Microsoft Windows

Virtual Hard Disk Image Format Specification

October 11, 2006 - Version 1.0

Abstract

This paper describes the different virtual hard disk formats supported by Microsoft Virtual PC and Virtual Server and provides information about how to store the data.

For comments or questions, please send e-mail to vhdtalk@microsoft.com.

Contents

Introduction

This paper describes the different hard disk formats supported by Microsoft Virtual PC and Virtual Server products. It does not explain how hard disks interface with the virtual machine, nor does it provide information about ATA (AT Attachment) hard disks or Small Computer System Interface (SCSI) hard disks. This paper focuses on how to store the data in files on the host file system.

The reader should be familiar with virtual machine technology and terminology, such as the terms *guest* and *host* as used in the context of virtual machine architectures. The user should also be familiar with hard disk technologies and should understand how data is accessed and laid out on the physical medium. The following terminology is used in this paper:

System
Refers to Virtual PC, Virtual Server, or both.

Absolute Byte Offset
Refers to the byte offset from the beginning of a file.

Reserved
Fields marked reserved are deprecated, or are reserved for future use.

Sector length
Sector length is always 512 bytes.

All values in the file format, unless otherwise specified, are stored in network byte order (big endian). Also, unless otherwise specified, all reserved values should be set to zero.

Overview of Virtual Hard Disk Image Types

Virtual machine hard disks are implemented as files that reside on the native host file system. The following types of virtual hard disk formats are supported by Microsoft Virtual PC and Virtual Server:

- Fixed hard disk image
- Dynamic hard disk image
- Differencing hard disk image

Each virtual hard disk image has its own file format, as described in the following sections.

Fixed Hard Disk Image

A fixed hard disk image is a file that is allocated to the size of the virtual disk. For example, if you create a virtual hard disk that is 2 GB in size, the system will create a host file approximately 2 GB in size.

The space allocated for data is followed by a footer structure. The size of the entire file is the size of the hard disk in the guest operating system plus the size of the footer. Because of size limitations in the host file system, a fixed hard disk might be limited. For example, on a FAT32 file system, the maximum size of the virtual hard disk is 4 GB.

Dynamic Hard Disk Image

A dynamic hard disk image is a file that at any given time is as large as the actual data written to it plus the size of the header and footer. Allocation is done in blocks. As more data is written, the file dynamically increases in size by allocating more

blocks. For example, the size of file backing a virtual 2-GB hard disk is initially around 2 MB on the host file system. As data is written to this image, it grows with a maximum size of 2 GB.

Dynamic hard disks store metadata that is used in accessing the user data stored on the hard disk. The maximum size of a dynamic hard disk is 2040 GB. The actual size is restricted by the underlying disk hardware protocol. For example, ATA hard disks have a 127-GB limit.

The basic format of a dynamic hard disk is shown in the following table.

Dynamic Disk header fields
Copy of hard disk footer (512 bytes)
Dynamic Disk Header (1024 bytes)
BAT (Block Allocation table)
Data Block 1
Data Block 2
…
Data Block n
Hard Disk Footer (512 bytes)

Every time a data block is added, the hard disk footer must be moved to the end of the file. Because the hard disk footer is a crucial part of the hard disk image, the footer is mirrored as a header at the front of the file for purposes of redundancy.

Differencing Hard Disk Image

A differencing hard disk image represents the current state of the virtual hard disk as a set of modified blocks in comparison to a parent image. This type of hard disk image is not independent; it depends on another hard disk image to be fully functional. The parent hard disk image can be any of the mentioned hard disk image types, including another differencing hard disk image.

For details about this format, see “Implementing a Differencing Hard Disk” later in this paper.

Hard Disk Footer Format

All hard disk images share a basic footer format. Each hard disk type extends this format according to its needs.

The format of the hard disk footer is listed in the following table.

