



# MFT for Linux Installation Instructions

## Table of Contents

- What MFT Does..... 1
- Version Number ..... 2
- Contacting Us for Support of MFT..... 2
- Choice of a Proper Erase Block Size (Read Carefully!)..... 2
- Loading the MFT Software..... 2
- Setup your Flash SSDs. .... 3
- Single Flash SSD..... 3
- Use the Whole SSD Without a Partition Table..... 3
- Use a Partition on the SSD..... 3
- Use a Logical Volume..... 4
- Single Flash SSD Mirrored to a Hard Disk..... 4
- RAID-0 Flash SSD Arrays ..... 4
- RAID-1 Flash SSD Arrays ..... 5
- RAID-5 Flash SSD Arrays ..... 5
- RAID-6 Flash SSD Arrays ..... 5
- RAID-10 Flash SSD Arrays ..... 5
- RAID-50/RAID-60 Flash SSD Arrays ..... 6
- Larger Symetric and Asymmetric Arrays with Hard Disks ..... 6
- Activating MFT on your Linux System ..... 6
- fdm-maint Documentation..... 7
- The dm-fdm.ko Kernel Module ..... 7
- The fdm Startup Script ..... 8
- Actually Start MFT ..... 8
- Create a File System on MFT..... 8
- Create LVM on MFT ..... 8
- Setup fdm to Startup on Boot ..... 8
- Performance Benchmarks..... 8

### What MFT Does

Managed Flash Technology (MFT) is a software solution for both Linux and Windows platforms which accelerates the random write performance of Flash Solid State Disks (SSDs) by converting traditional random writes into clusters of writes which are expressed as linear writes into free space. The new location of these written elements is then preserved both in a memory table and within the metadata of the writes themselves.

This methodology has been shown to improve random write performance on some Flash SSDs as much as 3,000 fold, while also improving the random write life of the device as much as 200 fold. It increases the overall performance of most Flash SSDs by a factor of 10 to 20 while increasing the useful life of these a hundred fold. It also generally improves the useful life and performance of even the best-engineered Flash SSDs significantly, even though these often cost 2x to 3x as much as basic Flash SSDs.

MFT reduces data integrity problems. It writes all data in the order in which it was received: First-in-first-out (FIFO). Similarly, accumulated data is written within a half second of when it is received. Thus, there is no need to have a forced flush process on

shutdown. All MFT does is write more data faster, assuring that data is on disk rather than in the limbo of main system memory. This rapid flushing also often contributes to improved read performance in that more memory is available to handle reusable reads.

## **Version Number**

The version covered by this documentation is [fbd-102108a.tar.gz](#), posted 22 October 2008.

## **Contacting Us for Support of MFT**

We are committed to seeing that customers receive the help they need to install and use MFT. If you need help in installing a copy of MFT we will be happy to provide you that support subject to the limitations specified here.

To get support, you can either write us via email at [support@easyco.com](mailto:support@easyco.com), or you may call us during normal business hours: Monday to Friday, 8am to 6pm Eastern Time, United States holidays excepted. The support phone number is (+1) 610-237-2000. Select the support option when prompted.

Full free support is available to all individuals who have purchased a copy of MFT for one year after the date of new purchase. Free email support is available to beta testers as well as individuals who's copy is more than one year old.

Various support documents including general product documentation and whitepapers can be found online at <http://EasyCo.com> and <http://ManagedFlash.com>.

## **Choice of a Proper Erase Block Size (Read Carefully!)**

MFT improves Flash SSDs because these use the concept of an erase block. Functionally, in most Flash SSDs, this block size is most commonly two megabytes (or 2048 kilobytes), which means that two megabytes of data must be erased and rewritten for every write, even if the write is only four kilobytes in size.

Thus, we use a default erase block size of 2048.

However, due to the nature of controller chip design, 2048 is often not the best value to use. Rather, a larger value produces a write speed that is often six or eight times higher.

Specifically, we have discovered that the best value for drives that use the JMicron controller chip (generally: OCZ, Ridata, SuperTalent, Transcend, Patriot, and some forty or fifty other manufacturers), the best performing value is 16384 (or 16MB).

While some brands such as Mtron have been found to work best at 2048, others work at 8192 (8MB).

Finally, some brands such as Intel use a multiplier of 5 rather than 4 in their write channel designs. This is generally indicated by their odd size which is likely to be 40, 80, or 160 gigabytes. For these, the best factor is likely to be some multiplier of 3060.

