PC CARD STANDARD

Volume 10

Media Storage Formats Specification

Revision History

Date	Media Storage Formats Specification Version	PC Card Standard Release	Revisions
02/95	5.0	February 1995 (5.0) Release	Initial release MS-DOS BPB/FAT Format Linear File Store
03/95	N/A	March 1995 (5.01) Update	None
05/95	N/A	May 1995 (5.02) Update	None
11/95	N/A	November 1995 (5.1) Update	None
05/96	5.2	May 1996 (5.2) Update	Added Flash Translation Layer (FTL)
03/97	6.0	6.0 Release	None
04/98	6.1	6.1 Update	Corrections to Flash Translation Layer (FTL)
02/99	7.0	7.0 Release	None
03/00	7.1	7.1 Update	None
11/00	7.2	7.2 Update	None
04/01	8.0	8.0 Release	Changed to PC Card Standard Volume Number 10 (was Volume Number 7)

©2001 PCMCIA/JEITA

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording or otherwise, without prior written permission of PCMCIA and JEITA. Published in the United States of America.

CONTENTS

1. Introduction ____ 1

1.1 Purpose 1

1.2 Scope 1

1.3 Related Documents 1

2. Overview ____ 3

2.1 Storing Data 3

2.2 Partitions 3

2.3 Files and File Systems 3

2.4 Storage Media 3

2.5 In Summary 4

3. Partitions ____ 5

3.1 Card Information Structure (CIS) Partitioning 6

3.1.1 Overview 6

3.1.2 Partition Operations 6

3.1.2.1 Creation 6

3.1.2.2 Deletion 6

3.1.2.3 Extension 6

3.1.2.4 Evaluation Order 7

3.1.3 Initial Program Load 7

3.1.4 Data Structures 7

3.2 Master Boot Record (MBR) 8

3.2.1 Overview 8

3.2.2 Partition Operations 8

3.2.2.1 Creation 8

3.2.2.2 Deletion 9

3.2.2.3 Extension 9

3.2.2.4 Evaluation Order 9

3.2.3 Initial Program Load 10

3.2.4 Data Structures 11

3.2.4.1 Master Boot Record 11

3.2.4.2 Partition Entry 11

4. File Formats ____ 13

4.1 MS-DOS BPB/FAT Format 13

4.1.1 Overview 13

4.1.2 Versions or Revisions ... 14
4.1.2.1 12-Bit FATs ... 14
4.1.2.2 16-Bit FATs ... 14
4.1.2.3 Huge Partitions ... 14
4.1.3 Partition Recognition ... 14
4.1.4 Partition Formatting ... 15
4.1.5 File Operations ... 15
4.1.5.1 File Creation ... 15
4.1.5.2 File Deletion ... 15
4.1.5.3 Read and Write Operations ... 15
4.1.6 Initial Program Load ... 16
4.1.7 Data Structures ... 17
4.1.7.1 Partition Boot Record (PBR) ... 17
4.1.7.2 BIOS Parameter Block (BPB) ... 18
4.1.7.3 Directory Entry ... 19
4.2 Linear File Store ... 20
4.2.1 Overview ... 20
5. Translation Layers ... 23
5.1 Virtual Block Device Flash Translation Layer - FTL ... 24
5.1.1 Versions or Revisions ... 24
5.1.2 Overview ... 24
5.1.2.1 Emulating Traditional Block Devices - Virtual Block Device ... 24
5.1.2.2 Flash Characteristics ... 24
5.1.2.3 Erase Zones and Erase Units ... 26
5.1.2.4 Erase Unit Header and Block Allocation Information ... 27
5.1.2.5 Block Allocation Information and the Block Allocation Map (BAM) ... 28
5.1.2.6 Virtual Block Map - Mapping Virtual Blocks to Logical Addresses ... 30
5.1.2.7 Virtual Page Map - Locating the Pages of the Virtual Block Map ... 32
5.1.2.8 Replacement Pages ... 33
5.1.2.9 Mapping Logical Addresses to Physical Addresses ... 34
5.1.3 Data Structures ... 34
5.1.3.1 Erase Unit Header (EUH) ... 34
5.1.3.2 Flags ... 37
5.1.4 Partition Recognition ... 38
5.1.5 Partition Formatting ... 39
5.1.6 Logical Block Operations ... 39
5.1.6.1 Read ... 39
5.1.6.2 Write ... 40
5.1.6.3 Unit Recovery ... 40
5.1.7 Initial Program Load ... 41
6. Storage Devices ... 43
6.1 Static RAM Cards ... 43

© 2001 PCMCIA/JEITA

6.2 Flash Memory Cards 43
6.3 PC Card ATA Drives 43

© 2001 PCMCIA/JEITA

FIGURES

Figure 5-1: Erase Zones....26
Figure 5-2: Read/Write Blocks....27
Figure 5-3: Erase Unit Layout....28
Figure 5-4: Block Allocation Map....30
Figure 5-5: Virtual Block Map....32
Figure 5-6: Page Mapping....33

© 2001 PCMCIA/JEITA

1. INTRODUCTION

1.1 Purpose

This document describes how data is formatted on PC Cards used as storage devices to promote the exchange of these cards among different host systems. These include memory cards using various types of volatile and non-volatile devices and ATA disk drives, both silicon and rotating media. Each of these storage technologies have unique characteristics which may benefit from different storage techniques and handling. This has resulted in the development of different storage formats and/or partitioning for PC Cards using these devices.

NOTE: The inclusion of a partition, file format, or media type information in this document does not constitute an endorsement by PCMCIA or JEITA. PCMCIA and JEITA are only acknowledging this information has been used to record data on a PC Card and, in some cases, that PCMCIA/JEITA members have agreed that using the documented implementation may reduce problems encountered when attempting to interchange data between host systems.

1.2 Scope

This document is intended to provide enough information to allow software developers to use data stored on PC Cards by other host systems using potentially different operating and file systems. Unless required to understand the data structures used on the PC Card, algorithms for updating the data on the PC Card are not specified, only the storage format.

1.3 Related Documents

The following documents which comprise the ***PC Card Standard***:

PC Card Standard Release 8.0 (April 2001), PCMCIA /JEITA

Volume 1. ***Overview and Glossary***
Volume 2. ***Electrical Specification***
Volume 3. ***Physical Specification***
Volume 4. ***Metaformat Specification***
Volume 5. ***Card Services Specification***
Volume 6. ***Socket Services Specification***
Volume 7. ***PC Card ATA Specification***
Volume 8. ***PC Card Host Systems Specification***
Volume 9. ***Guidelines***
Volume 10. ***Media Storage Formats Specification***
Volume 11. ***XIP Specification***

Microsoft Corporation, ***Microsoft MS-DOS Programmer's Reference, 2nd edition: version 6.0***, 1993, Microsoft Press.

© 2001 PCMCIA/JEITA

2. OVERVIEW

2.1 Storing Data

Most computer programs need to store and retrieve information. While running, a program may use system memory for limited amounts of information, but often more information is required than may fit in memory. In addition, when a program terminates, the system memory it was using is re-used by other programs and information may be overwritten and lost.

To accommodate large and long-term storage needs, information is stored on disks and other types of external media. Several types of PC Cards are used to store information including static RAM (SRAM) and flash memory cards and silicon and rotating media PC Card ATA drives.

2.2 Partitions

Storage media may be divided into separately addressable areas known as partitions. Each partition is addressed by a file system as if it were a separate storage device with its own directory and allocation information.

To minimize the number of storage formats a file system needs to be able to recognize to avoid erroneously assuming a partition or PC Card is unformatted, the ***Card Services Specification*** provides two services which return partition information. They are **GetFirstPartition** and **GetNextPartition**. A Card Services implementation determines partition information using the CIS or by performing searches for file system specific data structures.

2.3 Files and File Systems

Information is usually stored in units called files. Files are managed by a file system, usually a part of or an extension to an operating system. The file system records the structure used to store files in a system file or area known as a directory. Each file typically has a record in the directory known as a directory entry.

A file system also keeps track of where files are stored on the media and what areas are available. A file system keeps track of storage space in units known as blocks or clusters. Block devices are usually limited to reading or writing information in units known as sectors. A block or cluster may be one sector or may be two or more contiguous sectors. By making blocks or clusters more than one sector, a file system reduces the amount of storage required to track whether or not space on the media is allocated.

2.4 Storage Media

PC Cards provide several technologies for storing information, each with its own unique requirements. Static RAM (SRAM) devices are the most flexible allowing any byte to be separately read or written.

Flash memory devices may allow byte or block accesses for reads, but require special write algorithms and pre-erased bytes before most write operations. In addition, flash device require erase operations to be performed on a block of contiguous bytes as a single unit.

PC Card ATA drives are accessed in the same manner as traditional block devices requiring an entire sector be read or written at one time. The actual storage media may be silicon or magnetic oxide on a rotating disk.

2.5 In Summary

To access data on storage media requires a complete understanding of how the media is partitioned, the file format used and whether a translation layer is used. Client device drivers first check the Card Information Structure (CIS) to determine if a PC Card is partitioned. If the card is partitioned, the tuples used to describe the partition also describe the storage format used on the media.

 © 2001 PCMCIA/JEITA

3. PARTITIONS

PCMCIA/JEITA recognize two methods for partitioning PC Cards. First, linear memory PC Cards such as flash and S-RAM cards use tuples in the Card Information Structure (CIS) to describe how a PC Card is partitioned. Second, PC Card ATA drives are partitioned using a Master Boot Record with a partition table.

Each recognized partitioning method is described separately in the following sections. The following information is provided about each recognized method:

Overview	An overview of the partitioning method and where it is used.
Partition Operations	How partition operations are performed.
Initial Program Load	How a host system boots using the partitioning method.
Data Structures	The data structures used by the partitioning method.

All PC Cards used for data storage must provide partition information as described in this section. PC Cards used for data storage that do not contain partition information described in this section may be assumed to be unformatted.

3.1 Card Information Structure (CIS) Partitioning

3.1.1 Overview

The PC Card's CIS describes partitions using the following tuples:

Tuple Name	Tuple Constant	Tuple Value
Format	CISTPL_FORMAT	41H
Organization	CISTPL_ORG	46H

Both tuples must be present. The Format Tuple describes where that partition is located on the media and the Organization Tuple identifies the data storage format used within the partition. Data storage within a partition is also affected by the following tuples, if they are present:

Tuple Name	Tuple Constant	Tuple Value
Geometry	CISTPL_GEOMETRY	42H
Byte-Order	CISTPL_BYTEORDER	43H
Software Interleave	CISTPL_SWIL	23H

3.1.2 Partition Operations

3.1.2.1 Creation

A partition is created by adding the tuples described above to a PC Card's Card Information Structure (CIS). The ability to write to a PC Card's CIS is dependent on the card and potentially installed device drivers. A PC Card may require that the entire CIS be erased and then re-written to modify the CIS.

Some PC Cards use multiple tuple chains to describe physical characteristics of the card separately from how the card is used. For example, a primary tuple chain in a PC Card's attribute memory space might describe the physical characteristics of the card, such as the type and size of the memory device used on the card. A secondary tuple chain, in the PC Card's common memory space, might be used to describe the logical characteristics of the card, such as the partitioning. In this manner, the PC Card might be manufactured with the physical information hard-coded into read-only memory in attribute memory space and logical partitioning information would be added by using writable memory in common memory space. (See the ***Metaformat Specification*** for more information about how tuple chains are linked together within the CIS.)

3.1.2.2 Deletion

To delete a partition, all of the tuples describing the partition must be deleted from the Card Information Structure (CIS). Depending on the PC Card, the CIS may have to be erased and then re-written without the tuples that describe the partition.

3.1.2.3 Extension

Some PC Cards allow the Card Information Structure to be extended without erasing existing tuples. These cards permit additional partitions to be defined by adding tuples to the end of the last tuple chain on the PC Card.

 © 2001 PCMCIA/JEITA

3.1.2.4 Evaluation Order

Host software evaluates partition information as it is encountered in the Card Information Structure (CIS). If host software recognizes a partition type, the next available drive specifier is assigned to the partition. If there are multiple partitions of different types on a PC Card, each partition type may be recognized by a separate host device driver. For this reason, the order host drive specifiers are assigned to partitions is host system specific.

3.1.3 Initial Program Load

The PC Card Standard does not currently define a method for booting from a PC Card using partition definitions in the CIS.

3.1.4 Data Structures

See the ***Metaformat Specification*** for a complete definition of the tuples used to describe partitions.

