# MOUNTING A WINDOWS SOFTWARE RAID AS A VIRTUAL DISK

BY

DANIEL N. DUCHARME

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2012





ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

MASTER OF SCIENCE THESIS

OF

DANIEL N. DUCHARME

APPROVED:

Thesis Committee:

Major Professor Dr. Victor Fay-Wolfe

Dr. Gerard Baudet

Dr. Qing Yang

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2012

## ABSTRACT

When an investigator attempts to bring a write blocked Windows dynamic disk online, Windows will refuse to mount it. This forces investigators to use the few tools that have built-in support for dealing with the RAID or to image the partition, and then mount the image. While imaging did not use to be an issue, with the rising sizes of disks available at low cost, it is becoming prohibitively expensive to image every software RAID. The solution is to mount the RAID through the use of a driver as a virtual disk.

The research was conducted by first analysing the Windows Dynamic Disk Logical Disk Manager database for the information needed in order to mount the RAID. Once the important information was identified, a Storport miniport driver was modified in order to mount the RAID after receiving the information. Finally the read function of the driver was designed handle mirrored, simple, spanned, and striped dynamic disks.

Speed results show that the driver achieves speeds between 4-10% slower on average and up to 15% slower when write blocked. The driver has been proven to be compatible with 32 bit Windows Vista, Server 2008 and 7, as well as 64 bit Windows 7 while in test mode. The hashes of the volume show it to be a bit-perfect copy of the Windows implementation, and several different file types were tested and open correctly without modifying the hash. Finally the driver has been tested and functions correctly on spanned, striped, mirrored, and simple RAIDs as well as correctly handling corrupted, linux, or GPT RAIDs when the RAID data was hand entered.

## ACKNOWLEDGMENTS

It is a pleasure to thank those who made this thesis possible. I would like to thank my major professor, Dr. Victor Fay-Wolfe, whose guidance and editing was essential for the writing of this thesis.

This thesis would not have been possible without the help Sean Alvarez and Kevin Bryan both of whom were instrumental in helping to clear up the bugs and getting the front end working.

And most importantly I would like to thank my wife, Tracey Ducharme, who stood besides me these last few years while I pursue my education.

## TABLE OF CONTENTS

**Page**

**Page**

**Page**

## LIST OF TABLES

**Table** **Page**

## LIST OF FIGURES

# CHAPTER 1

# Introduction

## 1.1 Problem

Digital Forensic investigators must ensure that the disks they investigate are unaltered during the investigation process. Furthermore, the investigation process is greatly simplified if disks can be accessed *logically* by having them presented as a logical operating system volume. URI has developed Windows software write blocking for many kinds of disks, but there is a major problem when attempting to block the Windows implementation of dynamic disks (a form of software RAID[1]). When the disks are correctly write-blocked, Windows will not mount them as a logical volume. This leaves investigators currently stuck with two options, either not to write-block the disks, or to use images of the disks and third party software to rebuild a new image of the RAID. The first option is not forensically sound, while the second option demands a special skill set and requires a lot of time and disk space.

Windows dynamic disks are becoming more prevalent as Windows has made it easier to set them up and utilize them. For many people this is the only RAID system they can afford, and they are still powerful enough to handle most companies needs. While this is good news for the consumer, the prevalence of Windows Dynamic Disks can be problematic for digital forensic investigators.

Dynamic disks are handled completely by the operating system (there is no hardware support). Windows makes some assumptions such as: if a user or the system marks the disk read-only, it can still write to the Logical Disk Manager(LDM)[2] database, which is used to map locations in the RAID and is stored outside of the partition. When you attach the drives using a write blocker and attempt to bring the RAID online, Windows will not allow it, simply stating

that DiskPart could not mount the disk. This means that if an investigator follows proper procedure and uses any type of write blocker, hardware or software, Windows will not allow the investigator to mount it. The current practice is: an investigator images the entire dynamic disk pack, then either uses a tool such as WinHex[3] to view the data or RAID Recovery[4] to put the dynamic disk back together so that it can be viewed with Windows Explorer. While this method works, it requires as much storage space as the full dynamic disk pack, which can be multiple terabytes.

## 1.2 Goals

The goal of this project is to create a Windows driver application, called *Software RAID Virtual Disk*, capable of mounting a Windows Dynamic Disk even when the disk is offline, write blocked, or contains a small amount of corruption in its LDM database. The application is being designed from the ground up with law enforcement requirements in mind. In order to accomplish this overriding goal, there are several programs that will need to be designed and implemented. The first is a *Storport* miniport driver which will mount the disk as a virtual drive. The second is an *automated front end* that will mount all of the dynamic disks found in the system automatically. Finally the *manual front end* will allow an investigator to add a RAID as long as he knows the information, circumventing the need to process the LDM database.

The Software RAID Virtual Disk tool will build on the idea of a *virtual drive*[5]. While there are programs that can mount an ISO as a CD in a virtual drive, two open-source programs by VMBack called Virtual Floppy and Virtual Disk[6, 7] are particularly useful. These programs already provide a driver that mounts a virtual drive by modifying the incoming address request to point to the right place in a file. What these programs do not do is to handle a dynamic disk.