Hard disk footer fields	Size (bytes)
Cookie	8
Features	4
File Format Version	4
Data Offset	8
Time Stamp	4
Creator Application	4
Creator Version	4
Creator Host OS	4
Original Size	8
Current Size	8
Disk Geometry	4
Disk Type	4
Checksum	4
Unique Id	16
Saved State	1
Reserved	427

Note: Versions previous to Microsoft Virtual PC 2004 create disk images that have a 511-byte disk footer. So the hard disk footer can exist in the last 511 or 512 bytes of the file that holds the hard disk image.

Hard Disk Footer Field Descriptions

The following provides detailed definitions of the hard disk footer fields.

Cookie

Cookies are used to uniquely identify the original creator of the hard disk image. The values are case-sensitive.

Microsoft uses the “conectix” string to identify this file as a hard disk image created by Microsoft Virtual Server, Virtual PC, and predecessor products. The cookie is stored as an eight-character ASCII string with the “c” in the first byte, the “o” in the second byte, and so on.

Features

This is a bit field used to indicate specific feature support. The following table displays the list of features. Any fields not listed are reserved.

Feature	Value
No features enabled	0x00000000
Temporary	0x00000001
Reserved	0x00000002

No features enabled. The hard disk image has no special features enabled in it.

Temporary. This bit is set if the current disk is a temporary disk. A temporary disk designation indicates to an application that this disk is a candidate for deletion on shutdown.

Reserved. This bit must always be set to 1.

All other bits are also reserved and should be set to 0.

File Format Version

This field is divided into a major/minor version and matches the version of the specification used in creating the file. The most-significant two bytes are for the major version. The least-significant two bytes are the minor version. This must match the file format specification. For the current specification, this field must be initialized to `0x00010000`.

The major version will be incremented only when the file format is modified in such a way that it is no longer compatible with older versions of the file format.

Data Offset

This field holds the absolute byte offset, from the beginning of the file, to the next structure. This field is used for dynamic disks and differencing disks, but not fixed disks. For fixed disks, this field should be set to 0xFFFFFFFF.

Time Stamp

This field stores the creation time of a hard disk image. This is the number of seconds since January 1, 2000 12:00:00 AM in UTC/GMT.

Creator Application

This field is used to document which application created the hard disk. The field is a left-justified text field. It uses a single-byte character set.

If the hard disk is created by Microsoft Virtual PC, "vpc " is written in this field. If the hard disk image is created by Microsoft Virtual Server, then "vs " is written in this field.

Other applications should use their own unique identifiers.

Creator Version

This field holds the major/minor version of the application that created the hard disk image.

Virtual Server 2004 sets this value to `0x00010000` and Virtual PC 2004 sets this to `0x00050000`.

Creator Host OS

This field stores the type of host operating system this disk image is created on.

Host OS type	Value
Windows	0x5769326B (Wi2k)
Macintosh	0x4D616320 (Mac)

Original Size

This field stores the size of the hard disk in bytes, from the perspective of the virtual machine, at creation time. This field is for informational purposes.

Current Size

This field stores the current size of the hard disk, in bytes, from the perspective of the virtual machine.

This value is same as the original size when the hard disk is created. This value can change depending on whether the hard disk is expanded.

Disk Geometry

This field stores the cylinder, heads, and sectors per track value for the hard disk.

Disk Geometry field	Size (bytes)
Cylinder	2
Heads	1
Sectors per track/cylinder	1

When a hard disk is configured as an ATA hard disk, the CHS values (that is, **C**ylinder, **H**eads, **S**ectors per track) are used by the ATA controller to determine the size of the disk. When the user creates a hard disk of a certain size, the size of the hard disk image in the virtual machine is smaller than that created by the user. This is because CHS value calculated from the hard disk size is rounded down. The pseudo-code for the algorithm used to determine the CHS values can be found in the appendix of this document.

Disk Type

Disk Type field	Value
None	0
Reserved (deprecated)	1
Fixed hard disk	2
Dynamic hard disk	3
Differencing hard disk	4
Reserved (deprecated)	5
Reserved (deprecated)	6

Checksum

This field holds a basic checksum of the hard disk footer. It is just a one’s complement of the sum of all the bytes in the footer without the checksum field.