© 2001 PCMCIA/JEITA

3.2 Master Boot Record (MBR)

3.2.1 Overview

PC Card ATA drives are partitioned using a Master Boot Record (MBR) with a Partition Table in the first physical sector of the media.. Partition Table Entries describe the size, location and type of data within a partition. A Partition Table Entry may also describe an Extended Partition which is further divided into one or more partitions.

The MBR contains a word of 55AAH at offset 1FEH. The sector contains operating system independent code to perform Initial Program Load on x86 architecture host systems. For system and PC Card interoperability, all systems, including those that do not use the IPL code for booting, must include such information when formatting the MBR. The MBR also contains a Partition Table with four (4) Partition Table Entries at offset 1BEH. When booting from a device with an MBR on an x86 architecture system, code within the MBR evaluates the Partition Table for a partition marked as the default boot partition. Only one partition may be marked as the default boot partition.

If the x86 bootstrap code locates a default boot partition in the MBR's Partition Table, the Partition Boot Record (PBR), the first sector of the partition, is loaded into memory. If the word at offset 1FEH of the PBR is 55AAH, control is passed to the next stage bootstrap loader in the PBR image in memory.

During operating system initialization, MBRs on all fixed disk devices are evaluated by the file system for partition definitions in reverse order starting with the entry at offset 1EEH. The host system assigns unit designations (drive letters under MS-DOS), to each partition matching a type supported by the file system.

3.2.2 Partition Operations

3.2.2.1 Creation

A partition is created by setting the fields of a partition entry in the partition table of the Master Boot Record (MBR) to describe the desired partition. A partitioning utility first reads the MBR into host system memory. If the word at offset 1FEH of the MBR is not 55AAH, the device is not formatted and the utility must create an initial MBR with an empty partition table before preceding.

If the word at offset 1FEH of the MBR is 55AAH, the partitioning utility searches the partition table for an empty entry. An entry is considered empty if the NumSectors field is zero (0). If there are no empty entries in the partition table, a partition cannot be created.

If there is an empty partition entry, the partitioning utility creates a new partition using contiguous unallocated space on the media. The utility determines if there is any available space on the media by subtracting the space allocated to other partitions from the total size of the media. The total size of the media is determined in a host system dependent manner. For example, on x86 systems with a PC-compatible ROM BIOS the Get Drive Parameters function (Int 13H Function 8) is typically used.

If there is unallocated space on the media, the partitioning utility must also determine where the space is located by comparing the Start and End of each allocated partition. How a partitioning utility decides which space to use when multiple unallocated spaces are available is implementation specific.

After a partition entry is created the partitioning utility needs to notify the host system of the change to the MBR.

© 2001 PCMCIA/JEITA

A partition table entry uses Cylinder, Head and Sector (CHS) addressing to describe the starting and ending boundaries of a partition. Some PC Card ATA drives translate their physical CHS information to logical values that are compliant with limits imposed by some host systems that are unable to address cylinder, head or sector values that exceed system-specific limits. Once a partition table entry has been created, all subsequent accesses to the media must use the same logical CHS addressing.

The PC Card Standard requires that all partitions described in the partition table within the MBR end on a logical cylinder boundary based on the logical CHS addressing in use when the first partition was created. This allows a host system to validate the logical CHS addressing in use is correct by confirming the maximum head and sector values used for media access are the same as those used to indicate the ending head or sector of all partitions on the media.

3.2.2.2 Deletion

A partition is deleted by resetting all of the fields of a partition entry in the partition table of the Master Boot Record to zero (0). After a partition entry is deleted the partitioning utility needs to notify the host system of the change to the MBR.

3.2.2.3 Extension

Some partition types extend the partition table in a system-specific manner. For example, MS-DOS defines a special partition type called the Extended MS-DOS partition. The space allocated to an Extended MS-DOS partition is sub-divided into logical drives. The first sector of an Extended MS-DOS Partition contains a partition table formatted in the same manner as the partition table in a Master Boot Record (MBR). The extended partition table typically contains two entries, an MS-DOS partition and another Extended MS-DOS partition entry.

The MS-DOS partition entry in an extended MS-DOS partition describes a logical drive. If an Extended MS-DOS partition entry is also present in the partition table, another potential logical drive may exist within the area described by the Extended MS-DOS partition entry. Extended MS-DOS partition entries create a forward-linked list of logical drives within the Extended MS-DOS partition in the MBR.

The one difference between partition entries in an MBR and partition entries in the partition table in an Extended MS-DOS partition is the StartSector field of the partition entries. In the MBR this field is relative to the beginning of the media. In an Extended MS-DOS partition this field is relative to the beginning of the Extended MS-DOS partition described in the MBR.

3.2.2.4 Evaluation Order

The order partition entries are evaluated in the partition table of the Master Boot Record (MBR) is dependent on the operating system. For example, MS-DOS evaluates primary partition types on the first two physical fixed drives on x86 systems addressed by the ROM BIOS Int 13H Disk I/O handler as drives 80H and 81H. Primary partition types are 01H, 04H and 06H.

If the first physical fixed drive has a primary partition, MS-DOS assigns the next available logical drive letter to the partition. If there is a second physical fixed drive and it has a primary partition, MS-DOS assign the next available logical drive letter to this partition. After MS-DOS assigns the primary partition types on the first two physical fixed drives as logical drives, Extended MS-DOS partitions are evaluated.

If the first physical fixed drive has an Extended MS-DOS partition, each logical drive described in the chain of Extended MS-DOS partitions is added as a logical drive letter. If there is a second physical

fixed drive and it has an Extended MS-DOS partition, each logical drive described in the chain of Extended MS-DOS partitions is added as a logical drive letter.

3.2.3 Initial Program Load

The PC Card Standard does not describe a system independent method for booting from a device with a Master Boot Record (MBR). However, x86 systems use the MBR as the first stage of a multi-stage program loader. The host system reads the MBR of the first physical drive into host system memory at 0000H:7C00H. If the word at offset 1FEH of the MBR is not 55AAH, the media is not bootable and the system continues the boot process with another device or displays an error message.

If the word at offset 1FEH of the MBR is 55AAH, the host system transfers control to the code at 0000H:7C00H. No arguments are provided to the code by the host system. No stack is established and no indication of which device the MBR was read from is provided.

The boot code in the MBR evaluates the partition table from the last entry at offset 1EEH to the first entry at 1BEH. If an entry is found with the default x86 boot partition field set to 80H, the first sector of the partition described by the partition entry, known as the Partition Boot Record (PBR), is loaded into host system memory. If the word at offset 1FEH of the PBR is 55AAH, control is transferred to offset zero of the PBR and the boot process continues. If the word at offset 1FEH of the PBR is not 55AAH, the code in the MBR displays an error message and halts.

 © 2001 PCMCIA/JEITA

3.2.4 Data Structures

3.2.4.1 Master Boot Record

The Master Boot Record contains the following fields:

Offset	Size (Bytes)	Description
000H	446	Boot code
1BEH	16	Partition Entry (See below)
1CEH	16	Partition Entry (See below)
1DEH	16	Partition Entry (See below)
1EEH	16	Partition Entry (See below)
1FEH	2	Signature Word (0x55AA)

3.2.4.2 Partition Entry

Each of the four Partition Entries in the Master Boot Record have the following format:

Offset	Size (Bytes)	Description
00H	1	x86 default boot partition
		00H = Not default boot partition
		80H = Default boot partition
01H	1	StartHead - Zero-based (0) head number where partition starts on media.
02H	1	StartSector - Bits 0 .. 5 are one-based (1) sector number where partition starts on media. Bits 6 and 7 are high bits of zero-based (0) cylinder number where partition starts on media.
03H	1	StartCylinder - Least significant eight bits of zero-based (0) cylinder number where partition starts on media. Upper two bits of starting cylinder number are in StartSector field.
04H	1	Partition Type
		00H: Unknown or deleted if NumSectors is zero
		01H: MS-DOS 12-bit BPB/FAT < 16 MB
		04H: MS-DOS 16-bit BPB/FAT < 32 MB
		05H: Extended MS-DOS Partition
		06H: MS-DOS 16-bit BPB/FAT >= 32 MB
05H	1	EndHead - Zero-based (0) head number where partition ends on media.
06H	1	EndSector - Bits 0 .. 5 are one-based (1) sector number where partition ends on media. Bits 6 and 7 are high bits of zero-based (0) cylinder number where partition ends on media.
07H	1	EndCylinder - Least significant eight bits of zero-based (0) cylinder number where partition ends on media. Upper two bits of ending cylinder number are in StartSector field.
08H	4	StartSector (relative to beginning of media or Extended MS-DOS Partition)
0CH	4	NumSectors

© 2001 PCMCIA/JEITA

4. FILE FORMATS

To achieve backward compatibility and speed acceptance of PC Card technology, the formats used by file systems for traditional storage devices are often used for PC Cards. This section describes the data structures used by specific formats and how file operations using those structures are performed. Each subsection provides the following information about a file format:

Overview	An overview of the format and where it is used.
Versions or Revisions	Variations of the file format in common use.
File Operations	How common file operations are performed using the defined data structures.
Initial Program Load	How a host system boots using the file format.
Data Structures	The data structures used by the file format.

4.1 MS-DOS BPB/FAT Format

4.1.1 Overview

In the MS-DOS environment, the native file system stores information in what is known as the BPB/FAT format. Static RAM (SRAM) cards and PC Card ATA drives commonly use this format on x86 architecture systems. Flash memory cards may also use this format if they are read-only or are accessed using a Flash Translation Layer (FTL).

BPB/FAT media is accessed in blocks known as sectors. The number of bytes in a sector is described in the BIOS Parameter Block (BPB) **BytesPerSector** field. The total number of sectors on the media is described by the BPB **0** field if there are less than 65,536 sectors. If there are more than 65,535 sectors, the **TotalSectors** field is zero (0) and the number of sectors on the media is described by the **HugeSectors** field.

BPB/FAT media may have hidden sectors. Hidden sectors are usually associated with partitioned media. The BPB **HiddenSectors** field describes the number of hidden sectors on the media.

BPB/FAT media have one or more reserved sectors. The BPB **ReservedSectors** field describes the number of reserved sectors on the media. The first reserved sector is known as the Boot Sector. This sector contains the BIOS Parameter Block (BPB) describing the size and format of the media.

After any hidden and reserved sectors, BPB/FAT media have one or more File Allocation Tables (FATs). If there is more than one FAT, the additional copies are maintained to allow data recovery in case the first FAT is damaged. Most BPB/FAT media is formatted with two (2) FATs. The number of FATs is described by the BPB **NumFATs** field.

The FAT tracks the allocation of the media's data space. Space on BPB/FAT media is allocated in units of one or more contiguous sectors called clusters. If a cluster has more than one sector, the number of sectors in the cluster must be a power of two (for example, two, four, eight, etceteras). The BPB **SectorsPerCluster** field describes the number of sectors per cluster.

FAT entries may be 12 or 16-bits depending on the number of clusters on the media. Media with more than 4085 clusters use 16-bit entries. Media with 4085 or less clusters use 12-bit entries. Each entry in the FAT represents the allocation status of a cluster. The first two entries are reserved for FAT ID information. The first data area tracked by the FAT is cluster number two (2).

© 2001 PCMCIA/JEITA

A FAT entry has the following meanings. The digit in parentheses represents the upper four bits associated with 16-bit FAT entries.

Value	Meaning
(0)000H	Available or unallocated cluster
(0)001H	Reserved, do not use this value
(0)002H - (F)FF6H	Next cluster in file or directory
(F)FF7H	Bad cluster, do not store data in this cluster
(F)FF8H - (F)FFFH	Last cluster of file or directory

The number of sectors used to store each FAT is described in the BPB **NumFATSectors** field. The next area on the media after the FAT(s) is the Root Directory. The Root Directory is an array of Directory Entries. The number of directory entries in the Root Directory is described by the BPB **RootDirEntries** field. The Directory Entry **StartCluster** field describes the first cluster used to store data for the file or directory identified by the entry.

The space on the media immediately following the Root Directory is used for file and subdirectory data storage. The Directory Entry **Attributes** field identify an entry as a subdirectory of a file. Subdirectories are special file entries containing additional directory entries for files and further subdirectories in the data area of the media.

4.1.2 Versions or Revisions

4.1.2.1 12-Bit FATs

BPB/FAT media formatted with less than 4086 clusters uses 12-bit File Allocation Table (FAT) entries. Each FAT entry requires 1.5 bytes.