## **Loading the MFT Software.**

The MFT software actually uses a driver called "Fast Block Device" or 'fbd'. The MFT software is designed to run from /usr/local/fbd. You can load this from the MFT distribution 'tar file' as:

```
cd /usr/local
tar xvzf /tmp/fbd-083108.tar.gz (your archive name and location may vary)
```

## Setup your Flash SSDs.

MFT is very flexible. You can use just about any block device as the MFT “backing device”. In Linux, this can include hybrid mixtures of SSDs and hard disk drives. Depending on your situation, you may need to pay attention to layouts to achieve the best levels of performance and drive life.

### Single Flash SSD

If you have a single Flash drive, you still have a number of options. You can setup MFT with a single Flash drive using:

- The whole drive without a partition table
- Part of the drive by using a partition table
- Part of the drive by using LVM

It is also possible to mirror a Flash SSDs (or Flash SSD partition or logical volume) with a regular hard drive to create a raid-1 “asymmetric” array.

### Use the Whole SSD Without a Partition Table

This is the easiest case. You can simply format the Flash SSD with:

```
/usr/local/fbd/fbd-maint format /dev/sdc -wtblksz=2048 -quick
```

This example assumes a 2048K erase block, which is a common size for many Flash SSDs. Remember, from the discussion above, that your write block is likely to vary and that the most likely best value will be 16384.

### Use a Partition on the SSD

This is still easy, but you will need to deal with partition alignment. MFT is designed to exactly overlay the erase blocks structure of the SSD. If you miss this alignment, your write performance and drive wearout will degrade about 30%.

The easiest way to do this is to always partition a Flash SSD so that every partition starts on an erase block boundary. With most drives, this means the first partition will start at block 4096 instead of the default of 63. If you partition this way, then sector0 offsets are not necessary. The ‘-u’ option on the fdisk program allows you to enter partitions offsets as sectors instead of tracks. If you don’t map partitions to erase block boundaries, then simple layouts still work, but you need to do some math.

First, you have to find out where the partition starts:

```
fdisk -ul /dev/sdc
```

Look at the starting sector number for the partition your are using for MFT. You then need to tell MFT how many sector to skip so that MFT data starts at an erase block boundary. Lets assume that fdisk shows that the partition starts at sector 123,457:

```
123,457 / 2048 = 60.28 61 * 2048 = 124,928 124,928 - 123,457 = 1,471
```

So in this case we need to skip the first 1,471 sectors before MFT starts storing data. We then format the partition with:

```
/usr/local/fbd/fbd-maint format /dev/sdc3 -wtblksz=2048 -offset0=1471 --  
quick
```

This example assumes a 2048K erase block, which is a common size for many Flash SSDs. Remember, from the discussion above, that your write block is likely to vary and that the most likely best value will be 16384.

## Use a Logical Volume

If you use LVM then the goal is the same as for a partition. LVM usually resides inside of a partition, so first find the starting sector number for the partition. Then use `vgdisplay` to see how the volume group is setup:

```
vgdisplay vg0
```

If the “PE Size” is a multiple of the erase blocks size, then LVM is being kind and you can just use the `offset0` number for the partition as calculated in 2.A.2. Most LVM setups have a 32 MB PE size which is a multiple of a single Flash SSDs erase block size. If LVM uses another size, then you need to see which physical extents are actually in use with:

```
lvdisplay -maps /dev/vg0/fbd
```

If the physical extents occupy a single contiguous range of blocks, then you can take the 1<sup>st</sup> extent number and multiple it by the PE size (expressed in sectors) and then add this number to the partitions starting sector number. The math in 2.A.2 will then tell you what to use in `offset0`:

```
/usr/local/fbd/fbd-maint format /dev/vg0/volname -wtblksz=2048 -  
offset0=1571 -quick
```

This example assumes a 2048K erase block, which is a common size for many Flash SSDs. Remember, from the discussion above, that your write block is likely to vary and that the most likely best value will be 16384.

## Single Flash SSD Mirrored to a Hard Disk

You can setup a “poor man’s” RAID-1 array by mirroring a Flash SSD with a regular hard disk. This is done with Linux’s software RAID-1 driver using the ‘`--write-mostly`’ option. This option will cause all writes to the “device” to mirror to both the SSD and the hard disk but all reads will be serviced by the SSD. This gives you the performance of the SSD with RAID-1 data integrity without having to buy two SSDs. If the SSD crashes, your data is still intact on the HDD and your application will continue to run un-interrupted, albeit a lot slower.

```
mdadm -create /dev/md10 -a yes -level=1 -raid-devices=2 /dev/sdc -write-  
mostly /dev/sda5  
  
/usr/local/fbd/fbd-maint format /dev/md10 -wtblksz=2048 -quick
```

This same method works with partitions and logical volumes located on the Flash SSD, although you will need to supply an ‘`--offset0=nnn`’ to get the erase block to line up in the correct location.

