". This is the name of the PJB as you see it when the unit is booted. It is also the name from the topmost item in the table of contents.
IT_FEATURES	100C	Int	Bitmask of new features. Currently, this value is zero.
IT_SSN	7012	Record 6 bytes	The software serial number of the PJB (6 bytes).

3.5 Get Filesystem Info Request (MTC_GETINFO)

Description:

This command is used to obtain information about the PJB's file system.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)

Class	PCLASS_CONFIG (0)
Command	MTC_GETINFO (4)
Format	PJB_FMT_NODATA (0)
Status	0
XID	Transaction ID
Block	0
Click	0

Data Field:

None.

3.6 Get Filesystem Info Reponse (MTR_GETINFO)

Description:

Filesystem info response. This is the response to an MTC_GETINFO message. It contains information about the PJB's disk and file system.

Fixed Header:

Field	**Description**
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_CONFIG (0)
Command	MTR_GETINFO (5)
Format	PJB_FMT_ITEMLIST (2)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

An item list, with the following item codes

Field	**Code (hex)**	**Type**	**Description**
IT_FSVERSION	1002	Int	File system version number (0)
IT_CLICKSINTOC	1003	Int	Maximum number of clicks that can be stored in the TOC. The client application should not attempt to store more than this number in the PJB's TOC.
IT_TOTALBLOCKS	1004	Int	The total number of allocation units that the

			disk supports.
IT_ERRORCOUNT	1005	Int	Count of disk failures on this disk (cumulative)
IT_VALIDTOCS	1006	Int	The number of valid copies of the table of contents on this disk. You can use this to determine if the disk is blank, since the value will be zero in this case.
IT_CURTOCCLICKS	1007	Int	Number of clicks in the current TOC.
IT_ISOLDTOC	1008	Int	Will return TRUE if the current TOC is not the most recent one. The PJB will only return this as TRUE if there are disk failures reading the most recent TOC.

3.7 Read Table-of-contents Request (MTC_READTOC)

Description:

This command reads a single click of the table-of-contents file.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTC_READTOC (0)
Format	PJB_FMT_NONE (0)
Status	0
XID	Transaction ID
Block	0
Click	Click number within the table of contents file (0 is the first click).

Data Field:

None.

3.8 Read Table-of-contents Response (MTR_READTOC)

Description:

This is the response to the MTC_READTOC message. It contains one click of data from the table of contents file. If an attempt is made to read past the end of the table of contents, a status code is returned.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTR_READTOC (1)
Format	PJB_FMT_BULK (1)
Status	Status code (0=ok)
XID	Transaction ID
Block	Status code (0=ok)
Click	Click number within the table of contents file (0 is the first click).

Data Field:

The data field contains 1024 bytes of data from the table of contents file.

3.9 Write Table-of-contents Request (MTC_WRITETOC)

Description:

This command writes a single click to the unused table-of-contents file. The PJB maintains two table-of-contents files. Once all of the data is written, use the MTC_COMMITDIR command to make the new TOC active.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTC_WRITETOC (2)
Format	PJB_FMT_BULK (2)
Status	0
XID	Transaction ID
Block	0
Click	Click number within the table of contents file (0 is the first click).

Data Field:

The data field contains 1024 bytes of data to be written to the table of contents file..

3.10 Write Table-of-contents Response (MTR_WRITETOC)

Description:

This is the response to the MTC_WRITETOC message.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTR_WRITETOC (3)
Format	PJB_FMT_NONE (0)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	Click number within the table of contents file (0 is the first click).

Data Field:

None.

3.11 Commit Table-of-contents Request (MTC_COMMITTOC)

Description:

Commits the current TOC file and makes it the active one. This command causes the PJB to checksum the TOC file that was being transmitted via MTC_WRITETOC. If the checksum matches the one in the item list to this message, the new TOC is activated.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTC_COMMITTOC (4)
Format	PJB_FMT_ITEMLIST (2)
Status	0
XID	Transaction ID
Block	0

Click	0

Data Field:

An item list, with the following item codes

Field	Type	Description
IT_CLICKSINTOC	Int	Number of clicks in the new TOC
IT_CHECKSUM	4BREC	The checksum for the new TOC (transmitted as a 4-byte record).

The checksum field is calculated by using the cksum() routine on the entire table of contents file. See the example code for the implementation of cksum().

3.12 Commit Table-of-contents Response (MTR_COMMITTOC)

Description:

This is the response to the MTC_COMMITTOC message. If the checksum does not match, the error code will be returned here.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTR_COMMITTOC(5)
Format	PJB_FMT_NONE (0)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

None.