If the checksum verification fails, the Virtual PC and Virtual Server products will instead use the header. If the checksum in the header also fails, the file should be assumed to be corrupt. The pseudo-code for the algorithm used to determine the checksum can be found in the appendix of this document.

Unique ID

Every hard disk has a unique ID stored in the hard disk. This is used to identify the hard disk. This is a 128-bit universally unique identifier (UUID). This field is used to associate a parent hard disk image with its differencing hard disk image(s).

Saved State

This field holds a one-byte flag that describes whether the system is in saved state. If the hard disk is in the saved state the value is set to 1. Operations such as compaction and expansion cannot be performed on a hard disk in a saved state.

Reserved

This field contains zeroes. It is 427 bytes in size.

Dynamic Disk Header Format

For dynamic and differencing disk images, the “Data Offset” field within the image footer points to a secondary structure that provides additional information about the disk image. The dynamic disk header should appear on a sector (512-byte) boundary.

The format of the Dynamic Disk Header is listed in the following table.

Dynamic Disk Header fields	Size (bytes)
Cookie	8
Data Offset	8
Table Offset	8
Header Version	4
Max Table Entries	4
Block Size	4
Checksum	4
Parent Unique ID	16
Parent Time Stamp	4
Reserved	4
Parent Unicode Name	512
Parent Locator Entry 1	24
Parent Locator Entry 2	24
Parent Locator Entry 3	24
Parent Locator Entry 4	24
Parent Locator Entry 5	24
Parent Locator Entry 6	24
Parent Locator Entry 7	24
Parent Locator Entry 8	24
Reserved	256

Dynamic Disk Header Field Descriptions

The following provides detailed definitions of the dynamic disk header fields.

Cookie

This field holds the value "cxsparse". This field identifies the header.

Data Offset

This field contains the absolute byte offset to the next structure in the hard disk image. It is currently unused by existing formats and should be set to 0xFFFFFFFF.

Table Offset

This field stores the absolute byte offset of the Block Allocation Table (BAT) in the file.

Header Version

This field stores the version of the dynamic disk header. The field is divided into Major/Minor version. The least-significant two bytes represent the minor version, and the most-significant two bytes represent the major version. This must match with the file format specification. For this specification, this field must be initialized to `0x00010000`.

The major version will be incremented only when the header format is modified in such a way that it is no longer compatible with older versions of the product.

Max Table Entries

This field holds the maximum entries present in the BAT. This should be equal to the number of blocks in the disk (that is, the disk size divided by the block size).

Block Size

A block is a unit of expansion for dynamic and differencing hard disks. It is stored in bytes. This size does not include the size of the block bitmap. It is only the size of the data section of the block. The sectors per block must always be a power of two. The default value is `0x00200000` (indicating a block size of 2 MB).

Checksum

This field holds a basic checksum of the dynamic header. It is a one's complement of the sum of all the bytes in the header without the checksum field.

If the checksum verification fails the file should be assumed to be corrupt.

Parent Unique ID

This field is used for differencing hard disks. A differencing hard disk stores a 128-bit UUID of the parent hard disk. For more information, see “Creating Differencing Hard Disk Images” later in this paper.

Parent Time Stamp

This field stores the modification time stamp of the parent hard disk. This is the number of seconds since January 1, 2000 12:00:00 AM in UTC/GMT.

Reserved

This field should be set to zero.

Parent Unicode Name

This field contains a Unicode string (UTF-16) of the parent hard disk filename.

Parent Locator Entries

These entries store an absolute byte offset in the file where the parent locator for a differencing hard disk is stored. This field is used only for differencing disks and should be set to zero for dynamic disks.

The following table describes the fields inside each locator entry.

Parent locator table field	Size (bytes)
Platform Code	4
Platform Data Space	4
Platform Data Length	4
Reserved	4
Platform Data Offset	8

Platform Code. The platform code describes which platform-specific format is used for the file locator. For Windows, a file locator is stored as a path (for example. “c:\disksimages\ParentDisk.vhd”). On a Macintosh system, the file locator is a binary large object (blob) that contains an “alias.” The parent locator table is used to support moving hard disk images across platforms.