This example assumes a 2048K erase block, which is a common size for many Flash SSDs. Remember, from the discussion above, that your write block is likely to vary and that the most likely best value will be 16384.

## RAID-0 Flash SSD Arrays

All of these examples are using Linux “software raid”. You can use hardware raid as well with MFT. See the “comparison of raid controllers” article for information on hardware vs

software raid. If you are going to run RAID-0 (striped), then you need to either stripe the entire hard drive as in:

```
mdadm -create /dev/md10 -a yes -level=0 -raid-devices=2 /dev/sdc /dev/sdd
```

Or else you need to partition the devices so that the partitions start on exact erase block long boundaries.

```
fdisk -ul /dev/sdc
```

```
/dev/sdc3 starts on 1,822,720 (which is an exact multiple of 4096)
```

### **RAID-1 Flash SSD Arrays**

You can build simple mirrored arrays using either the entire drive or with partitions. If you use partitions, then the partition layout of both drives should be identical. Setting up partitions at erase block boundaries is the easiest solutions:

```
fdisk -u /dev/sdc
```

```
fdisk -u /dev/sdd
```

Create partitions on both drives that are on 4096 sector multiples. You can then setup raid and format the device:

```
mdadm -create /dev/md10 -a yes -level=1 -raid-devices=2 /dev/sdc3  
/dev/sdd3
```

```
/usr/local/fbd/fbd-maint format /dev/md10 -quick
```

### **RAID-5 Flash SSD Arrays**

With RAID-5, you must either use entire drives or setup the partitions on erase block boundaries. It is not possible to use an offset0 value with RAID-5.

```
fdisk -u /dev/sdc
```

```
fdisk -u /dev/sdd
```

```
fdisk -u /dev/sde
```

```
fdisk -u /dev/sdf
```

Create partitions on all drives that are on 4096 sector multiples. You can then setup raid and format the device:

```
mdadm -create /dev/md10 -a yes -level=5 -raid-devices=4 /dev/sdc3  
/dev/sdd3 /dev/sde3 /dev/sdf3
```

```
/usr/local/fbd/fbd-maint format /dev/md10 -quick
```

### **RAID-6 Flash SSD Arrays**

RAID-6 is setup just like RAID-5.

### **RAID-10 Flash SSD Arrays**

RAID-10 arrays work just fine with MFT, with small to medium sized arrays, RAID-10 can actually underperform RAID-5. With large arrays, RAID-10 can perform better, especially for high bandwidth arrays, but at a significantly higher cost.

## RAID-50/RAID-60 Flash SSD Arrays

These work fine as well, but with Linux software raid drivers, flat RAID-5 and RAID-6 arrays tend to perform just as well.

### Larger Symetric and Asymmetric Arrays with Hard Disks

Depending on the devices you have, it might make sense to create a hybrid Flash and hard disk array with more than a simple mirror. For example, you might want to:

RAID-0 4 MLC drives

RAID-0 2 Hard disks

RAID-1 the pair together

When you raid hard disks to Flash SSDs, you want to use a hard disk configuration that has enough write throughput to keep up with the SSDs. In this example 4 MLC drives will write at 40 MB/sec each for a maximum bandwidth of 160 MB/sec. One hard drive is not enough to get to 160 MB/sec, but two high performance drives at 80 MB/sec each will get there. The commands for this are:

```
mdadm -create /dev/md10 -a yes -level=0 -raid-devices=4 /dev/sdc /dev/sdd  
/dev/sde /dev/sdf
```

```
mdadm -create /dev/md11 -a yes -level=0 -raid-devices=2 /dev/sda3  
/dev/sdb3
```

```
mdadm -create /dev/md12 -a yes -level=1 -raid-devices=2 /dev/md10 -write-  
mostly /dev/md11
```

While you can extend this technique, our suggestion is “Don’t get too cute”. If you have one or two Flash SSDs that are going into an existing server with HDDs, then consider an asymmetric RAID-1 array. If you have more drives, just run RAID-5 and keep it simple.

### Activating MFT on your Linux System

Activation of your MFT file system is done using the authorize and activate variants of the fbd-maint program. To generate a string to authorize your MFT copy, use the following command syntax:

```
...fbd-maint authorize {SerialNumber Capacity "Enterprise" {demo} }
```

In the above, the parameters have the following meaning:

SerialNumber is an eight-digit alpha-numeric serial number that you will be provided. Provision is either done by purchase of a permanent license, or by going to <http://EasyCo.com/mft/downloads/index.htm> and requesting a temporary serial number suitable only for 30 day activations.

Capacity is a numerical value indicating the amount of licensed space. This must be equal to or greater than the quantity of MFT net addressable partition space actually created.

