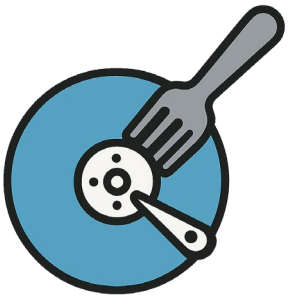
Forkable Virtual Disk (FVD) Specification



Version 1.0 (04/10/2025)

Clarifications and changes since the initial release of this specification are in red.

Copyright

Introduction

This specification describes Forkable Virtual Disk (FVD) image format. It does not explain how hard disks interface with virtual machines, 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 filesystem.

The reader should 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:

Virtual Disk

A virtual device exposed to the user that has all properties of and provides the same interfaces as a physical disk.

Disk Image/Backing File

File(s) which store the data used by a virtual disk, along with metadata. In the case of FVD, a disk image or backing file consists of a disk container file and a refcount metadata file.

Disk Container File

A file consisting of a series of records, to hold information stored in the virtual disk as well as certain metadata.

Refcount Metadata File

A file consisting of a series of unsigned 8-bit integers, storing the number of references each record in the disk container file has.

Sector

Smallest unit of storage of a disk, and what data is addressed in terms of when referring to the virtual disk. Sector length is always 512 bytes in FVD version 1.

Record

Smallest unit of storage in a FVD disk container file, and what data is addressed in terms of when referring to the disk image/backing file. Record length is always 512 bytes in FVD version 1.

Disk Geometry

How data is organised on a disk in terms of cylinders, heads and sectors per track, used for CHS addressing. The size of the disk in number of sectors may be calculated through number of cylinders times number of heads times number of sectors per track.

Refcount

Short for reference count, the number of entities currently using a particular block of data.

Reserved

Field reserved for future use, should not be written to but can be read from, however the data within bears no meaning and may be anything.

Note

All values in the file format, unless otherwise specified, are stored in network byte order (big endian). All numerical values in this specification are in decimal, unless prefixed with 0x for hexadecimal.

FVD Files

An FVD image consists of a disk container file and a refcount metadata file. Together, they are referred to as an "FVD image". The disk container file usually has the extension `.fvd` (though it may use any extension or none), and the refcount metadata file is always the path to the disk container file plus `.ref`. Example:

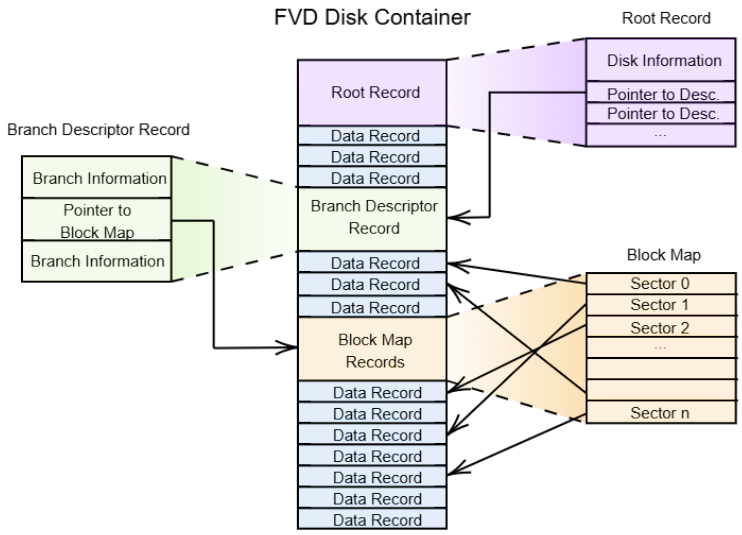
- FVD image: `comp3301.fvd`
    - Disk container: `comp3301.fvd`
    - Refcount metadata: `comp3301.fvd.ref`
- FVD image: `backup`
    - Disk container: `backup`
    - Refcount metadata: `backup.ref`

FVD Disk Container Layout

An FVD disk container file consists of a series of records, where each record has a fixed length of 512 bytes. There are four types of records:

- Root Record
    - The first record of an FVD disk container file, containing information about the virtual disk. There is only one root record per FVD image.
- Branch Descriptor Record
    - There is one branch descriptor record for each branch in the FVD image, containing information about that particular branch.
- Block Map Record
    - There is one Block Map per branch in the FVD image, consisting of a fixed number of Block Map Records stored contiguously.
- Data Record
    - Records for storing disk sector data, not guaranteed to be contiguous and may appear at any location. Each Data Record stores a single sector.