4.1.2.2 16-Bit FATs

BPB/FAT media formatted with more than 4085 clusters uses 16-bit File Allocation Table (FAT) entries. Each FAT entry requires two (2) bytes.

4.1.2.3 Huge Partitions

BPB/FAT media with more than 65,535 sectors indicates the number of sectors in the BPB **HugeSectors** field and sets the **TotalSectors** field to zero (0). All huge partitions use 16-bit FAT entries requiring two (2) bytes for each entry.

4.1.3 Partition Recognition

All BPB/FAT partitions begin with a Partition Boot Record (PBR). All PBRs start with a byte of E9H or EBH. If the PBR begins with a byte of EBH, the byte at offset two (2) must also be 90H. If the previous bytes are present, the byte at offset 26H of the PBR must be 29H indicating an Extended BPB is present.

If the PBR does not contain the above bytes, the partition is not BPB/FAT format.

Further consistency checks may be performed on an implementation specific basis. The BIOS Parameter Block (BPB) fields may be checked for valid values. For example, the **BytesPerSector** field must be a power of two and at least 128 bytes. The **SectorsPerCluster** field must be one or any other

© 2001 PCMCIA/JEITA

power of two. The **ReservedSectors** field must be at least one (1). The **NumFATs** field must be one (1) or two (2).

4.1.4 Partition Formatting

The first step in formatting a BPB/FAT partition is to write a Partition Boot Record (PBR) with a BIOS Parameter Block (BPB) to the first reserved sector of the partition. The next step is to initialize the number of File Allocation Tables (FATs) indicated by the BPB **NumFATs** field. The first byte of each FAT is the partition's FAT ID byte followed by two bytes of FFH. If the FAT uses 16-bit entries, the next byte is also FFH. All of the remaining bytes of the FAT are zero (00H).

The sectors for the Root Directory are then initialized. All bytes of the Root Directory sectors are written as zeroes (00H). Optionally, a scan of the media may be performed to determine if any of the data clusters are defective. How or if this scan is performed is implementation specific. However, if bad sectors are located, the clusters containing the bad sectors are marked as bad in all FATs.

4.1.5 File Operations

4.1.5.1 File Creation

A file is created by filling in the fields in a directory entry. The directory may be in the Root Directory or a subdirectory. A directory entry may be created with the **StartCluster** field set to zero (0) if the file or subdirectory does not contain any data. The first write to the file or subdirectory will allocate a cluster if there is one available on the media.

4.1.5.2 File Deletion

A file or subdirectory is deleted by overwriting the first byte of the **Name** field with the value E5H. In addition, any clusters allocated to the directory entry must be freed by writing a zero (0) to the corresponding cluster entries in all the File Allocation Tables on the media.

A directory entry for a subdirectory may not be deleted until all entries in the subdirectory are deleted as described above.

4.1.5.3 Read and Write Operations

The first step in performing a read or write is to determine where on the media the operation should begin. Such operations typically specify an offset within the file or directory. The BIOS Parameter Block (BPB) **BytesPerSector** and **SectorsPerCluster** fields are used to determine the relative cluster where the operation is to begin.

The desired offset is divided first by the **BytePerSector** field. The result is further divided by the **SectorsPerCluster** field. The result of the second divide is the relative cluster where the operation begins. The remainder of the second divide is the relative sector within the cluster where the operation begins. If the remainder of the first divide is non-zero, the operation begins in the middle of a sector and must be buffered by host software.

For example, if the media is formatted for 512 bytes per sector and there are two sectors per cluster, a read from offset 1024 of a file would begin at the start of the second cluster. A read from offset 1536 begins at the start of the second sector of the second cluster. A read from offset 1792 begins in the middle of the second sector of the second cluster and requires host system buffering.

© 2001 PCMCIA/JEITA

Once the appropriate relative cluster is determined, the File Allocation Table (FAT) is used to convert the relative cluster to an absolute cluster on the media. The Directory Entry's **StartCluster** field determines the first cluster. The value in that entry of the FAT is the next cluster in the chain. When a FAT entry is FF8H to FFFH, the end of the cluster chain has been reached. Attempts to read past the end of the cluster chain are failed.

If an operation continues beyond the end of a cluster, the location of the next cluster is determined from the contents of the current FAT entry. Write operations that continue past the end of the cluster chain need to extend the chain.

A cluster chain is extended by overwriting the current FAT entry with the number of an unallocated cluster. The unallocated cluster entry in the FAT is then marked as the end of the chain. Attempts to extend cluster chains on full media are failed.

When a directory entry is extended, the **FileSize** field of the entry is increased. The **FileSize** field represents the actual size of the file or directory, in bytes, and may be less than the space allocated. This is due to the fact the file or directory may not fill the last cluster allocated.

4.1.6 Initial Program Load

The Partition Boot Record (PBR), in addition to containing the BIOS Parameter Block (BPB), also contains code to continue Initial Program Load (IPL). Host system software loads the PBR into system memory at 0000H:7C00H. If the word at offset 1FEH of the PBR is 55AAH, the host systems transfers control to the PBR code at 0000H:7C00H. No arguments are provided to the code by the host system. No stack is established and no indication of which device the PBR was read from is provided.

If the PBR contains an MS-DOS bootstrap loader, the information in the BPB within the PBR is used to locate systems files in the Root Directory of the media. If these files are located, they are loaded into host system memory and control is transferred to them to complete IPL.

© 2001 PCMCIA/JEITA

4.1.7 Data Structures

4.1.7.1 Partition Boot Record (PBR)

The Partition Boot Record contains the following fields:

<table>
<tr><th>Offset</th><th>Size (Bytes)</th><th>Description</th></tr>
<tr><td>000H</td><td>3</td><td>JMP instruction to PBR boot code</td></tr>
<tr><td>003H</td><td>8</td><td>OEMName and version</td></tr>
<tr><td>00BH</td><td>25</td><td>BIOS Parameter Block (BPB)</td></tr>
<tr><td>024H</td><td>1</td><td>DriveNumber (00H = Floppy, 80H = Fixed)</td></tr>
<tr><td>025H</td><td>1</td><td>Reserved, do not use</td></tr>
<tr><td>026H</td><td>1</td><td>ExtBootSignature - 29H</td></tr>
<tr><td>027H</td><td>4</td><td>VolumeID or Serial Number</td></tr>
<tr><td>02BH</td><td>11</td><td>VolumeLabel - ASCII characters, padded with blanks if less than eleven (11) characters</td></tr>
<tr><td>036H</td><td>8</td><td>FileSysType - ASCII characters identifying file system type. Padded with blanks if less than eight (8) characters. One of the following values:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>FAT12</td><td>12-bit File Allocation Table (FAT)</td></tr><tr><td>FAT16</td><td>16-bit FAT</td></tr></table></td></tr>
<tr><td>03EH</td><td>448</td><td>Boot code</td></tr>
<tr><td>1FEH</td><td>2</td><td>Signature word - 55AAH</td></tr>
</table>

© 2001 PCMCIA/JEITA

4.1.7.2 BIOS Parameter Block (BPB)

The BIOS Parameter Block (BPB) contains the following fields:

<table>
<tr><th>Offset</th><th>Size (Bytes)</th><th colspan="2">Description</th></tr>
<tr><td>000H</td><td>2</td><td colspan="2">BytesPerSector - Number of bytes per sector</td></tr>
<tr><td>002H</td><td>1</td><td colspan="2">SectorsPerCluster - Number of sectors in a cluster</td></tr>
<tr><td>003H</td><td>2</td><td colspan="2">ReservedSectors - Number of reserved sectors at the beginning of the media. Must be at least one (1) to accommodate the Partition Boot Record (PBR)</td></tr>
<tr><td>005H</td><td>1</td><td colspan="2">NumFATs - Number of File Allocation Tables (FATs) on the media.</td></tr>
<tr><td>006H</td><td>2</td><td colspan="2">RootDirEntries - Number of Root Directory entries</td></tr>
<tr><td>008H</td><td>2</td><td colspan="2">TotalSectors - Number of sectors on media. If media has more than 65,535 sectors, this field is zero and the actual number of sectors is in the HugeSectors field.</td></tr>
<tr><td rowspan="11">00AH</td><td rowspan="11">1</td><td colspan="2">MediaIDByte - Used to quickly identify how the media is formatted</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>F0H</td><td>Various types of media</td></tr>
<tr><td>F8H</td><td>Hard disk, any size</td></tr>
<tr><td>F9H</td><td>720K 3.5" or 1.2M 5.25"</td></tr>
<tr><td>FAH</td><td>320K 5.25"</td></tr>
<tr><td>FBH</td><td>640K 3.5"</td></tr>
<tr><td>FCH</td><td>180K 5.25"</td></tr>
<tr><td>FDH</td><td>360K 5.25"</td></tr>
<tr><td>FEH</td><td>160K 5.25"</td></tr>
<tr><td>FFH</td><td>320K 5.25"</td></tr>
<tr><td>00BH</td><td>2</td><td colspan="2">NumFATSectors - Number of sectors in each FAT</td></tr>
<tr><td>00DH</td><td>2</td><td colspan="2">SectorsPerTrack - Number of sectors on a track</td></tr>
<tr><td>00FH</td><td>2</td><td colspan="2">NumHeads - Number of heads</td></tr>
<tr><td>011H</td><td>4</td><td colspan="2">HiddenSectors - Number of hidden sectors in front of reserved sectors</td></tr>
<tr><td>015H</td><td>4</td><td colspan="2">HugeSectors - Number of sectors on media if TotalSectors is zero (0).</td></tr>
</table>

© 2001 PCMCIA/JEITA

4.1.7.3 Directory Entry

Each directory entry contains the following fields:

<table>
<tr><th>Offset</th><th>Size (Bytes)</th><th colspan="2">Description</th></tr>
<tr><td>000H</td><td>8</td><td colspan="2">Name — File or directory name. If less than eight (8) characters, padded with blanks.</td></tr>
<tr><td>008H</td><td>3</td><td colspan="2">Extension — File or directory extension. If less than three (3) characters, padded with blanks</td></tr>
<tr><td rowspan="8">00BH</td><td rowspan="8">1</td><td colspan="2">Attributes — Bit-mapped field using following values:</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>xxxxxxx1B</td><td>Read-only</td></tr>
<tr><td>xxxxxx1xB</td><td>Hidden</td></tr>
<tr><td>xxxxx1xxB</td><td>System</td></tr>
<tr><td>xxxx1xxxB</td><td>Volume label (root directory only)</td></tr>
<tr><td>xxx1xxxxB</td><td>Subdirectory</td></tr>
<tr><td>xx1xxxxxB</td><td>Archive (new or modified entry)</td></tr>
<tr><td>00CH</td><td>10</td><td colspan="2">Reserved, do not use</td></tr>
<tr><td rowspan="5">016H</td><td rowspan="5">2</td><td colspan="2">Time — Bit-mapped field describing time file or subdirectory created or modified:</td></tr>
<tr><td>Bits</td><td>Meaning</td></tr>
<tr><td>0 .. 4</td><td>Two-second interval (0 .. 29)</td></tr>
<tr><td>5 .. 10</td><td>Minute (0 .. 59)</td></tr>
<tr><td>11 .. 15</td><td>Hour (0 .. 23)</td></tr>
</table>

© 2001 PCMCIA/JEITA

4.2 Linear File Store

4.2.1 Overview

Define a partition type for storing files formatted in a PCMCIA/JEITA-prescribed manner within contiguous sections of PC Card memory. This Linear File Store (LFS) uses a well-defined header record to allow directory functions and file access to be performed across a wide variety of operating environments and host platforms.

Partitions on PC Cards are defined using the Data Recording Format Tuples in Layer 2 of ***Metaformat Specification***. A memory-like format tuple is used to describe the PC Card Standard's LFS partition. (See the ***Metaformat Specification*** for more information about format tuples.)

Byte	7	6	5	4	3	2	1	0
0	TPL_CODE		Format tuple code (CISTPL_FORMAT, 41H) or (CISTPL_FORMAT_A, 47H).					
1	TPL_LINK		Link to next tuple (0BH)					
2	TPLFMT_TYPE		Format type code (TPLFMTTYPE_ MEM, 01H)					
3	TPLFMT_EDC		Error Detection Method (TPLFMTEDC_NONE)					
	RFU	000B			0000B			
4 .. 7	TPLFMT_OFFSET		Byte address of the first data byte in this partition.					
8 .. 11	TPLFMT_NBYTES		Number of data bytes in this partition.					
12	TPLFMT_FLAGS		Various Flags (00H - Flags not used)					
	(Reserved)						0	0

The above tuple definition describes a partition which is formatted as a memory-like region with no error detection and no requirement for direct mapping into a specific area of host system address space. For the example an election has been made to shorten the tuple by omitting optional fields which are not used.