Some current platform codes include the following:

Platform Code	Description
None (0x0)	
Wi2r (0x57693272)	[deprecated]
Wi2k (0x5769326B)	[deprecated]
W2ru (0x57327275)	Unicode pathname (UTF-16) on Windows relative to the differencing disk pathname.
W2ku (0x57326B75)	Absolute Unicode (UTF-16) pathname on Windows.
Mac (0x4D616320)	(Mac OS alias stored as a blob)
MacX(0x4D616358)	A file URL with UTF-8 encoding conforming to RFC 2396.

Platform Data Space. This field stores the number of 512-byte sectors needed to store the parent hard disk locator.

Platform Data Length. This field stores the actual length of the parent hard disk locator in bytes.

Reserved. This field must be set to zero.

Platform Data Offset. This field stores the absolute file offset in bytes where the platform specific file locator data is stored.

Reserved

This must be initialized to zeroes.

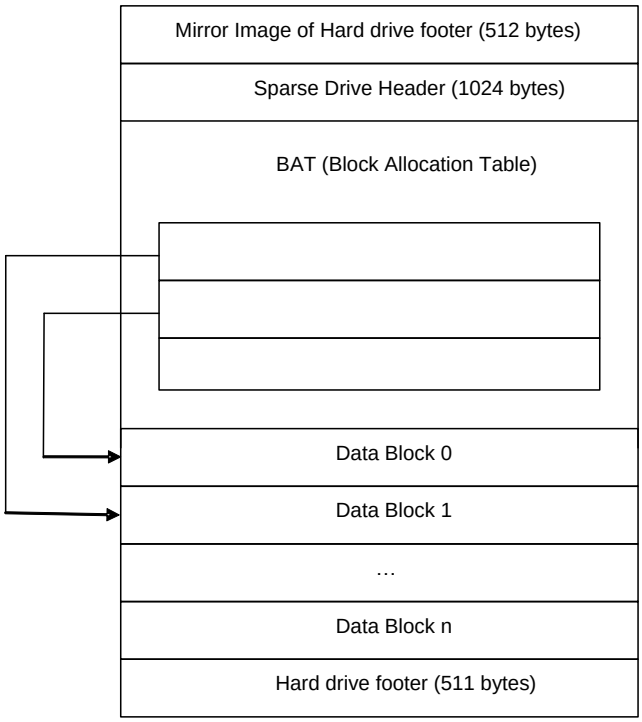
Block Allocation Table and Data Blocks

The Block Allocation Table (BAT) is a table of absolute sector offsets into the file backing the hard disk. It is pointed to by the “Table Offset” field of the Dynamic Disk Header.

The size of the BAT is calculated during creation of the hard disk. The number of entries in the BAT is the number of blocks needed to store the contents of the disk when fully expanded. For example, a 2-GB disk image that uses 2-MB blocks requires 1024 BAT entries. Each entry is four bytes long. All unused table entries are initialized to 0xFFFFFFFF.

The BAT is always extended to a sector boundary. The “Max Table Entries” field within the Dynamic Disk Header indicates how many entries are valid.

Each entry in the BAT refers to a block in the disk image.



The data block consists of a sector bitmap and data. For dynamic disks, the sector bitmap indicates which sectors contain valid data (1’s) and which sectors have never been modified (0’s). For differencing disks, the sector bitmap indicates which sectors are located within the differencing disk (1’s) and which sectors are in the parent (0’s). The bitmap is padded to a 512-byte sector boundary.

A block is a power-of-two multiple of sectors. By default, the size of a block is 4096 512-byte sectors (2 MB). All blocks within a given image must be the same size. This size is specified in the “Block Size” field of the Dynamic Disk Header.

All sectors within a block whose corresponding bits in the bitmap are zero must contain 512 bytes of zero on disk. Software that accesses the disk image may take advantage of this assumption to increase performance.

Note: Although the format supports varying block sizes, Microsoft Virtual PC 2004 and Virtual Server 2005 have only been tested with 512K and 2 MB block sizes.