Together, these records form an FVD disk container file:



(Diagram Not Drawn To Scale)

Root Record

A Root Record has the following structure:

Offset	Name	Meaning	Type	Size	Value
0	MAGIC	Magic	uint32_t	4	0x46564449
4	VMAJ	Version (Major)	uint8_t	1	1
5	VMIN	Version (Minor)	uint8_t	1	0
6	NBRCHS	No. Branches	uint16_t	2	1 – 122
8	NRECS	No. Records	uint32_t	4	⩾ 3
12	NSECTS	No. Sectors	uint32_t	4	⩾ 1
16	NCYLS	No. Cylinders	uint32_t	4	1 – 65536
20	NHEADS	No. Heads	uint16_t	2	1 – 16
22	NSPT	No. Sectors Per Track	uint16_t	2	1 – 255
24	BRCHS[122]	Global Branch Index	uint32_t[122]	488	Rec. Number

The MAGIC field must be 0x46564449 in order for a file to be recognised as an FVD disk container file. The VMAJ and VMIN fields must be 1 and 0 respectively for FVD version 1. The NBRCHS field contains the total number of branches within the FVD image, which must be a number between 1 and 122. The NRECS field contains the number of records within the FVD disk container file. The NSECTS field contains the number of sectors within the virtual disk, which is the product of NCYLS, NHEADS and NSPT. NCYLS, NHEADS and NSPT collectively describe the disk geometry in CHS notation, and should be within the limits imposed by the ATA interface (at most 65536 cylinders, 16 heads and 255 sectors).

The BRCHS array contains a list of record numbers, each pointing to a Branch Descriptor Record. Only the first NBRCHS elements of the array are populated, with BRCHS[n] pointing to the Branch Descriptor Record for the n^{th} branch. The first element of the BRCHS array, BRCHS[0], is the **default branch**.

Branch Descriptor Record

A Branch Descriptor Record has the following structure:

Offset	Name	Meaning	Type	Size	Value
0	MAGIC	Magic	uint32_t	4	0x46564449
4	NCHILDS	No. Child Branches	uint16_t	2	0 – 16
6	CTIME	Creation Time	uint64_t	8	Unix Timestamp
14	BLKMAP	Block Map Location	uint32_t	4	Rec. Number
18	PARENT	Parent Branch Descriptor	uint32_t	4	Rec. Number
22	CHILDS[16]	Child Branch Descriptors	uint32_t[16]	64	Rec. Number
86	NAME	Branch Name	char[32]	32	NUL-Term. String
118	RESERVED	Reserved Field	N/A	394	N/A

The `MAGIC` field must be `0x42524348` in order for a record to be recognised as a Branch Descriptor Record. The `NCHILDS` field holds the number of child branches this branch has, which is a number from `0` to `16`. The `CTIME` field holds a Unix timestamp describing when the branch was created (forked from parent). The `BLKMAP` field holds a record number pointing to the starting record of the block map for this branch. The `PARENT` field holds a record number pointing to the Branch Descriptor Record of its parent branch, or `0` if the current branch is the default branch.

The `CHILDS` array is an array of record numbers, each pointing to the Branch Descriptor Record of a child branch. Only the first `NCHILDS` elements of the `CHILDS` array are populated. The `NAME` field is a `NUL`-terminated string holding the branch name, storing up to 31 characters. The default branch always has name `default`, and each branch within an FVD image must have a unique name.

Block Map

There is one Block Map for each branch of an FVD image. The Block Map maps sectors of the virtual disk to data records, similar to indexed allocation used in filesystems (covered in week 9 lecture). Each Block Map consists of a fixed number of consecutive records called Block Map Record, stored in the disk container file.

At a high level, the Block Map can be seen as a consecutive array of record numbers (`uint32_t`), with the array having the same number of elements as the number of sectors in the virtual disk. This allows the record number of sector n to be retrieved through a simple lookup: RecordNumber(n) = Block Map[n].

The number of entries in each Block Map Record is as follows:

$$\mathrm{N}_{\mathrm{Entries}} = \left\lfloor \frac{\text{FVD Record Size}}{\text{Size of Each Entry}} \right\rfloor$$

The number of Block Map Records per Block Map is calculated through the following formula:

$$\mathrm{N}_{\mathrm{Records}} = \left\lceil \frac{\text{No. Virtual Disk Sectors}}{\mathrm{N}_{\mathrm{Entries}}} \right\rceil$$

The location of the map entry for sector S within the Block Map can be obtained using:

$$\mathrm{BlockMapRecord}_S = \text{Block Map Location} + \left\lfloor \frac{S}{\mathrm{N}_{\mathrm{Entries}}} \right\rfloor$$
$$\mathrm{IndexWithinRecord}_S = S \bmod \mathrm{N}_{\mathrm{Entries}}$$

To facilitate with the on-demand space allocation nature of FVD, if a virtual disk sector had never been written to before, it is assumed to contain all binary 0s. For such sectors, they are not physically mapped to any data record but instead use the special record number of `0` in the Block Map to indicate that they are not allocated.