3.13 Read Block Header Request (MTC_READBLOCKHDR)

Description:

This command reads the block header (the first click of a given allocation unit). Inside the PJB, the entire allocation unit is read and checksummed, and the checksum is checked against the value stored in the header.

This command can be used to write a disk scavenger program to reconstruct a destroyed TOC, or to verify the file system by following the link pointers.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_STORAGE (2)
Command	MTC_READBLOCKHDR (0)
Format	PJB_FMT_NONE (0)
Status	0
XID	Transaction ID
Block	Allocation unit (block number)
Click	0

Data Field:

None.

3.14 Read Block Header Response (MTR_READBLOCKHDR)

Description:

This is the response to the MTC_READBLOCKHDR message. The data field of this message contains the first click of the requested allocation unit.

In the event of a checksum error, the requested data block is still returned.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_STORAGE (2)
Command	MTR_READBLOCKHDR (1)
Format	PJB_FMT_BULK (1)
Status	Status code (0=ok)
XID	Transaction ID
Block	Allocation unit (block number)
Click	0

Data Field:

The first click of data from the specified block on the PJB's disk.

3.15 Write Block Request (MTC_WRITEBLOCK)

Description:

This command writes a single click to a block on the disk. To fill an entire allocation unit, 128 Write Block Request (MTC_WRITEBLOCK) commands should be issued, one for each click in the block.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_STORAGE (2)
Command	MTC_WRITEBLOCK (2)
Format	PJB_FMT_BULK (2)
Status	0
XID	Transaction ID
Block	Allocation unit (block number)
Click	Click number within block (0..127)

Data Field:

The data field contains 1024 bytes of data to be written to the specified allocation unit.

3.16 Write Block Response (MTR_WRITEBLOCK)

Description:

This is the response to the MTC_WRITEBLOCK message. It contains the status of the write operation.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_TOC (1)
Command	MTR_WRITEBLOCK (3)
Format	PJB_FMT_NONE (0)

Status	Status code (0=ok)
XID	Transaction ID
Block	Allocation unit (block number)
Click	Click number within block (0..127)

Data Field:

None.

3.17 Software Upgrade Request (MTC_OPENSOFT)

Description:

This command only works in flash update mode.

The MTC_OPENSOFT command prepares the PJB for a software upgrade. It should be sent immediately prior to any MTC_WRITESOFT commands.

Fixed Header:

Field	**Description**
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_OPENSOFT (0)
Format	PJB_FMT_NODATA (0)
Status	0
XID	Transaction ID
Block	0
Click	0

Data Field:

Nothing.

3.18 Software Upgrade Response (MTR_OPENSOFT)

Description:

This command only works in flash update mode.

The MTR_OPENSOFT response indicates that the PJB is ready to accept MTC_WRITESOFT commands.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_OPENSOFT (1)
Format	PJB_FMT_NODATA (0)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

Nothing.

3.19 Write Software Request (MTC_WRITESOFT)

Description:

This command only works in flash update mode.

The MTC_WRITESOFT stores up to 1K of data in DRAM for eventual transfer to the flash. After all of the blocks of program data are stored via MTC_WRITESOFT, MTC_COMMITSOFT can be used to transfer this data to flash.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_WRITRESOFT (2)
Format	PJB_FMT_BULKDATA (2)
Status	0
XID	Transaction ID
Block	0
Click	“Click number” (1K block number) of program data to store in DRAM.

Data Field:

Nothing.

3.20 Write Software Response (MTR_WRITESOFT)

Description:

This command only works in flash update mode.

The MTR_WRITESOFT response indicates that the PJB has stored 1K of program data in DRAM and is ready for another MTC_WRITESOFT or MTC_COMMITSOFT message..

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_WRITESOFT (3)
Format	PJB_FMT_NODATA (0)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

Nothing.

3.21 Commit Software Request (MTC_COMMITSOFT)

Description:

This command only works in flash update mode.

The MTC_COMMITSOFT command causes the PJB to write the software that was collected in DRAM via the MTC_WRITESOFT command and commit it to flash. During the execution of this command, you may receive one or more CONTINUE responses, since it can take several seconds to execute.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_COMMITSOFT (4)
Format	PJB_FMT_ITEMLIST (1)
Status	0

XID	Transaction ID
Block	0
Click	0

Data Field:

An item list, with the following item codes

Field	Code (hex)	Type	Description
IT_SECTOR	1011	Int	Destination flash sector
IT_CHECKSUM	600C	4BREC	The checksum for the new flash image file (transmitted as a 4-byte record).