The data organization of the PC Card Standard's LFS partition is defined by an Organization Tuple. (See the ***Metaformat Specification*** for more information about data organization tuples.)

Byte	7	6	5	4	3	2	1	0
0	TPL_CODE		Organization tuple code (CISTPL_ORG, 46H).					
1	TPL_LINK		Link to next tuple (0BH)					
2	TPLORG_TYPE		Data organization code (TPLORGTYPE_FS, 00H)					
3 .. 12	TPLORG_DESC		Linear File Store specified by the PC Card Standard ("LFS100") (Note: Text description of this organization, terminated by 00H)					

Within the partition, each file is preceded by a PC Card Standard-defined entry record. This structure provides enough information for a simple bootstrap loader to select an appropriate file. If a specific environment requires information beyond that defined by the PC Card Standard, the organization responsible for that environment defines any additional fields required immediately after the PC Card Standard-defined fields creating an extended header record.

For XIP files, a header record is also defined to provide information about the length of the root segment of the file, whether the file uses paging and the entry point for execution within the file. Because the header record is part of the file, the data in the record is available during file execution.

© 2001 PCMCIA/JEITA

The PC Card Standard-defined entry record is located at the base of the window that maps the root section into host system address space.

Note:

XIP definition to be developed by XIP Working Group.

Files are stored consecutively within the partition using a forward-linked list managed by a field which points to the next header record. If the file is the last in a partition, the pointer to the next header record is set to all ones (-1 or FFFFFFFFH). This value allows files stored in partitions using flash memory devices to be extended if sufficient space remains within the partition.

Each entry in the PC Card Standard's LFS Partition has the following structure:

```
typedef struct {
      DWORD    NextEntry;
      DWORD    EntrySize;
      DWORD    EntryType;
      DWORD    OffsetToHeader;
      DWORD    Flags;
      DWORD    OffsetToString;
      DWORD    UniqueID;
      BYTE     Reserved[4];
      BYTE     EntryInfo[];
} ENTRY;
```

Member	Description
NextEntry	This field points to the next entry within the partition, relative to the start of this entry on the PC Card. The actual location of the next entry is determined by adding the value in this field to the offset of this field on the PC Card. If all of the bits in this field match Bit D0 of the *Flags* field, this is the last entry in the partition.
EntrySize	This field contains the actual size of the entry, i.e. the number of bytes in this instance of the ENTRY structure. *EntrySize* is less than the *NextEntry* when padding is added to align the next entry on a boundary.
EntryType	This field contains a PCMCIA assigned value that defines the interpretation of the data in the *EntryInfo* byte array. *EntryType* values are assigned by PCMCIA. The definition of any header structure within the *EntryInfo* area for a particular *EntryType* value must be provided at the time the *EntryType* is assigned. If all of the bits in this field match Bit D0 of the *Flags* field, this entry is not defined. For this reason, the *EntryType* values zero and minus one are reserved.
OffsetToHeader	This field contains the offset from the beginning of the ENTRY structure where additional *EntryType* specific information is stored. The actual location of the specific information is determined by adding the value in this field to the offset of the *NextEntry* field on the PC Card. If this field is zero, there is no specific information and this field is ignored.
Flags	This field is bit-mapped. The bytes of this field are stored in little-endian order. Bit D1 indicates the current state of the entry. If Bit D1 matches D0, this entry is active. If Bit D1 does not match Bit D0, this entry has been deleted. All of the remaining bits are reserved and must be set to the value of Bit D0. Bit D0 is used in this manner to accommodate the unique erase and write characteristics of some types of storage media such as flash memory devices. Bit D0 is typically maintained in the natural erase state of the media.
OffsetToString	The field contains the offset from the beginning of the ENTRY structure where an ASCIIZ string describing the entry is stored. The actual location of the string is determined by adding the value in this field to the offset of the *NextEntry* field on the PC Card. If this field is zero, there is no ASCIIZ string and this field is ignored.
UniqueID	This field contains a value that is unique for each file stored in the partition. It is intended to be used to uniquely identify a file even if it is relocated within the partition. Formatting software should attempt to prevent the reuse of UniqueID values previously assigned to deleted entries. How this is accomplished is implementation specific.
Reserved	These bytes are reserved for future use. They shall be set to match the value of bit D0 of the *Flags* field.
EntryInfo	The data for this entry.

© 2001 PCMCIA/JEITA

5. TRANSLATION LAYERS

When sector remapping is performed by a translation layer, two storage formats are used. From the host system perspective, the same block storage format used for other media is being recorded. With the translation layer in place, the host system believes it is using traditionally formatted media.

However, the translation layer requires sector mapping information to be stored on the media and what the host feels are contiguous sectors are actually placed on the media out of sequence. To allow another system to recover the data stored on the media requires an understanding of how the translation layer remaps sectors and the native storage format intended by the original file system.

Any file format used for block data storage may be used on top of a Flash Translation Layer (FTL). To boot from a Flash Translation Layer (FTL) partition requires the FTL to translate host requests for virtual blocks.

© 2001 PCMCIA/JEITA

5.1 Virtual Block Device Flash Translation Layer - FTL

5.1.1 Versions or Revisions

Release	FTL Version
PC Card Standard Release 7.0 (February 1999)	1.2
PC Card Standard April 1998 (Release 6.1)	1.1
PC Card Standard May 1996 (Release 5.2)	1.0

5.1.2 Overview

Traditional block storage devices read and write data in small blocks sized in power-of-two multiples beginning at 256 (i.e. 256, 512, 1024, 2048, etc. byte blocks). File systems and the data structures they use are optimized for read/write units of this size. While flash media storage devices may be able to perform read and write operations on similar size blocks, they usually require a data area be erased before it may be written.

Adding an erase operation immediately before a write would appear to solve the problem and for some flash devices, this is all that is required. Unfortunately, for many flash devices, erase operations must often be applied to a contiguous area of the media known as an Erase Zone that is larger (in some cases, much larger) than traditional storage devices use for read and write access. This can make flash devices difficult to use with traditional file systems unprepared to relocate adjacent data areas to prepare for erase operations.

Two approaches have been taken to deal with the unique characteristics of flash storage devices: the development of new file systems customized for flash device characteristics and the introduction of a translation layer between the file system and the storage media to mask any differences between flash storage devices and traditional block storage devices. This section describes the later method, known as a Flash Translation Layer (FTL).

5.1.2.1 Emulating Traditional Block Devices - Virtual Block Device

An FTL masks the characteristics of flash devices from higher level software layers such as file systems by emulating a traditional block device. From the perspective of such higher level layers, a block storage device is a contiguous array of blocks numbered from zero (0) to one less than the number of available blocks. These higher level software layers expect to be able to write these blocks at will, without any regard for the need to first erase the media and certainly without any need to erase an area that exceeds the size of the block being written.

The FTL delivers this capability to the higher level software layers by remapping requests to write blocks to unallocated or free areas of the media and invalidating the area on the media previously containing the block's data. The FTL also records where the remapped block is placed on the media to allow subsequent read accesses to return the data written. In effect, the FTL presents a virtual block storage device to the higher level software layers. The size of these virtual blocks is determined when the storage media is formatted.

5.1.2.2 Flash Characteristics

A unique characteristic of flash media is its data content after erasure. If erased, flash media data bytes are either set to all ones (FFH) or all zeroes (00H). In addition, once a flash data bit has been set to a value other than its erase state, an entire Erase Zone must be erased to return the bit to its erased

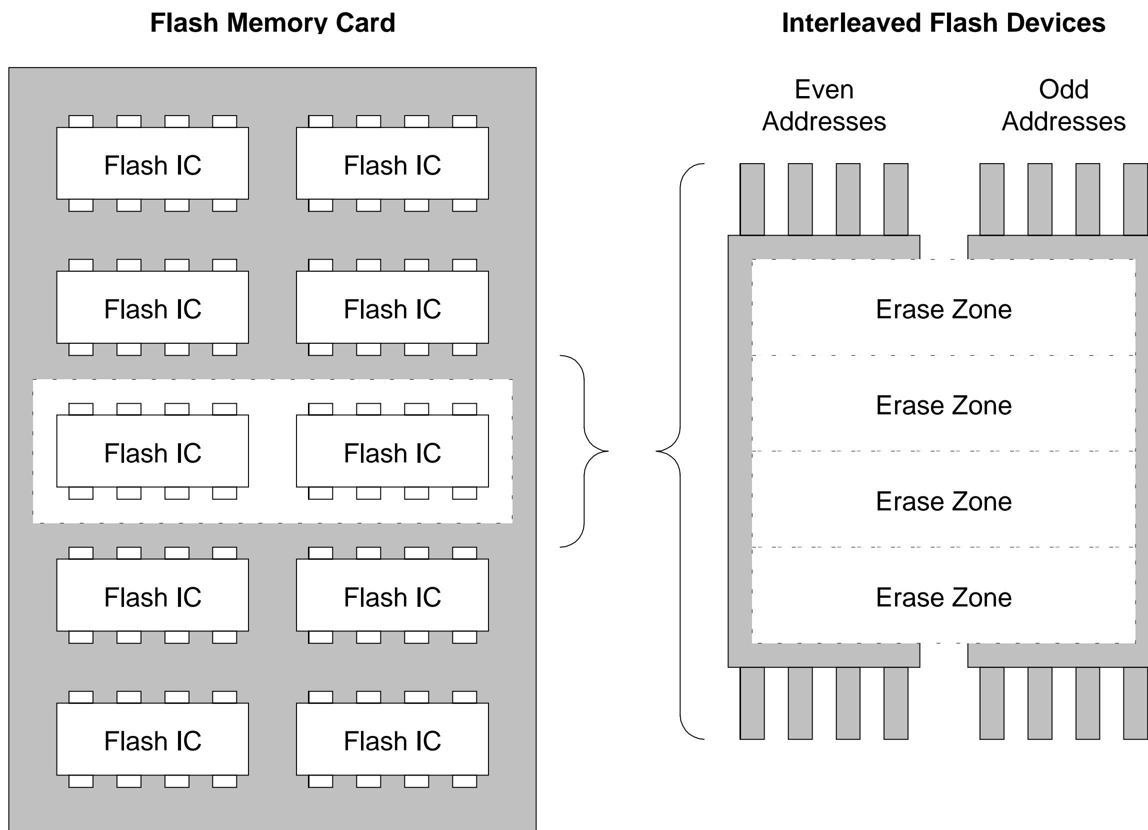
© 2001 PCMCIA/JEITA

state. However, single bits within a byte may be transitioned from the erased state to the non-erased state without an erase operation. FTLs use this ability to modify updatable fields in control structures. See the description of the ReversePolarityFlash bit of the **Flags** field in the Erase Unit Header.

5.1.2.3 Erase Zones and Erase Units

Depending on its technology, each flash IC on a PC Card may be divided into one or more Erase Zones of equal size. Each Erase Zone is the minimum contiguous area that can be erased in a single operation. If eight-bit devices are interleaved to provide sixteen-bit storage, the corresponding physical zones on two adjacent devices are combined together as a single Erase Zone. One device provides even addresses and the other device provides odd addresses.

An Erase Unit is a multiple of one or more contiguous Erase Zones. The size of an Erase Unit is set when the media is formatted. See ***Figure 5-1: Erase Zones***.