Implementing a Dynamic Disk

Blocks are allocated on demand. When a dynamic disk is created, no blocks are allocated initially. A newly created image contains only the data structures described earlier (including the Dynamic Disk Header and the BAT).

When data is written to the image, the dynamic disk is expanded to include a new block. The BAT is updated to contain the offset for each new block allocated within the image.

Mapping a Disk Sector to a Sector in the Block

To calculate a block number from a referenced sector number, the following formula is used:

BlockNumber = floor(*RawSectorNumber* / *SectorsPerBlock)*

SectorInBlock = *RawSectorNumber* % *SectorsPerBlock*

BlockNumber is used as an index into the BAT. The BAT entry contains the absolute sector offset of the beginning of the block’s bitmap followed by the block’s data. The following formula can be used to calculate the location of the data:

ActualSectorLocation = BAT [*BlockNumber*] + *BlockBitmapSectorCount* + *SectorInBlock*

In this manner, blocks can be allocated in any order while maintaining their sequencing through the BAT.

When a block is allocated, the image footer must be pushed back to the end of the file. The expanded portion of the file should be zeroed.

Splitting Hard Disk Images

Versions prior to Microsoft Virtual Server 2005 supported splitting of disk images, if the disk image grew larger than the maximum supported file size on the host file system.

Some file systems, such as the FAT32 file system, have a 4-GB limit on file size. If the hard disk image expands more than 4 GB, Microsoft Virtual PC 2004 and previous versions will split the hard disk image into another file. The split files do not have any headers or footers, just raw data. The last split file has the footer stored at the end of the file. The first file in the split disk image has an extension of .vhd. The following split files use the *.v01, .v02, …* filename extension. The split files will be in the same directory as the main hard disk image. The maximum number of split files that can be present is 64.The size of the split file cannot be altered.

Implementing a Differencing Hard Disk

A differencing hard disk stores the file locator of the parent hard disk inside the differencing hard disk itself. When a virtual machine tries to open a differencing hard disk, both the differencing hard disk and the parent hard disk are opened. The parent hard disk can also be a differencing hard disk, in which case there could be a chain of differencing hard disks which finally end in a non-differencing hard disk.

To have the ability to move hard disks across platforms, the hard disk format is designed in such a way that it can store parent hard disk file locators for different platforms at the same time.

The parent locator table is used only by the differencing hard disks, as described in “Dynamic Disk Header Format” earlier in this paper. The parent locator table stores

a platform code for every parent file locator stored in the file. The virtual machine reads the appropriate parent file locator for the current platform and opens the hard disk image.

In Windows, there are two types of platform locators: **W2ku** and **W2ru**. The former is the absolute pathname of the parent hard disk, and the latter is a pathname to the parent hard disk relative to the differencing hard disk.

For example, a parent hard disk image located in the root drive on a typical Windows-based machine would be stored as follows:

Type	Example
W2ku	c:\directory\parent.vhd
W2ru	.\directory\parent.vhd

As an example on a typical Apple Macintosh-based machine, the parent hard disk image would be stored as follows:

Type	Example
Mac	(Mac OS alias stored as a blob)
MacX	file://localhost/directory/parent.vhd

The advantage of the relative pathname is that it allows portability of the differencing and parent hard disk to different locations. With the absolute pathname, whenever the parent hard disk is moved, the parent and child hard disks must be explicitly re-linked.

When a differencing disk is being created, pathnames for both types of platform locators on the respective platforms should be initialized if possible.

Note: Versions previous to Microsoft Virtual PC 2004 only stored the absolute pathnames.

Write Operation for a Differencing Hard Disk

For a write operation, all data is written to the differencing hard disk image. The block bitmap is marked dirty for all the sectors written to the particular block.

Read Operation for a Differencing Hard Disk

When a virtual machine reads sectors of a hard disk image, the differencing hard disk subsystem checks the block bitmap in the differencing hard disk. The differencing hard disk subsystem reads the sectors marked dirty from the differencing hard disk and the sectors marked clean from the parent hard disk.