If the IT_SECTOR field has the value 2 (FLASHREC_MAINCODE), the program file is written to flash. If the IT_SECTOR field has the value 3 (FLASHREC_EXECUTE), the program file is executed in place without being written to flash.

3.22 Commit Software Response (MTR_COMMITSOFT)

Description:

This command only works in flash update mode.

The MTR_COMMITSOFT response indicates that the program has been successfully written to flash.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_COMMITSOFT (5)
Format	PJB_FMT_NODATA (0)
Status	Status code (0=ok)
XID	Transaction ID
Block	0
Click	0

Data Field:

Nothing.

3.23 Reboot Request (MTC_REBOOT)

Description:

The MTC_REBOOT request causes the PJB to be rebooted to the specified program in the flash. You can use this program to activate flash update mode from the player program, or to activate the player program from flash update mode.

This command works both in flash update mode and while the player program is running.

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_REBOOT (6)
Format	PJB_FMT_NODATA (0)
Status	0
XID	Transaction ID
Block	0
Click	0 to boot to the player program, 1 to boot to the flash update program

Data Field:

Nothing.

3.24 Reboot Response (MTR_REBOOT)

Description:

The MTR_REBOOT response indicates that the PJB is about to reboot.

Note: Sometimes the PJB reboots before this message can be sent. Be prepared for a timeout waiting for this message to arrive!

Fixed Header:

Field	Description
Version	PJB_PROTOCOL_VERSION (1)
Class	PCLASS_UPGRADE (2)
Command	MTC_REBOOT (7)
Format	PJB_FMT_NODATA (0)
Status	Status code (0=ok)
XID	Transaction ID

Block	0
Click	0

Data Field:

Nothing.

4. Table of Contents Format

4.1 Overview

The PJB's table of contents file (TOC) is a simple ASCII text file. The TOC contains a complete description of the contents of a PJB, including the allocated space map.

The TOC consists of records, which are lines of text. The terminating character for each record is the newline ('\n') character. The record type is the first character in the record, and the record's data immediately follows the record type up to the newline character.

Since the TOC is uploaded to the PJB 1KB at a time, you should pad the remaining parts of the last block with newline characters.

String fields may contain characters from the ISO-Latin-1 character set.

4.2 TOC record types

Format	Description
V*a.b*	Version record. The current TOC version is 2.0, so this record will always be "V2.0"
R*string*	"Root" record. This record contains the PJB's name (this is the same name that is displayed on the splash screen when the PJB is turned on).
S*string*	"Set" record. This identifies a PJB set (collection of discs). The *string* field is the name of the set.
D*string*	"Disc" record. This identifies a PJB disc (collection of tracks). The *string* field is the name of the disc.
T*string*	"Track" record. This identifies a PJB track (or song). The *string* field is the name of the track.
B*a b.c d.e*	Track Address record. This record contains the starting and ending location of a track on the disk. Field *a* is the encoding type (1 for MP3) Field *b.c* is the starting block number and byte offset within the block. Field *d.e* is the ending block number and byte offset within the block.

	For example, "B1 100.0 193.21365" is a track address record that indicates an MP3 file starting at allocation unit #100, offset 0, and ending at allocation unit #193, offset 21365.
I*a.b.c d*	Track identifier record. This record contains information about the source CD and the track's bitrate. Field *a* is the CD UID index. The UID for the source CD is the *a*'th entry in the UID table at the end of the TOC. Field *b* is the track number from the source CD. Field *c* is the playing time, measured in 75ths of a seocnd. If there is no source CD (in the case of an MP3 file stored on the user's PC), the *a* and *b* fields are –1. Field *d* is the encoded bitrate. If the bitrate field is less than 1024, it is in units of 8000 bits per second (128kbps is encoded as 16). If the field is more than 1024, it is the actual bitrate. For variable bitrate files, this field should be the average bitrate over the entire track.
U*string*	UID string record. There is one UID record for each "I" record that contains a UID index field. The *string* field is the CDDB query information, and can be used to determine if a particular CD is already present on the PJB, even though its track names have been changed.
.	End of TOC record. This should be the last record in the file, and it is also used to separate the TOC area from the allocation map.
+*a.b.c*	Allocation Map record. Field *a* is the first block in a run, and *b* is the length of the run. Field *c* is the last block of the previously allocated run within the same rip unit, or –1 if there is none. Refer to the source code and the next chapter more information about the allocation map.