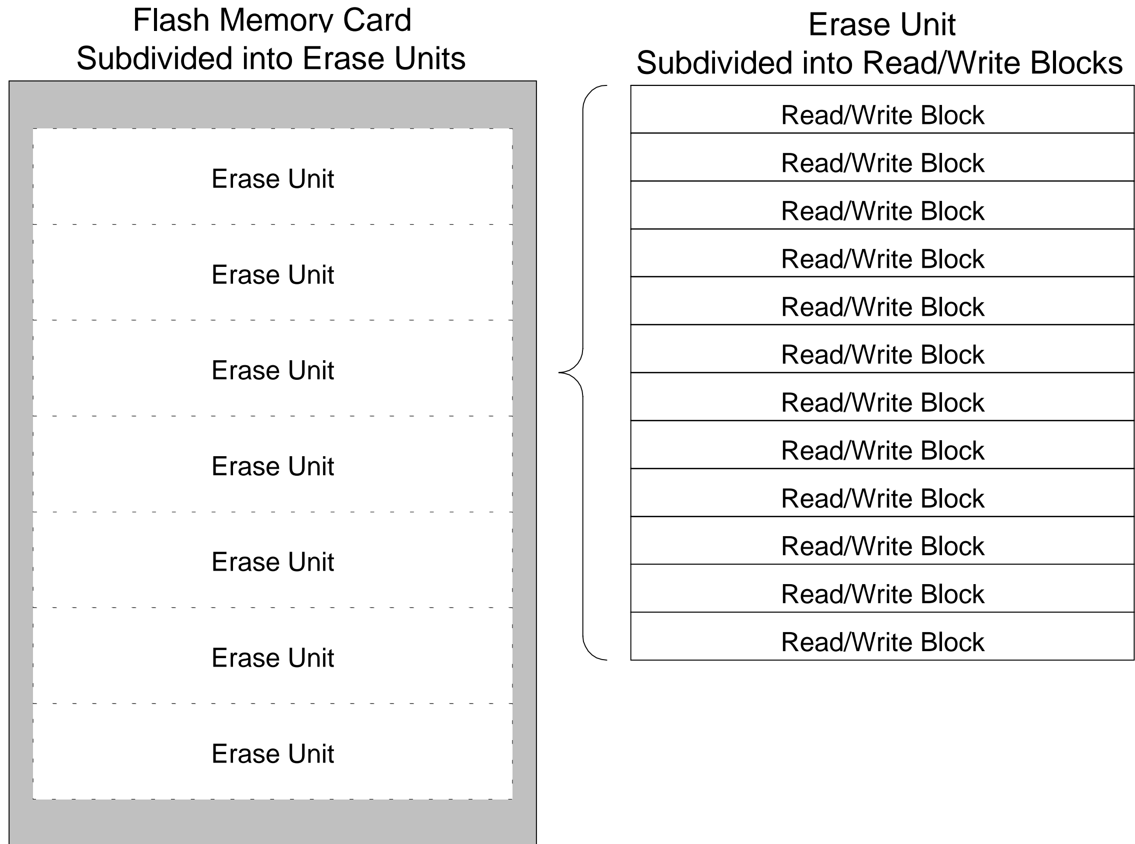
When a flash IC does not require the entire device to be erased as a single unit, a single flash IC contains multiple Erase Zones. When eight-bit flash devices are used, an Erase Zone spans two flash ICs.

Figure 5-1: Erase Zones

© 2001 PCMCIA/JEITA

5.1.2.4 Erase Unit Header and Block Allocation Information

For allocation purposes, an Erase Unit is evenly divided into one or more Read/Write Blocks of equal size. For example, a 128 KByte Erase Unit might be divided into 256 Read/Write Blocks, each 512 Bytes in size. Each of these Read/Write Blocks is the same size as the Virtual Blocks presented to the higher level software layers by the FTL. See ***Figure 5-2: Read/Write Blocks***.



An Erase Unit is evenly divided into read/write blocks for allocation purposes. Each of these read/write blocks is the same size as the virtual blocks presented to the higher level software layers by the FTL.

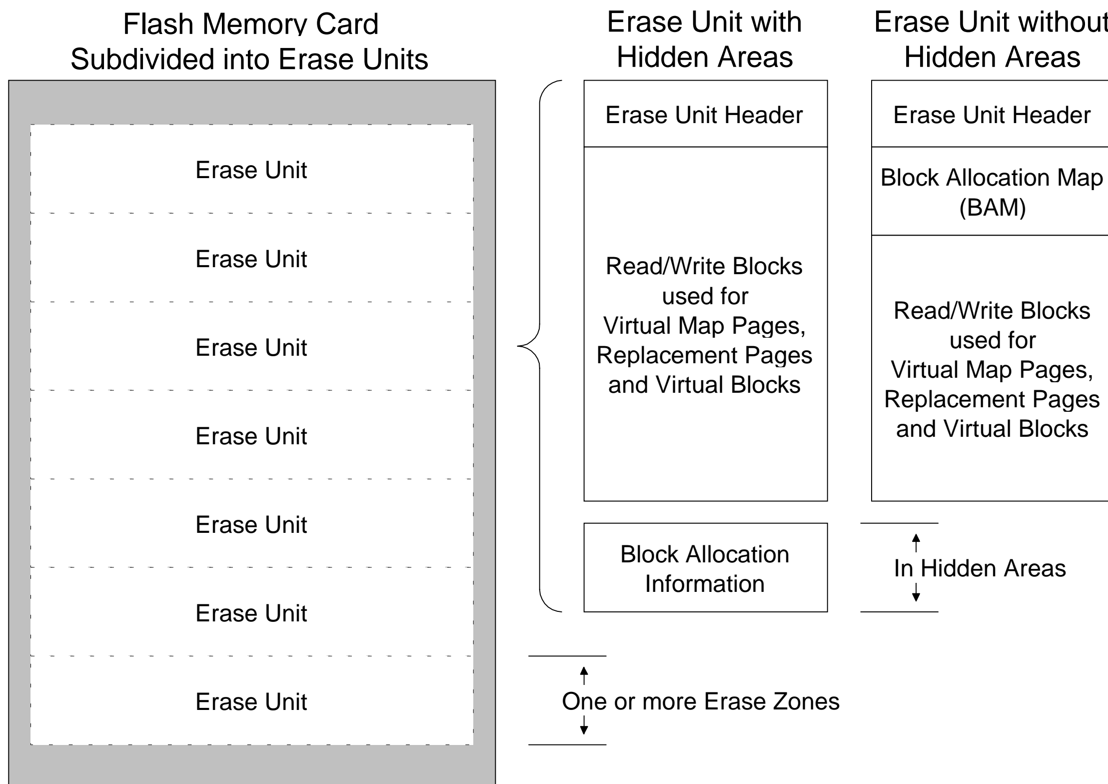
Figure 5-2: Read/Write Blocks

Within each Erase Unit is an Erase Unit Header (EUH). The EUH includes specific information about the Erase Unit and global information about the format of the FTL partition. Each Erase Unit also contains allocation information for all of the Read/Write Blocks within the unit.

Some flash devices have a small number of bytes of storage adjacent to each Read/Write Block which are not addressed through the linear address space of the media. Erase Units with these "hidden" areas may place block allocation information for a Read/Write Block in the hidden area adjacent to the block. If the Erase Unit does not have hidden areas or an FTL partition is not formatted to use the hidden areas, block allocation information is stored in an array known as the Block Allocation Map or BAM in the Erase Unit's linear address space. The Flags field of the EUH indicates whether block allocation information is stored in hidden areas or a BAM.

For data integrity purposes, two copies of the block allocation information may be stored in the Erase Unit. The Flags field of the EUH indicates the number of copies of the block allocation information that are present on the media. When BAMs are used to store block allocation information, the second copy is stored in a separate BAM immediately following the first BAM in the Erase Unit.

Optional Checksums, CRCs or ECCs may also be present for each Read/Write Block. If present, these codes are stored immediately following the block allocation information, in either the BAM or the media's hidden areas as indicated by the Flags field of the EUH. The length of these codes varies depending on the type of code being used. See ***Figure 5-3: Erase Unit Layout***.



Erase Units with hidden areas may place Block Allocation Information in those areas rather than in the linear address space of the media. Optional Checksums, CRCs or ECCs are not shown. If present, these codes are stored immediately following the Block Allocation Information in either the Block Allocation Map (BAM) or the media's hidden areas.

Figure 5-3: Erase Unit Layout

5.1.2.5 Block Allocation Information and the Block Allocation Map (BAM)

Each Erase Unit on the media contains allocation information for the Read/Write Blocks within the unit. For each Read/Write Block, a four-byte value tracks the block's current state. At any point in time, a Read/Write Block in an Erase Unit is free, deleted, bad, reserved for bad area management, or allocated. For each Read/Write Block, a four (4) byte value tracks the block's current state. The following table identifies the corresponding block allocation entry values for the state of a Read/Write Block:

 © 2001 PCMCIA/JEITA

BAI	Meaning	Description
FFFFFFFFH	Free	Read/Write Block is available, erased and ready to be written.
FFFFFFFEH OR 00000000H	Deleted	Data in block is not valid. Read/Write Block must be erased before it can be re-used. The value FFFFFFFEH indicates write operations were started on the Read/Write Block, but were interrupted before they were completed. The value 00000000H indicates the data in the Read/Write Block was superseded by normal update operations.
00000070H	Bad	Block is unusable.
xxxxxx10H	Bad Area Management	Block allocated for storing information about bad areas in the flash device.
Any Other Value	Allocated	Block is allocated. The actual BAI value describes the type of data stored in the Read/Write Block. Other values could be valid and should not cause operation errors. If a value is unrecognized, the block should be marked by software as "deleted" at initialization.

When allocated, Read/Write Blocks are used to store four types of data: FTL control structures, Virtual Block data for higher level software layers, Virtual Block Map Pages and Replacement Pages. When a Read/Write Block is allocated to an FTL control structure, the block's allocation information entry is set to the following:

00000030H	Control	Read/Write Block contains one or more FTL control structures: an Erase Unit Header, a Block Allocation Map (BAM), or an array of Checksums, CRCs or ECCs.

The block allocation entries for Virtual Block data, Virtual Block Map Pages and Replacement Pages have two parts. The least significant eight (8) bits indicate whether the Read/Write Block contains Virtual Block data, a Virtual Block Map Page or a Replacement Page as indicated by the following table:

xxxxxx40H	Data or Map Page	Read/Write Block contains Virtual Block data or a Virtual Map Page.
xxxxxx60H	Replacement Page	Read/Write Block contains a Replacement Page for a Virtual Map Page.

The most significant twenty-four (24) bits of the block allocation entry for a Read/Write Block containing Virtual Block data, a Virtual Block Map Page or a Replacement Page are the most significant bits of the virtual address of the data. The least significant eight (8) bits of the virtual address for these entries are assumed to be zero (0). To determine if the block is Data, Map, or Replacement page, compare the least significant 8 bits to 40H and/or 60H.

The FTL assigns a virtual address to each Virtual Block in the contiguous array of blocks presented to higher level software layers. The virtual address is computed by multiplying a Virtual Block's sequence number (zero to n - 1) by the size of a Virtual Block. The first Virtual Block is always at virtual address zero (0) as indicated by a block allocation entry of 00000040H. If the Virtual Block size is 512 bytes, the second block's virtual address is 00000200H and its block allocation entry is 00000240H. The third block's virtual address is 00000400H and its block allocation entry is 00000440H. See ***Figure 5-4: Block Allocation Map***.

The Virtual Block Map (VBM) is built from the virtual addresses of Read/Write Blocks used to store Virtual Block data. Read/Write Blocks containing Virtual Block data use positive virtual addresses. If the VBM is maintained on the media, the Read/Write Blocks containing VBM Pages or Replacements

Pages use negative virtual addresses. The use of negative addresses is discussed further in the sections that follow.

Allocation information is maintained in one of two ways. First, allocation information for all of the Read/Write Blocks in the Erase Unit may be stored together in an array in the unit known as the Block Allocation Map (BAM). Second, allocation information may be stored in hidden areas next to or related to the Read/Write Block to which it refers. The **Flags** field in the Erase Unit Header describes how allocation information is stored on the media.

NOTE: For reverse polarity flash the block allocation information stored on the media is inverted (for example, BAI entry 00000070H is FFFFFF8FH, etc. when a reverse polarity flash device is used). See the ReversePolarityFlash bit of the **Flags** field in the Erase Unit Header.