For example, consider a block that holds sectors 4096 through 8191 in both the parent and child hard disk image. The first sector of the block holds the bitmap for the block. A single cell represents a bit in the bitmap and a black dot represents the particular sector in the block has been written to by the virtual machine.

Parent Block

4096	4097	4098	4099	4100	4101	• 4102	• 4103	• 4104	4105	4106	4107								

Sector 4096
Sector 4097
Sector 4098
Sector 4099
Sector 4100
Sector 4101
Sector 4102
Sector 4103
Sector 4104
Sector 4105 (Clean. Data has been never written to this sector.)
Sector 4106 (Clean. Data has been never written to this sector.
…
…

Child Block

4096	4097	4098	4099	4100	4101	• 4102	• 4103	• 4104	4105	4106	4107								

Sector 4096 (Clean. Data has been never written to this sector.)
Sector 4097 (Clean. Data has been never written to this sector.)
Sector 4098 (Clean. Data has been never written to this sector.)
Sector 4099 (Clean. Data has been never written to this sector.)
Sector 4100 (Clean. Data has been never written to this sector.)
Sector 4101 (Clean. Data has been never written to this sector.)
Sector 4102
Sector 4103
Sector 4104
Sector 4105
Sector 4106
…
…

If the virtual machine issues a read operation for sectors from 4098 to 4104, the differencing hard disk subsystem would read sectors 4098 through 4101 from the parent hard disk block and would read 4102 through 4104 from the child block.

If the virtual machine issues a write operation for sectors from 4102 to 4106, the entire data will be written to the child block and the bitmap will be marked dirty for sectors 4105 and 4106 in the child block.

Identification of the Parent Hard Disk Image

Every hard disk has a UUID stored in the hard disk footer. When a differencing hard disk is created, it stores the UUID of the parent hard disk inside the differencing hard disk. The UUID and the parent disk name is used to recognize the parent hard disk.

Modification of Parent Hard Disk Image

After a differencing hard disk is created for a parent hard disk, the parent hard disk should not be modified. Modifying the parent hard disk invalidates the state of the differencing hard disk. To ensure this does not happen, the parent modification date is stored in the differencing hard disk structure.

Both the parent hard disk UUID and parent hard disk modification date should be checked to ensure a valid parent-child relationship exists.

Appendix: CHS Calculation

The CHS is calculated based on the total data sectors present in the disk image.

CHS Calculation

Variables in CHS calculation	Description
totalSectors	Total data sectors present in the disk image
cylinders	Number of cylinders present on the disk
heads	Number of heads present on the disk
sectorsPerTrack	Sectors per track on the disk
cylinderTimesHead	Cylinders x heads

```
                     C      H     S
if (totalSectors > 65535 * 16 * 255)
{
   totalSectors = 65535 * 16 * 255;
}

if (totalSectors >= 65535 * 16 * 63)
{
   sectorsPerTrack = 255;
   heads = 16;
   cylinderTimesHeads = totalSectors / sectorsPerTrack;
}
else
{
   sectorsPerTrack = 17;
   cylinderTimesHeads = totalSectors / sectorsPerTrack;

   heads = (cylinderTimesHeads + 1023) / 1024;

   if (heads < 4)
   {
      heads = 4;
   }
   if (cylinderTimesHeads >= (heads * 1024) || heads > 16)
   {
      sectorsPerTrack = 31;
      heads = 16;
      cylinderTimesHeads = totalSectors / sectorsPerTrack;
   }
   if (cylinderTimesHeads >= (heads * 1024))
   {
      sectorsPerTrack = 63;
      heads = 16;
      cylinderTimesHead = totalSectors / sectorsPerTrack;
   }
}
cylinders = cylinderTimesHead / heads;
```

Checksum Calculation

Variables in checksum calculation	Description
driveFooter	Variable holding the drive footer structure
checksum	Variable that stores the checksum value
driveFooterSize	Size of the driveFooter structure
counter	Local counter

```
checksum = 0;
driveFooter.Checksum = 0;
for (counter = 0 ; counter < driveFooterSize ; counter++)
{
   checksum += driveFooter[counter];
}
driveFooter.Checksum = ~checksum;
```