4.3 Sample TOC

This is a sample TOC file. It demonstrates the order of the records in the file. This is an actual, valid TOC describing three CDs in three sets (each set contains one CD).

```
V2.0
RPersonal Jukebox
Sblues
DVarious/Evidence Blues Sampler
TEDDY CLEARWATER: Blues Hang Out
B1 0.0 38.88375
I0.0.23580 16
TLUTHER JOHNSON: Lonesome In My Bedroom
```



```
B1 38.88375 86.37233
I0.1.29017 16
TMAGIC SLIM: Help Me
B1 86.37233 114.109068
I0.2.17405 16
TKOKO TAYLOR: Big Boss Man
B1 114.109068 151.117790
I0.3.22593 16
TBIG JOE TURNER: Cherry Red
B1 151.117790 186.2888
I0.4.20795 16
TLOUIS JORDAN: Saturday Night Fish Fry
B1 186.2888 215.26194
I0.5.17785 16
TJOHN LEE HOOKER: Boogie Chillen
B1 215.26194 235.61622
I0.6.12357 16
TBUDDY GUY / JR. WELLS: Come On In This House
B1 235.61622 263.105872
I0.7.17275 16
TOTIS RUSH: Looking Back
B1 263.105872 303.101079
I0.8.24358 16
TLUTHER ALLISON: Key To The Highway
B1 303.101079 342.64567
I0.9.23602 16
TSCREAMIN’ JAY HAWKINS: I Put A Spell On You
B1 342.64567 371.28941
I0.10.17508 16
TJ. B. HUTTO: I Feel So Good
B1 371.28941 406.127616
I0.11.21797 16
TLUTHER JOHNSON, JR.: Sweet Home Chicago
B1 406.127616 441.109264
I0.12.21248 16
TLONNIE BROOKS: Reconsider Baby
B1 441.109264 465.59660
I0.13.14395 16
TPINETOP PERKINS: Pinetop Is Just Top
B1 465.59660 510.130032
I0.14.27758 16
Sclassical
DVarious Artists/RCA Victor Greatest Hits Sampler
TBach: Brandenburg Concerto No 2
B1 511.0 532.123860
I1.0.13385 16
TMozart: Eine lkeine Nachtmusik
B1 532.123860 576.26928
I1.1.26365 16
TBeethoven: Turkish March
B1 576.26928 588.96585
I1.2.7640 16
TGrieg: In the Hall of the Mountain King
B1 588.96585 609.126116
I1.3.12940 16
TGershwin: Strike Up The Band
B1 609.126116 630.115104
```



```
I1.4.12748 16
TRimsky-Korsakoff: Flight of the Bumblebee
B1 630.115104 641.37686
I1.5.6342 16
TTchaikovsky: Waltz from Serenade for Strings
B1 641.37686 670.59738
I1.6.17780 16
TWagner: Ride of the Walkyries
B1 670.59738 711.8922
I1.7.24753 16
TBach: Air on the G String
B1 711.8922 740.125014
I1.8.18222 16
TBizet: Farandole
B1 740.125014 762.72578
I1.9.13163 16
TVerdi: Triumphal March from Aida
B1 762.72578 789.71317
I1.10.16452 16
TBrahms: Hungarian Dance No. 5
B1 789.71317 807.34532
I1.11.10800 16
TTchaikovsky: 1812 Overture Conclusioon
B1 807.34532 831.130032
I1.12.15076 16
Sjazz
DVarious Artists/Priceless Jazz Sampler 4
TLouis Armstrong / Love Walked In
B1 832.0 850.55872
I2.0.11237 16
TDave Grusin / Mountain Dance
B1 850.55872 896.121852
I2.1.28348 16
TDiane Schur / A Time for Love
B1 896.121852 939.31237
I2.2.25785 16
TBrecker Brothers / And Then She Wept
B1 939.31237 975.79293
I2.3.22167 16
TCarmen McRae / You Took Advantage of Me
B1 975.79293 995.17337
I2.4.11900 16
TFreddie Hubbard / Caravan
B1 995.17337 1050.51131
I2.5.33683 16
TElla Fitzgerald / My Heart Belongs to Daddy
B1 1050.51131 1069.115028
I2.6.11882 16
TMcCoy Tyner / Autumn Leaves
B1 1069.115028 1115.60217
I2.7.27780 16
TJohnny Hartman / It Never Entered My Mind
B1 1115.60217 1142.64808
I2.8.16480 16
TStanley Turrentine & Shirley Scott / Let it Go
B1 1142.64808 1186.26391
I2.9.26638 16
```



```
      TMilt Jackson / I Love You
      B1 1186.26391 1220.71197
      I2.10.20935 16
      TBen Webster / Our Love is Here to Stay
      B1 1220.71197 1240.130032
      I2.11.12466 16
      Ud0103c0f+15+150+23730+52747+70152+92745+113540+131325+143682+160957+18
      5315+208917+226425+248222+269470+283865+4158
      Uac0a360d+13+150+13535+39900+47540+60480+73228+79570+97350+122103+14032
      5+153488+169940+180740+2616
      Uac0cfe0c+12+150+11387+39735+65520+87687+99587+133270+145152+172932+189
      412+216050+236985+3328
      .
      +0.1241.-1
.
```