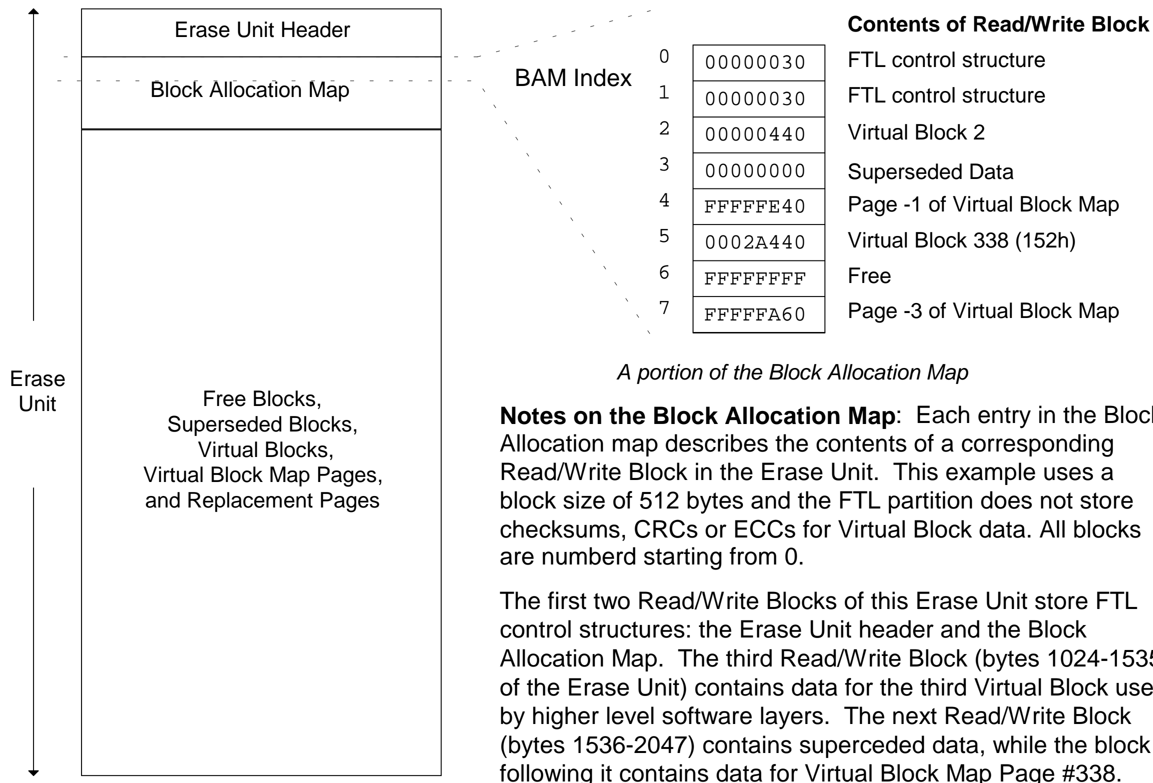


Figure 5-4: Block Allocation Map

5.1.2.6 Virtual Block Map - Mapping Virtual Blocks to Logical Addresses

The FTL uses a data structure known as the Virtual Block Map (VBM) to map requests for virtual blocks from higher level software layers to logical addresses on the media. The VBM is an array of 32-bit entries, each of which represents a logical address on the media where a Virtual Block's data is stored. The virtual block number requested by higher level software layers is used as an index into this array. See ***Figure 5-5: Virtual Block Map***.

The VBM is subdivided into Pages. Each Page of the VBM is the same size as the Virtual Blocks presented to the host system by the FTL.

 © 2001 PCMCIA/JEITA

The size of the VBM (in bytes) is determined by dividing the **FormattedSize** of the media by the **BlockSize** and multiplying the result by the size of a VBM entry (32-bits or four bytes). The number of Pages required for the VBM (**NumVMPages**) is the previous result divided by **BlockSize** and rounded up to the nearest Page.

For example, if **FormattedSize** is 12 megabytes and **BlockSize** is 512 bytes, the media contains 24K blocks. Each block has a four (4) byte entry in the VBM requiring 96 KBytes of the storage media to store the VBM. Dividing the size of the VBM by the block size indicates **NumVMPages** is 192. Each page of the VBM, in the example, is capable of recording the logical addresses used for 128 virtual blocks or 64 KBytes of data storage.

Space is always reserved on the media to store a VBM large enough to track the allocation of all the Virtual Blocks presented to higher level software layers by the FTL. However, when the media is formatted, the FTL may indicate only a portion of the VBM is maintained on the media. The **FirstVMAddress** field of the Erase Unit Header identifies the first virtual address on the media that has an entry in the VBM maintained by the FTL.

If the **FirstVMAddress** is reset to zero (0), the FTL must maintain all of the VBM entries on the media. If the **FirstVMAddress** exceeds the **FormattedSize**, none of the VBM entries are maintained on the media by the FTL. If the **FirstVMAddress** is greater than zero (0), but less than the **FormattedSize**, the FTL maintains VBM entries on the media for all virtual addresses greater than or equal to the **FirstVMAddress**. The system should determine if it has adequate resources to mount the media prior to mounting. All VBM entries stored on the media are in little-endian order.

When all or a portion of the VBM is not maintained on the media, the FTL must map requests for Virtual Blocks in some other way. An FTL might create and maintain a VBM in host system RAM. An FTL could also scan the media's block allocation information when each virtual block is requested. The choice to maintain the VBM on the media is typically based on the availability of system RAM and desired performance. Maintaining virtual to logical mapping information in system RAM relieves the FTL of the overhead of updating the VBM on the media for frequently updated Virtual Blocks.

If a VBM entry is all ones (FFFFFFFFH), the Virtual Block does not exist on the media. When asked to read data from this Virtual Block, the FTL may return any combination of bytes, such as binary 0's, as long as this combination is consistently returned until the Virtual Block is written.

There are two possibilities if a VBM entry is all zeroes (00000000H). First, the logical address of the Virtual Block is described on a Replacement Page. In this case, the FTL uses the logical address from the Replacement Page to locate the block. Second, if there is no Replacement Page, the block does not exist on the media. In the later case, when asked to read data from this Virtual Block, the FTL may return any combination of bytes, such as binary 0's, as long as this combination is consistently returned until the Virtual Block is written.

NOTE: For reverse polarity flash, the VBM entries stored on the media are inverted (for example, VBM entry FFFFFFFFH is 00000000H, etc. when a reverse polarity flash device is used). See the ReversePolarityFlash bit of the **Flags** field in the Erase Unit Header.

© 2001 PCMCIA/JEITA

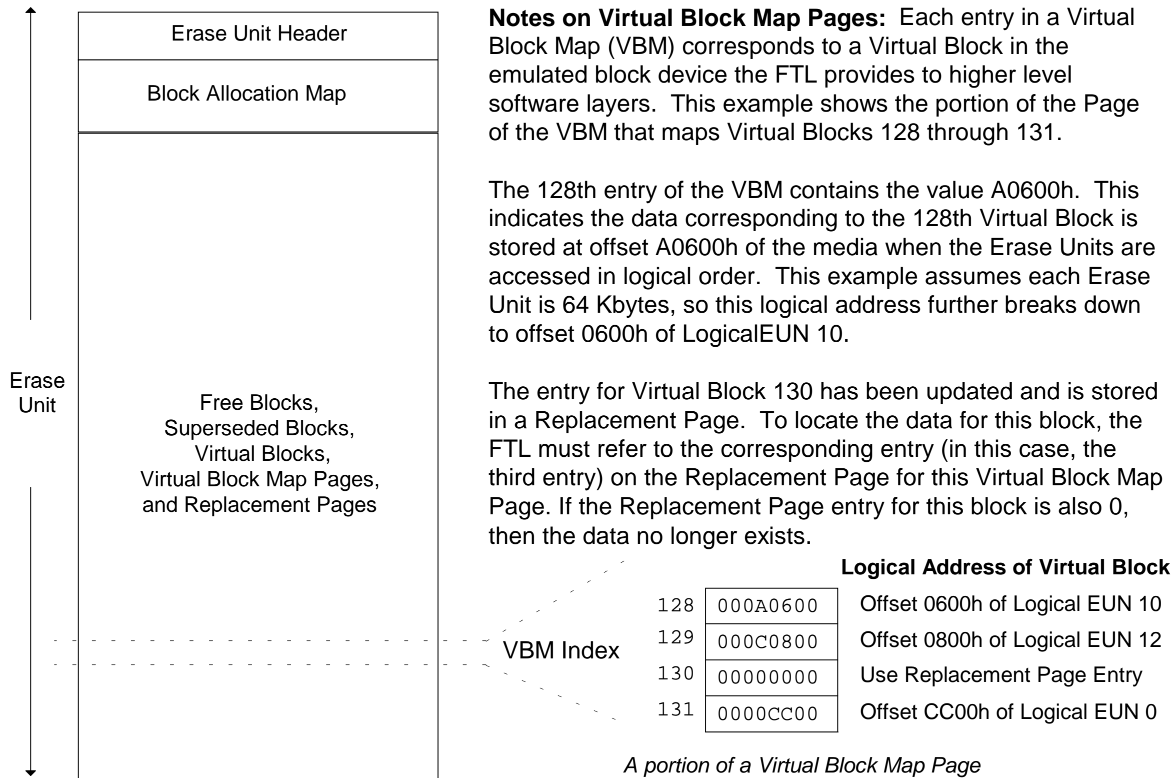


Figure 5-5: Virtual Block Map

5.1.2.7 Virtual Page Map - Locating the Pages of the Virtual Block Map

When the Virtual Block Map (VBM) is maintained on the media, the FTL must track the storage of the Pages of the VBM. In the same manner that VBM entries indicate the logical address of a Virtual Block, the entries of the Virtual Page Map (VPM) indicate the logical address on the media where the Pages of the VBM are stored. Unlike the VBM, the VPM is never stored on the media. Where the FTL stores the VPM is implementation dependent. See ***Figure 5-6: Page Mapping***.

The block allocation information entries describing Virtual Block Map Pages use negative values to distinguish them from the Read/Write Blocks used to store Virtual Block data which use positive values. In the example used in the previous section, the VBM requires 192 Pages. If the entire VBM is maintained on the media (see the previous section), each page of the VBM requires a Read/Write Block to store the VBM entries found on the page.

Virtual Block Map Pages are numbered using negative values, therefore their virtual addresses are negative. As with Virtual Blocks, the virtual address stored as the block allocation information for these pages is computed by multiplying the page number by the size of a Virtual Block (which is the same size as the Read/Write Block used to store the page's data). Returning to the previous example, the first page of the VBM (used to store the logical addresses of the first 128 Virtual Blocks) is number -192. The second page of the VBM is number -191 and the last page of the VBM is -1. With a block size of 512 bytes, the first VBM Page in the example is virtual address FFFE8000H. The virtual address of the second VBM Page is FFFE8200H, and the last VBM Page would be virtual address FFFFFE00H. To locate an address within a VBM Page, take the sector requested modulo 128. Multiply this result but the Virtual Block entry width and this yields the correct offset within the VBM Page.

 © 2001 PCMCIA/JEITA

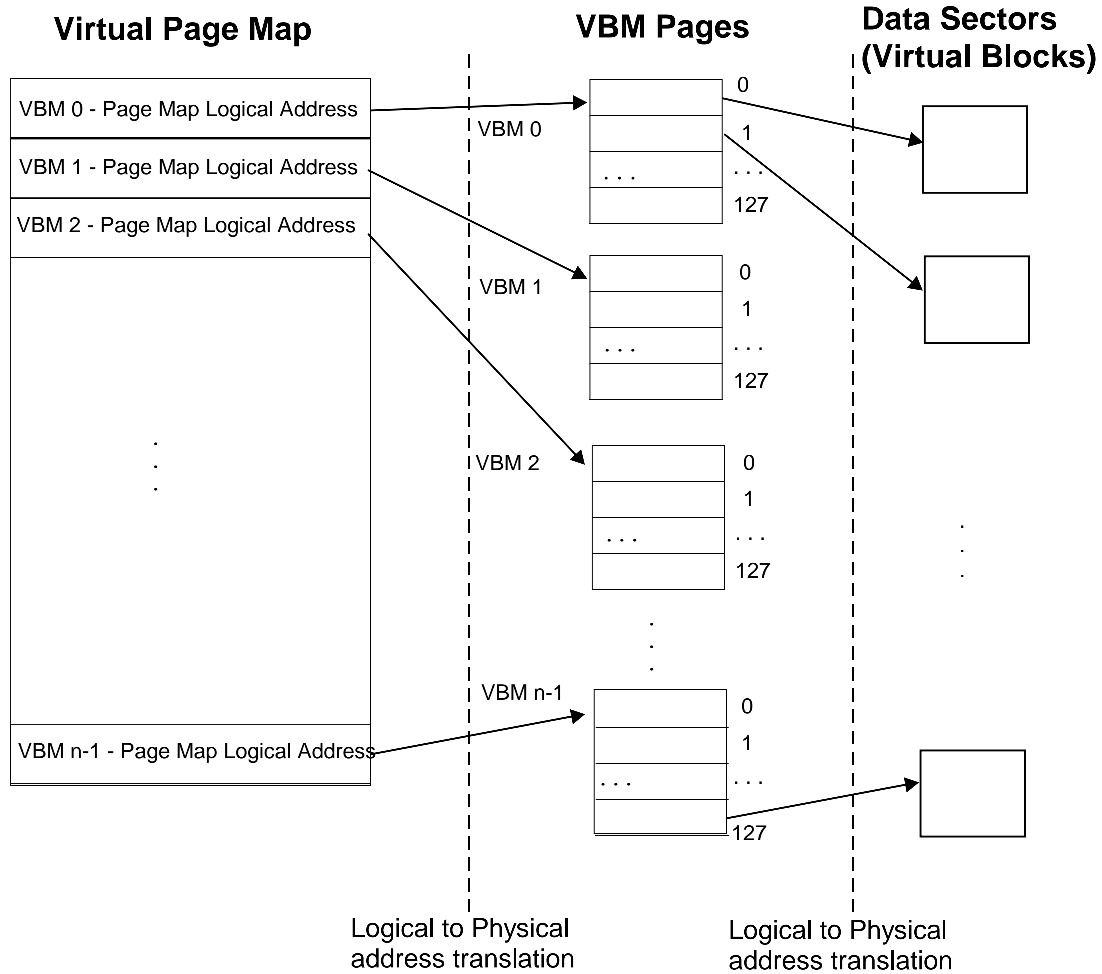


Figure 5-6: Page Mapping

5.1.2.8 Replacement Pages

Each page of the Virtual Block Map (VBM) may have a Replacement Page. Replacement pages may be used to improve performance of a system. Values in a Replacement Page override entries in the original VBM page as follows:

If an entry in an original VBM page is zero (0), the logical address of the Virtual Block is retrieved from the corresponding entry on the Replacement Page. If there is no Replacement Page, or the corresponding entry on the Replacement Page is zero (0), the Virtual Block does not exist on the media.