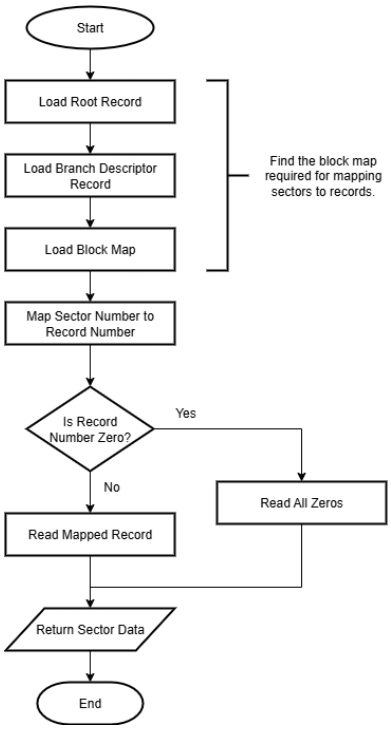
Refcount Metadata File

The refcount metadata file is a file composed of a series of unsinged 8-bit integers (`uint8_t`), with each holding the number of references the corresponding record in the disk container file has. The first 8-bit unsigned integer holds the refcount of the first record, and so on. The size of the refcount metadata file should be the same as the number of records in the disk container file.

The Root Record, all Branch Descriptor Records and all Block Map Records should have refcount of 1. Data Records may have refcounts greater than one if they are referenced by more than one branch and require copy-on-write. A record is unused if the refcount is 0, and data stored within is undefined and may be deleted or overwritten at any time.

Reading Sectors

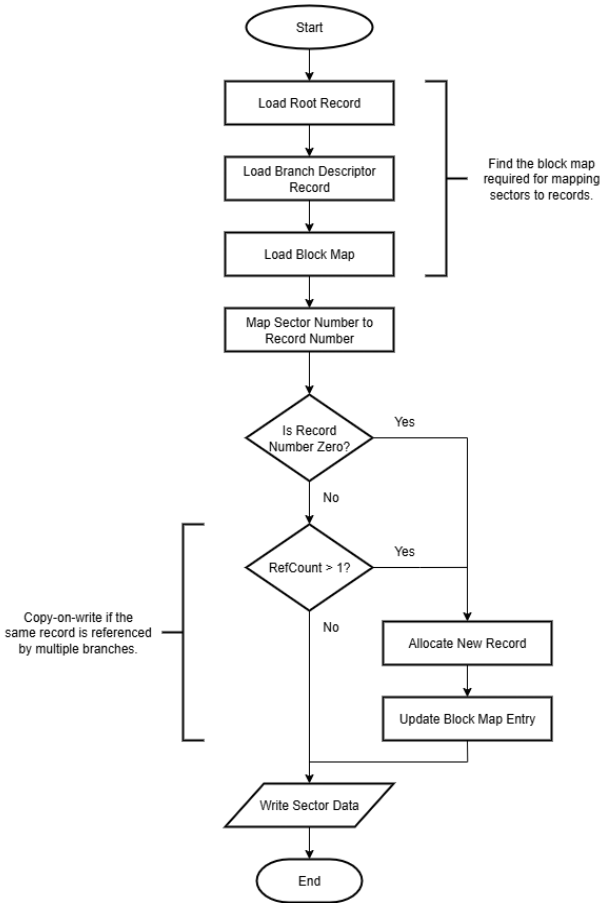
To read a sector from a particular branch:



1. Use the Root Record to locate the Branch Descriptor Record
2. Use the Branch Descriptor Record to locate the Block Map
3. Load the Block Map, or at least the Block Map Record which contains the entry for the requested sector
4. Use the Block Map to map the sector number to record number
5. Check if record number is zero:
    - Yes - the sector is unallocated, read all zeros
    - No – the sector is allocated, read the record pointed to by the record number
6. Return the sector data

Writing Sectors

Writing a sector is more involved than simply reading a sector, as it could involve sector allocation and copy-on-write operations:



1. Use the Root Record to locate the Branch Descriptor Record
2. Use the Branch Descriptor Record to locate the Block Map
3. Load the Block Map, or at least the Block Map Record which contains the entry for the requested sector
4. Use the Block Map to map the sector number to record number
5. Check if record number is zero:
    - Yes - the sector is unallocated, allocate it and update Block Map
    - No – do nothing as the sector is already allocated
6. Check if the refcount of the record is greater than 1:
    - Yes – perform copy-on-write, that is, allocate a new record and update Block Map
    - No – do nothing as there is nothing else using that record
7. Write the data

Hint

The above flowchart and steps only outline the major steps needed to write a sector, and there are minor steps which have been left out intentionally. To score the higher marks, you must ensure that the FVD image after writing is valid and consistent.

Changelog

Date	Version	Description
2025/10/04	1.0	Forkable Virtual Disk Specification 1.0