4.4 Order of records

The TOC should be constructed as follows:

- A Version record, specifying version 2.0
- The root record, containing the PJB's name
- A *Set* record, identifying the name of a set.
  - A *Disc* record, identifying the name of a Disc
    - A *Track* record, identifying the name of a track
    - A *Track Address* record, specifying the location of the music on the disk
    - A *Track ID* record, specifying the ID information for the track within the CD.
    - More *Track, Track Address,* and *Track ID* records, in that order, for each track on the disc.
  - Another *Disc* record, for the next disc.
    - *Track, Track Address,* and *Track ID* records, in that order, for each track on this disc.
  - … etc, for each disc in the set
- Another *Set* record, for the next set.
  - *Disc* records, and within them, the *Track, Track Address,* and *Track ID* records
- … etc, for each set in the PJB
- *UID* records, one for each *Track ID* record that refers to one. The index field of the *Track ID* record specifies which *UID* record the track is referring to.
- An EOF record
- One or more *Allocation Map* records, to describe the allocated space on the disk.
- An EOF record.

4.5 Duplicating tracks in the TOC

To duplicate a track, simply add the *Track*, *Track Address*, and *Track ID* records to another disc. Similarly, to duplicate an entire disc, duplicate all of the *Track*, *Track Address*, and *Track ID* records into another disc.

The Jukebox Manager code is responsible for counting references to allocated blocks to prevent space from being reclaimed.

5. File System Layout

5.1 Allocation Blocks

The PJB's file system is a simple linked list of blocks. Each block consists of a 1KB header, followed by 127KB of music data, and a short tail record. The total allocation unit size is 128Kbytes, broken into 128 one-kilobyte "clicks" to make USB transfers easier.

A "C" declaration for the allocation block would look something like this:

```
typedef struct fs_blkaddr {
    u8 addr[3];
} FS_BLKADDR;

typedef struct fs_allocblock {
      union {
          struct {
            FS_BLKADDR h_next;            /* Next block in chain */
            FS_BLKADDR h_prev;            /* Previous block in chain */
            u8 userdata[1020];            /* Data not used by player */
            FS_BLKADDR h_next2;           /* Copy of "next block in chain" */
            FS_BLKADDR h_prev2;          /* Copy of "prevblock in chain" */
          } h;
          u8 head[1032];
      } h;
        u8 body[130032];                      /* Body (mp3 data) */
      union {
          struct {
            FS_BLKADDR t_next;            /* next block in chain */
            u8 t_chksum[4];               /* checksum of block */
            u8 t_unused;
          } t;
          u8 tail[8];
      } t;
} FS_ALLOCBLOCK;
```

Because the usual reader of this information is the 24-bit DSP in the PJB, careful attention must be made to keep integer fields on 3-byte boundaries. Block addresses (the FS_BLKADDR structures) are 24-bit big-endian numbers so that the DSP can read them without additional effort.

The following fields are used by the DSP to read the disk. They must be correct for the firmware to functon normally:

Symbol	Offset	Size	Description

AB_NextLink	0	3	The allocation block number of the next block in the chain. –1 indicates the end of the chain.
AB_PrevLink	3	3	The allocation block number of the previous block in the chain. –1 indicates the beginning of the chain.
AB_NextLink1	1026	3	A duplicate of AB_NextLink. This appears in a different disk sector, so it should help with error recovery.
AB_PrevLink1	1029	3	A duplicate of AB_PrevLink
AB_AltNextLink	131064	3	A duplicate of AB_NextLink, at the end of the block
AB_CRC	131068	4	32-bit CRC of the block
AB_PayloadStart	1032	130032	The actual MP3 payload.