Replacement Pages are used to improve performance by minimizing the need to supersede Pages in the VBM when logical addresses on a Page are updated. To ensure compatibility with other host systems, when reading, the system must be able to handle usage of one Replacement Page and when writing is allowed to use up to one (1) at any one time.

Replacement Pages are allocated from Free Read/Write Blocks in any Erase Unit. The FTL locates allocated Replacement Pages by scanning the block allocation information on the media. This scan

may be performed when the media is inserted in the host system or when a VBM entry of zero (0) is encountered. Replacement Pages may **NOT** be replaced.

The block allocation information entry for a Replacement Page uses the same virtual address as the original VBM Page. The FTL distinguishes between the two using the least significant eight (8) bits of the block allocation information. VBM Pages have a value of 40H in the these bits while Replacement Pages have a value of 60H.

5.1.2.9 Mapping Logical Addresses to Physical Addresses

The addresses stored in the Virtual Block Map and Virtual Page Map arrays are logical addresses. A logical address represents a location in the media described when the Erase Units are ordered in LogicalEUN sequence. The FTL determines the relationship between LogicalEUN and PhysicalEUN by scanning the media. As each Physical Erase Unit is encountered, its LogicalEUN is noted. Later, the FTL uses this information to map the logical address in the VBM or VPM to a physical address on the media where the data for the Read/Write Block is stored.

Since Erase Units are always sized as a power of two (2), the logical address may be considered to contain a LogicalEUN in the most significant bits and an offset into the Erase Unit in the least significant bits. Which bits represent the offset address and which bits represent the LogicalEUN depend on the size of an Erase Unit.

As an example, if Erase Units were 64 KBytes, the low word of the logical address would be the offset within the Erase Unit and the upper word of the logical address would be the LogicalEUN. If the FTL maintains an array of PhysicalEUNs in LogicalEUN order, translating the LogicalEUN to a PhysicalEUN is a simple matter of indexing into the array using the LogicalEUN from the upper word of the logical address. Continuing with the example, the offset of the Read/Write Block storing the data being mapped is the double word created by placing the PhysicalEUN in the upper word and the offset into the Erase Unit into the lower word.

5.1.3 Data Structures

This section describes the data structures used on media formatted for the Flash Translation Layer (FTL). Unless otherwise noted, all values stored in FTL data structure fields are in little-endian format.

The FTL stores persistent allocation information on the media for each Virtual Block. This allocation information is stored in the same Erase Unit as the Virtual Block. A Virtual Block Map (VBM) is maintained by the FTL based on this allocation information that reorganizes the information into an array of four byte entries that describe the logical location of a Virtual Block on the media.

The VBM may reside on the media or in a non-persistent space available to the host system such as system memory. The portions of the VBM that are not stored on the media are built during FTL initialization when new media is inserted based on the block allocation information stored in each Erase Unit.

5.1.3.1 Erase Unit Header (EUH)

Each Erase Unit on the media contains an Erase Unit Header (EUH). The EUH is located at offset zero (0) of the Erase Unit or at the location specified by the AltEUHOffset field of an EUH at offset zero (0) of another Erase Unit. The Erase Unit Header contains information specific to the Erase Unit and global information about the entire FTL partition.

© 2001 PCMCIA/JEITA

Offset	Field	Size	Detail/Description
0	LinkTargetTuple	5	This field contains a Link Target Tuple (see the ***Metaformat Specification***). The contents of this field are the same for all Erase Units. TPL_CODE CISTPL_LINKTARGET (13H) TPL_LINK 3 TPLTG_TAG 'C', 'I', 'S'
5	DataOrganizationTuple	10	This field contains a CISTPL_ORG tuple (see the ***Metaformat Specification***). The contents of this tuple are the same for all Erase Units. The value used for the TPL_LINK field is the number of remaining bytes in the Erase Unit Header. The next tuple (after the Erase Unit Header) may be an End-Of-List Tuple to terminate the chain, or the tuple chain may continue with additional tuples. TPL_CODE CISTPL_ORG (46H) TPL_LINK At least fifty-seven (57), including the size of the remaining fields in the Erase Unit Header. TPLORG_TYPE TPLORGTYPE_FS (0) TPLORG_DESC "FTL100", 0
15	NumTransferUnits	1	Number of Transfer Units in the partition. All FTL partitions have at least one (1) Transfer Unit.
16	Reserved	4	This field shall be ignored when read and shall be set to the device's erased state when written.
20	LogicalEUN	2	The Logical Erase Unit Number currently assigned to this unit. The LogicalEUN of a formatted Transfer Unit is the media's erased state (for most flash devices this is all ones or FFFFH).
22	BlockSize	1	The size of all Virtual and Read/Write Blocks. This field is expressed as a log2 value. For example, for a block size of 512 bytes this field is set to nine (9). This field must be at least eight (8) to represent the minimum block size of 256 bytes.
23	EraseUnitSize	1	The size of an Erase Unit. This field is expressed as a log2 value. For example, for a unit size of 128 KBytes this field is seventeen (17). Erase Units are always a multiple of the flash device's erase zone size.
24	FirstPhysicalEUN	2	The Physical Erase Unit Number where the FTL partition begins. If the partition starts at physical address zero (0), this value is zero (0).
26	NumEraseUnits	2	Total number of Erase Units in the FTL partition. This field includes Erase Units used to store data, block allocation information, checksums (if present), transfer units, replacement pages, spare blocks and the Virtual Block Map. The total number of Erase Units also includes bad units.
28	FormattedSize	4	The formatted size of the partition. This is the total space available to the host system for data storage. This field does not include space marked as format blocks or used to for transfer units, replacement pages and the Virtual Map. Formatting utilities should also exclude a few additional blocks to use as spares. Without spare blocks, a Unit Recovery Procedure is required anytime a Virtual Block is updated after all the Virtual Blocks have been written once. This field is expressed in bytes. This field must be a multiple of BlockSize.
32	FirstVMAddress	4	The first virtual address for which Virtual Block Map (VBM) entries are maintained by the FTL on the media. If this field is zero (0), the entire VBM is maintained on the storage media. If this field exceeds the FormattedSize field, none of the VBM is maintained on the media. However, space for the entire VBM must be available on the media at all times.

36	NumVMPages	2	Number of Pages in the Virtual Block Map (VBM). If the FirstVMAddress field is greater than the FormattedSize field, the FTL does not maintain the VBM on the media, even though space is reserved on the media. If the FirstVMAddress field is less than the FormattedSize field, some or all of the VBM stored on the media is maintained by the FTL.
38	Flags	1	Bit-mapped field describing how checksum and block allocation information are stored on the media and the polarity of the media. See ***5.1.3.2 Flags*** below.
39	Code	1	Binary value describing the type of Checksum, CRC or ECC information maintained for Virtual Block data. If this value is the erase state of the media, no such information is present. If this value is the non-erase state of the media, such information was present at one time, but is no longer being maintained. Any other value indicates the type of information being maintained. If this value is set to one (1), a two (2) byte checksum is computed and maintained for each Virtual Block. These two (2) byte checksums are computed by adding each byte of the Virtual Block to a 16-bit value initially set to zero (0) and ignoring overflow. The checksum is stored in little-endian format. Where the checksum is stored depends on the setting of the HiddenAreaFlag of the Flags field. Any other value than those described above is reserved for future expansion.
40	SerialNumber	4	Partition serial number. May be used to distinguish between partitions and/or PC Cards.
44	AltEUHOffset	4	Offset of an alternate Erase Unit Header. Used when it is not possible to write an Erase Unit Header at offset zero (0) of the Erase Unit. If used, the value in this field is the same for all Erase Units in the partition. However, the FTL always attempts to place the EUH at offset zero (0). For each Erase Unit the FTL is able to place the EUH at offset zero (0), the alternate location is not used. When all Erase Unit Headers are at offset zero (0) and there is no Alternate Erase Unit Header in use, this field shall be in the erase state of the media. This allows an Alternate EUH to be assigned at run-time, if the beginning of an Erase Unit should go bad during use. Should an Alternate EUH be assigned at run-time, all EUHs must be updated to reflect the AltEUHOffset. The FTL first searches for the EUH at offset zero (0) of the Erase Unit. Only if the EUH is not found there does the FTL search at the location specified by this field. This presumes the FTL has successfully read an EUH at offset zero (0) of another Erase Unit to determine the location used for Alternate EUHs. If an EUH cannot be found at either location, the FTL assumes the Erase Unit is unformatted. The AltEUHOffset should be an integer multiple of 4 Kbytes.

© 2001 PCMCIA/JEITA

48	BAMOffset	4	The offset from the start of the Erase Unit Header to the Block Allocation Map (BAM) contained in the Erase Unit. This field is expressed in bytes. This field is only valid if the Flags - HiddenAreaFlag is reset to zero (0). The Block Allocation Map is not required to be aligned on a virtual block boundary. It may immediately follow the tuple chain encapsulating the Erase Unit Header. If the Flags - DoubleBAI is set to one (1), two copies of the BAM are present on the media. The second copy follows the first copy and precedes any Checksum, CRC or ECC information. If an Erase Unit is using Checksums, CRCs or ECCs as indicated by the Code field, these codes shall follow the block allocation information. If the allocation information is stored in the BAM, an array of codes follow the array of allocation information in the BAM.
52	Reserved	12	Reserved for future use. This field shall be left in the media's erased state. For most flash devices this is all ones (FFH).
64	FTLRevisionTuple	114	This optional field contains an FTL Revision Tuple. The contents of this field are the same for all Erase Units. TPL_CODE vendor unique (80H) TPL_LINK 10 FTL-VER 'FTL VER x.x' where x.x is the ASCII equivalent version number of the FTL specification that the software is compliant to. For example, FTL version 1.1 would have entries 46H, 54H, 4CH, 20H, 56H, 45H, 52H, 31H, 2EH, 31H.

5.1.3.2 Flags

The bit-mapped **Flags** field of the Erase Unit Header describes how checksum and block allocation information is stored on the media and identifies the media's erase state.

Bit	Description
0	HiddenAreaFlag - This flag indicates whether checksum and allocation information are stored in the Block Allocation Map (BAM) or the media has hidden, alternate storage areas for such information that do not appear in the media's normal address space. If this bit is set to one (1), checksum and/or allocation information are stored in hidden areas outside of the media's normal address space. If this bit is reset to zero (0), any checksum and/or allocation information are stored in the media's normal address space. The Block Allocation Map (BAM), an array of all of the block allocation information, precedes an array of the checksum information.
1	ReversePolarityFlash - This flag indicates whether the media's flash memory devices erase to all ones or to all zeroes. If this bit is set to one (1), the flash memory devices erase to all zeroes (0) and may be written to one. When this bit is set, all block allocation information, Virtual Map entries and LogicalEUNs must be inverted before they are written to the media and after they are read from the media. If this bit is reset to zero (0), the flash memory devices erase to all ones (1) and may be written to zero. When this bit is reset, all block allocation information, Virtual Map entries and LogicalEUNs are read and written without inversion.
2	DoubleBAI - This flag indicates whether there are one or two copies of the block allocation information stored on the media. If this bit is set to one (1), two copies of the block allocation information are present on the media. If this information is stored in BAMs, there are two complete BAMs on the media. In this case, the first BAM begins at the location specified by the BAMOffset field and the second BAM begins immediately after the first. If the block allocation information in stored in hidden areas, as indicated by the HiddenAreaFlag, the second entry for a Read/Write Block follows the first. Both copies of the block allocation information precede any Checksum, CRC or ECC codes. If this bit is reset to zero (0), only one copy of the block allocation information is stored on the media.
3 .. 7	Reserved for future use. All of these bits must be reset to zero (0).