“Enterprise” is one of several values which can be entered to indicate the license type you have. The other valid option for Linux platforms is “Single” which is a single-thread version of MFT available at lower cost.

demo is an optional value which can be entered to indicate that the system is being evaluated on a 30 day demonstration basis. If you are using a temporary serial

number, failure to use the demo keyword will prevent you from generating a valid activation code.

The resulting generated activation string might look something like the following:

```
50AQZZ12_12345-67890-ABCDE-FGHIJ-KLMNO_128S.
```

A demonstration string typically would have an additional value in the form “\_D#####” appended. To generate an activation code, go to <http://easyco.com/mft/downloads/index.htm> and select the activation code generation application. Enter the values including the string referenced above. You can avoid re-entering name and address by refreshing the information with the SUBMIT button. Once all data is properly entered, you will be provided with a 20-digit alpha-numeric string which you will use to activate your Linux platform. Activate it with the activate option:

```
...fbd-maint activate { AuthCode }
```

The activate option can also be used to display the activation status of your MFT install. If you do not include an AuthCode, then the current state of the MFT activation is displayed.

## **fbd-maint Documentation**

The /usr/local/fbd/fbd-maint program has the following command-line parameters:

```
...fbd-maint format device -wtblksz=nnnn -offset0=nnn -quiet/-q
```

Perform an MFT logical format on the device.

```
...fbd-maint status device
```

Display the MFT logical format on the device.

```
...fbd-maint authorize {SerialNumber Capacity "Enterprise" {demo} }
```

Generate an activation code. You can send this code to EasyCo to activate your software.

```
...fbd-maint activate { AuthCode }
```

Enter the code you get back from EasyCo. If you do not include an AuthCode, then the current state of the MFT activation is displayed.

```
/etc/mft.lic
```

This is where MFT stored it's license information. Email this to EasyCo if you experience activation problems.

## **The dm-fbd.ko Kernel Module**

In order to actually use MFT, you need to setup the correct version of the dm-fbd.ko loadable kernel module. This version of MFT ships a “partial source” version of MFT. This lets you compile and link “most” of MFT on your server. This can allow MFT to operate with kernels that EasyCo has not specifically targeted. Compiling the dm-fbd.ko kernel module requires that you have:

- The gnu C compiler
- Kernel headers

With RedHat style installations, this usually involves installing the packages ‘gcc’ and ‘kernel-devel’. Make sure that the kernel-devel package exactly matches your running kernel. Once you have the compiler and kernel loaded you can build the dm-fbd.ko module with:

```
cd /usr/local/fbd/obj/date/  
vi Makefile  
set the VER variable to match your configuration  
make install
```

## **The fbd Startup Script**

To facilitate the easy startup of MFT, there is a startup script in /usr/local/fbd/init.d/fbd. You should edit this script to customize it for your particular configuration.

## **Actually Start MFT**

Once everything is setup and activated, you can actually start MFT with:

```
/usr/local/fbd/init.d/fbd start
```

If this completes without error, you can display the device's size with:

```
fdisk -l /dev/mapper/fbd1
```

## **Create a File System on MFT**

Once MFT is started, you can create a filesystem on it and mount it:

```
mkfs.ext3 /dev/mapper/fbd1  
mount /dev/mapper/fbd1 /ssd
```

You can use any type of Linux file system that you wish. While EasyCo does most of it's testing with ext3, we have run MFT in conjunction with ext2, ReiserFS, XFS, FAT32, NTFS, and others. You should also realize that some options that are desirable with standard hard disks are much less important when MFT is in use. For example, the 'noatime' option is commonly used to prevent extra random disk writes. While this is still true with MFT, random disk writes are so inexpensive that the noatime option has little actual impact on an MFT volume.

## **Create LVM on MFT**

You can also setup LVM on top of MFT with:

```
pvcreate /dev/mapper/fbd1
```

If you setup LVM on top of MFT, you will need to setup the LVM rescan in

```
/usr/local/fbd/init.d/fbd.
```

## **Setup fbd to Startup on Boot**

The /usr/local/fbd/init.d/fbd script is designed to go into the Unix Sys-V startup infrastructure.

```
cd /etc/rc.d/init.d  
ln -s /usr/local/fbd/init.d/fbd fbd  
chkconfig -add fbd
```

## **Performance Benchmarks**

EasyCo's benchmark program 'bm-flash' is included in the /usr/local/fbd directory. You can benchmark regular or MFT devices. Warning: bm-flash works only with devices and will destroy the contents of the device. You can use bm-flash as:

```
/usr/local/fbd/bm-flash /dev/mapper/fbd1
```

... or

```
/usr/local/fbd/bm-flash /dev/md10
```

Please consider sharing your benchmarks, along with your hardware configuration with EasyCo.