The "userdata" field (1020 bytes in this structure) is used by the Jukebox Manager to store additional information in each block to identify what music it came from. This can be useful should it ever be necessary to write a filesystem checker for the PJB. This information is not used by the firmware. All numbers are stored in big-endian format.

Symbol	**Offset**	**Size**	**Description**
AB_HdrVersion	100	1	The version number of this header record. Currently '2'
AB_Origin	101	1	Origin of the music in this block. Specify 0 for a captured audio CD, 1 for an ID3 tagged MP3 file, or 2 for an untagged MP3 file
AB_TrackNumber	103	1	Track number from the original CD that this block came from
AB_TrackOffset	104	4	Offset into the current track (in bytes), or –1 if starting a new track
AB_BlockInTrack	108	4	The offset (in allocation units) of this block within the current track. Starts at zero, and is incremented for each block allocated to this track
AB_Encoding	112	1	Encoding type. Specify 1 for MP3.
AB_BitRate	113	3	Bitrate (128000 for 128kbps).
AB_CDID	160	N	CDDB ID for the CD that this track came from, if it was extracted directly from a CD.
AB_ID3_TAG	160	128	ID3V2 tag (the 128-byte variety) for this track, if it came from an MP3 file on the PC

5.2 Allocation Map

The *Allocation Map* describes the allocated space on the PJB's hard disk. It does not track what blocks are assigned to each track. The allocation map is used when new space is requested.

To construct the map, a bit-vector is initialized to zeroes. Each *Allocation Map* record is read from the TOC and the runs of sectors are used to mark the space allocated. The resulting bitmap can be used to choose which sectors are free for allocation when new music is added to the Jukebox.

For details on the allocation map, consult the example source code.

5.3 Representing an entire CD

One feature of the Personal Jukebox is its ability to store an entire CD as a single MP3 bitstream, while still allowing individual access to each track.

The way this is accomplished is to store the CD as a continuous chain of blocks, but create *Track* entries for each track, pointing into the allocated chain of blocks.

6. Sample Code

The sample code in this kit can be very useful for programmers wishing to write their own Jukebox Manager applications. It is not a complete Jukebox Manager, but it includes enough of the key components to build a Jukebox Manager for the open source community.

The example code includes sources to the software for implementing the USB protocol, file system, some test programs, and a command-line application to add MP3 files to a PJB.

There is no warranty for these sources. Use them at your own risk!

6.1 GNU Copyright

The source files in the example source tree are distributed under the GNU Public License (see http://www.gnu.org). This copyright notice should have been included in each source file.

```
**********************************************************************
** Copyright (C) 2000 by Compaq Computer Corporation                **
**                                                                  **
** This program is free software; you can redistribute it and/or    **
** modify it under the terms of the GNU General Public License      **
** as published by the Free Software Foundation; either version 2   **
** of the License, or (at your option) any later version.           **
**                                                                  **
** This program is distributed in the hope that it will be useful,  **
** but WITHOUT ANY WARRANTY; without even the implied warranty of   **
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the    **
** GNU General Public License for more details.                     **
**                                                                  **
** You should have received a copy of the GNU General Public License **
** along with this program; if not, write to the Free Software      **
** Foundation, Inc., 59 Temple Place - Suite 330,                   **
** Boston, MA  02111-1307, USA.                                     **
**                                                                  **
**********************************************************************
```

6.2 Example Source Directory Tree

Directory	Contents
`include/`	include files and other headers
`pjbtest/`	A test program for testing the USB interface and USB command packets. It also contains example code for updating the PJB's flash.

`pjbapi/`	The "client" code for the PJB's USB command interface
`pjbtoc/`	The PJB's file system and high-level calls for manipulating data on the PJB
`pjbcmds/`	A simple command-line test program to demonstrate adding and removing MP3s from a PJB.
`docs/`	The location of this file.
`usbdrv/`	The USB kernel driver source for Linux. This requires a very recent kernel, which you can obtain from http://www.kernel.org

6.3 Building the example sources

Note: If you plan to write your own Jukebox Manager, you should have at least firmware version 2.1.2. This firmware version contains code to better handle corrupted or invalid TOC files, so the PJB will be able to start with a bad TOC and operate well enough to have a good TOC installed.

To build these sources under Linux, you can use the "makeall" shell script at the top of the tree.

To build these sources under Windows, you can use the "makeall.bat" command file at the top of the tree. For Windows, you will need the Windows98 DDK installed to access the USB driver include files.

LAST PAGE OF DOCUMENT