5.1.4 Partition Recognition

Flash Translation Layers (FTLs) may be identified in two ways. They may be explicitly identified in a PC Card's Card Information Structure (CIS) or they may be recognized by searching the storage media for FTL data structures.

If the CIS contains partition information, an FTL partition is identified by the Data Organization Tuple (CISTPL_ORG, 46H). (See the ***Metaformat Specification*** for details on describing partitions in the CIS.) An FTL Data Organization Tuple is formatted as follows:

Byte	D7	D6	D5	D4	D3	D2	D1	D0
0	Tuple Code		CISTPL_ORG, 46H					
1	Tuple Link		Link to next tuple (at least 07H)					
2	TPLORG_TYPE	TPLORGTYPE_FS, 00H						
3 .. 9	TPLORG_DESC	"FTL100\0"		Null terminated string identifying FTL partition				

If the entire storage media is used by an FTL, the CIS is not required to contain partition information. In this case, an FTL partition is recognized if an FTL Erase Unit Header (see ***5.1.3 Data Structures***, above) is found in the first megabyte of the storage media and the information in the Unit Header is valid. The procedure for recognizing and validating an FTL Erase Unit Header is described in the following paragraphs.

The first step in recognizing an FTL partition is confirming the presence of the FTL Data Organization Tuple described above at offset five (5) of the Erase Unit Header. Since Erase Unit Headers always begin at offset zero (0) of an Erase Unit and an Erase Unit is always a multiple of a flash device's erase zone size, the search is limited to the first fifteen bytes of each flash erase zone in the first megabyte of the storage media. If the flash erase zone size is not known, the search for an Erase Unit Header is made in four (4) KByte increments.

If the FTL Data Organization Tuple is not found within the first megabyte of the partition area, the media is not an FTL data store. If the tuple is found, the rest of the Erase Unit Header must be validated. All Erase Units should have identical EUHs with the exception of the Logical EUN and reserved fields. One level of validation would be to check to see that all EUHs are the same. Further validation could include verifying that the FormattedSize is a multiple of the BlockSize, that the BlockSize is smaller than the EraseUnitSize, and that there is at least a 30H ID at the address that the BAMOffset points to. The validation process may also be used to build a dynamic map of the logical to physical translation performed by the FTL.

The FTL then reads every Erase Unit Header on the media, starting with the unit described by the **FirstPhysicalEUN** field. If not found at offset zero, the FTL looks for the Erase Unit Header at the location specified by the **AltEUHOffset** field of other Erase Units. If an Erase Unit Header is not found at either location, the Erase Unit is considered an unformatted Transfer Unit. If two units are found with the same value in the **LogicalEUN** field, either of the units may be assumed to be a Transfer Unit. After all Erase Unit Headers have been read, the number of units with non-negative and distinct **LogicalEUN** fields must equal the **NumEraseUnits** field less the **NumTransferUnits** field.

The number of Virtual Blocks used to store the Virtual Block Map (VBM) on the media is indicated by the **NumVMPages** field in the Erase Unit Header. During the block allocation scan, the FTL locates VBM Pages and Replacement Pages. The number of VBM Pages located must match the value in the **NumVMPages** field. If a VBM Page is missing and a Replacement Page exists, the Replacement Page is used in place of the original VBM Page. If a VBM Page is missing and a Replacement Page does not exist, implementation dependent recovery operations are required. If duplicate VBM Pages are found, all but one is ignored.

© 2001 PCMCIA/JEITA

The assignment of Replacement Pages is implementation specific. A Replacement Page is indicated by allocation information having the same virtual address as an original VBM Page with the least significant eight (8) bits set to 60H.

5.1.5 Partition Formatting

A Flash Translation Layer (FTL) partition is prepared for use as follows:

Determine the appropriate values for global fields in the Erase Unit Header:

DataOrganizationTuple
BlockSize
EraseUnitSize
FirstPhysicalEUN
NumEraseUnits
NumTransferUnits
FormattedSize
FirstVMAddress
NumVMPages
Flags
SerialNumber
AltEUHOffset
BAMOffset

For each Erase Unit on the media: Erase the unit and write out an Erase Unit Header with unit specific data. Unit specific data includes the following field:

LogicalEUN

LogicalEUNs range from zero (0) to one less than the **NumEraseUnits** field less the **NumTransferUnits** field. The order **LogicalEUN**s are assigned is not significant as long as each is unique in the above range. The LogicalEUN of Transfer Units is left in the media's erased state (for most flash devices this is all ones or FFFFH). See the ReversePolarityFlash bit of the **Flags** field.

For each Read/Write Block used to store Erase Unit Headers and the Block Allocation Map (if used) set the allocation information to 30H to indicate the blocks contain formatting data.

Once the FTL has prepared the media, any block-oriented file system may store its data formats on the media using the FTL to translate Virtual Block requests to the appropriate locations on the media.

5.1.6 Logical Block Operations

This section describes how the FTL performs accesses to Virtual Blocks on the media.

5.1.6.1 Read

Higher level software layers request a Virtual Block from the FTL. The FTL uses the number of the Virtual Block as an index into the Virtual Block Map (VBM). The entry in the VBM is the logical address where the Virtual Block data is stored. The FTL converts the logical address to a physical address. The conversion from logical to physical is based on the relationship of the **LogicalEUN** containing the logical address to the physical Erase Unit used for the logical Erase Unit. The FTL transfers the data block at this location into the buffer provided by the host file system.

If an entry in the VBM is all ones (FFFFFFFFH), the block does not exist on the media. The FTL may return any combination of bytes, such as binary 0's, as long as this combination is consistently returned until the Virtual Block is written.

There are two possibilities if a VBM entry is all zeroes (00000000H). First, the logical address of the Virtual Block is described on a Replacement Page. In this case, the FTL uses the logical address from the Replacement Page to locate the block. Second, if there is no Replacement Page or the entry on the Replacement Page is all zeroes or Fs (00000000H or FFFFFFFFH), the block does not exist on the media. In the later case, when asked to read data from this Virtual Block, the FTL may return any combination of bytes, such as binary 0's, as long as this combination is consistently returned until the Virtual Block is written.

Some mechanism should be employed to ensure proper handling if a power cycle is received while accessing the flash device.

5.1.6.2 Write

When writing Virtual Block data to the media, the FTL must first locate a free Read/Write Block. If a free block is not available, one is created using the Unit Recovery Procedure described below.

Once a free block is located, block allocation information for the area is marked to indicate a write operation is beginning by resetting the least significant bit (FFFFFFFEH). Once the write has been successfully completed, the block allocation information in the Erase Unit is updated to reflect the Virtual Block's virtual address.

Next, the Virtual Block Map is updated to reflect the new area assigned to the Virtual Block. Finally, if the new block replaces an existing block, the Read/Write Block used to store the superseded data is marked as deleted by resetting its block allocation information to zero (00000000H).

If the write process is interrupted at any point, the FTL is able to recover with minimal effort. If the interruption occurs before the data write completes, the block allocation information (FFFFFFFEH) indicates the block shall be treated as deleted and normal activity will recover the space when required.

If an interruption occurs after the block allocation information is updated, but before the VBM is updated, both Read/Write Blocks have the same block allocation information. This can also occur if the update of the VBM entry completes, but the superseded Read/Write Block's allocation information is not marked as deleted. In either case, the FTL selects the block pointed to by the VBM and treats the other block as superseded. The system is allowed to use up to one (1) Replacement Page during writes.

Some mechanism should be employed to ensure proper handling if a power cycle is received while accessing the flash device.

5.1.6.3 Unit Recovery

Deleted and/or superseded Read/Write Blocks may only be re-used after they are erased. All of the Read/Write Blocks in an Erase Unit must be erased at the same time. Rarely are all of the Read/Write Blocks within an Erase Unit deleted and/or superseded. The Unit Recovery Procedure performs the necessary processing to safely preserve data in allocated Read/Write Blocks and recover deleted and/or superseded blocks.

The first step is to locate a properly prepared Transfer Unit. To be properly prepared, the area used to store Virtual Block data, Virtual Block Map Pages, Replacement Pages and block allocation information must be erased and the global fields of the Transfer Unit's Erase Unit Header must be

© 2001 PCMCIA/JEITA

initialized. A formatted Transfer Unit contains the EUH and BAM. The BAM contains only Control (30H) entries for the FTL structures for that Transfer Unit.

Before Read/Write Block data from the Erase Unit being recovered is copied to the Transfer Unit, the **LogicalEUN** of the Transfer Unit is set to 7FFFH. After allocated Read/Write Blocks have been successfully transferred, the Transfer Unit's **LogicalEUN** is set to the **LogicalEUN** of the Erase Unit being recovered. If the Unit Recovery Procedure is interrupted, the Transfer Unit's **LogicalEUN** remains 7FFFH, and the FTL can determine the Transfer Unit is not properly prepared.

After all allocated Read/Write Blocks have been successfully moved to the Transfer Unit and the **LogicalEUN** updated, the original Erase Unit is erased and formatted as a Transfer Unit. If the Unit Recovery procedure is interrupted before the original Erase Unit is erased, the FTL will find two Erase Units with the same data. The FTL may use either Erase Unit as the specified Logical Erase Unit and the other Erase Unit becomes a Transfer Unit.

5.1.7 Initial Program Load

Any file format used for block data storage may be used on top of a Flash Translation Layer (FTL). To boot from a Flash Translation Layer (FTL) partition requires the FTL to translate host requests for virtual blocks.

© 2001 PCMCIA/JEITA

6. STORAGE DEVICES

PC Cards use a number of storage technologies with various read, write, erase and access characteristics. The relative performance of a technology is often affected by the data storage format used to record information on a PC Card. The characteristics of some technologies may dictate a particular access method or may prohibit or severely limit the ability to use a particular file format.

6.1 Static RAM Cards

S-RAM device allow byte-oriented read and write access and do not require the media to be erased before it may be written. This allows S-RAM PC Cards to store data without requiring a translation layer. S-RAM memory devices on a PC Card may be mapped into host system memory and directly accessed.

6.2 Flash Memory Cards

Flash memory devices may allow byte-oriented read and write access or may require block-oriented access. Flash memory devices typically require the media to be erased before it may be written and usually require an erase operations to be performed in blocks of contiguous bytes. The size of an erase block varies among flash devices, ranging from small blocks of 256 or 512 bytes to blocks of 64 KBytes or more. Most linear flash memory PC Card may be mapped into host system memory and directly accessed.

Unless access will be read-only, using traditional file systems with flash media is problematic due to the unique characteristics of flash devices. Two methods are used to deal with flash media.

First, custom file systems designed especially for data storage on flash devices may be developed. These file systems use a byte-oriented access to the media as opposed to the traditional block access. In addition, file allocation is typically handled using linked lists which allow file updates to be routed to erased areas of the media and the previously allocated areas marked as available for recovery. When all erased space has been used, a recovery routine creates, clears and erases a contiguous block of marked areas to allow them to be reused.

The second method for using flash media adds an additional translation layer between the traditional file system and the flash media. The file system continues to read and write sector-sized blocks, but instead of mapping directly to physical sectors, the translation layer tracks media use, remaps sector write requests to erased areas and marks the previously used sector(s) as available for recovery. Subsequent read requests are translated to access the appropriate remapped sectors.

6.3 PC Card ATA Drives

PC Card ATA drives use block-oriented read and write access and do not require the media to be erased before it is written. PC Card ATA drives are typically used with traditional block-oriented file systems and do not require a translation layer. The data storage area of a PC Card ATA drive is usually accessed indirectly. Information is transferred between a PC Card ATA drive and the host system by placing request parameters into task file registers located on the card.

© 2001 PCMCIA/JEITA

This Page Intentionally Left Blank

© 2001 PCMCIA/JEITA