These programs will help to illustrate how certain problems in handling Windows Dynamic Disks can be overcome. The final program that Software RAID Virtual Disk builds on is the example *Storport* miniport driver by OSR Online. Their driver is designed to mount a file as a disk, and with a few changes it is capable of mounting a physical drive instead. It handles this through the Windows Storport service, which replaced the old SCSIport service in Windows Server 2003.[8]

## 1.3 Background

As software RAIDs have become more prevalent and disk sizes have increased, law enforcement find themselves seizing handling more software RAIDs. The problem is while proper procedure dictates that all processing be done while the evidence is attached to some form of write blocker (hardware or software), Windows will not process a dynamic disk that it is unable to write to. Up until this point, that has left law enforcement with 2 options: either image the whole RAID and use a tool like RAID Reconstructor to build the RAID into an image, or work with 3rd party tools such as WinHex that are capable of handling the RAID but don't make it accessible for other programs.

### 1.3.1 A Review of RAIDs and Windows Dynamic Disks

As the price of disks have come down and the size of programs and files increased, the prevalence of redundant array of independent disks (hearinafter referred to as a RAID) has increased. These RAIDs can be used for multiple purposes, such as increasing performance or helping secure data even in the face of multiple hard drive failures. The first RAID developed was the RAID 5 which was built as a hardware RAID by the University of California.[9] Developers later managed to duplicate the hardware RAID in software, further driving down the cost and making it available to the common user.

Beginning in the year 2000, Microsoft began offering a easy alternative to the expensive hardware RAID[10]. Starting in Windows 2000 Windows created what they call a dynamic disk, which is software RAID and is now available in Windows 2000, Server 2000, XP, Server 2003, Vista, 7, and Server 2008. There are some restrictions though, such as, only the server operating systems can use either the mirrored or RAID-5 dynamic disks. This has revolutionized RAIDs because while not everyone has RAID controllers (and many of the best controllers are still expensive), any PC user has access to change their normal disks into a dynamic disk and have all of the power of a RAID without most of the cost. All that is required now is a computer with more than one hard drive and a copy of Windows.

#### 1.3.1.1 Spanned Volumes

The first of the popular RAID types is known simply as just a bunch of disks in hardware RAIDs or a Spanned volume in dynamic disks. In this configuration the user is simply attempting to maximize the available volume size with no care to redundancy or performance. As the figure below shows, the data begins on the first disk at A1 and just continues on until it hits the end of the disk. At that point it moves onto the second and continues. In the computer this could look like a 400 gigabyte drive even though it could be made up of four 100 gigabyte drives. Given that hard drive prices are not linear with size, this allows the user to simulate a large hard drive while saving money. However, if a drive dies while in this configuration, all of the data on that disk is lost and as such backups are important.


Figure 1. Just a Bunch of Disks.
[11]

#### 1.3.1.2 Striped Volumes

The second type of RAID that sees common use is RAID-0 or Striped volume as it is called in dynamic disks. This type of RAID attempts to maximize performance but at the cost of redundancy. It does this by striping the data from one disk to the next, so when the user asks for a large file, there is a good chance that it can be found on both disks, allowing both of them to access that file in parallel. Since, in many applications, the hard drive IO is one of the longest tasks, this can have a great effect on speeding up performance. This comes at a price, however, as if even one of these disks dies, the user loses access to all of the data whereas in the spanned volume he would only lose access to the files that were on that disk.

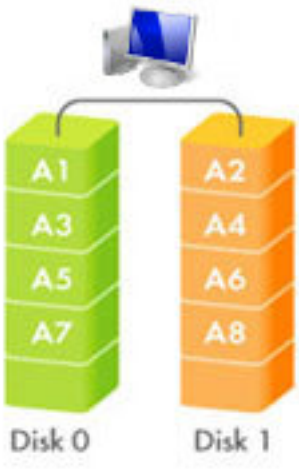



Figure 2. Raid-0

#### 1.3.1.3 Mirrored Volumes

If performance is not as big an issue, but having safe data is, that is where RAID-1 or Mirrored dynamic disks comes in. As the next figure shows, all of the data is not written once, but twice: one time on each of the disks. This means that if one of the disks were to die, you would have lost nothing because the other disk will have an identical copy. The downside is that it takes up twice as much room, and there is no increase in performance. But for applications where data storage is critical, such as a file server, mirrored volumes provide the peace of mind that a dead or dying hard drive won't cost you everything.

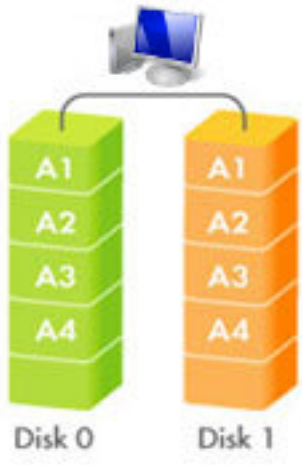



Figure 3. Raid-1

#### 1.3.1.4 RAID-5 Volumes

The last of the popular RAIDs was also the first one developed and is the perfect blend of redundancy and performance. In RAID-5 (which is also its dynamic disk name) the data is striped as it is in RAID-0, however, on one of the three disks, instead of storing more data, it will store what is known as the parity of the data. This parity can be combined with either of the other disks in order to calculate what belongs on the third disk. This parity is also alternated between disks so that it doesn't matter what disk dies, some parity will be lost (which is easily recalculated) and some data will be lost (which is easily recovered using the parity). This means that as long as only one of the three disks is lost, nothing is actually lost. Many of the hardware RAIDs even allow the dead disk to be swapped out while the system is running and will sync the new disk to contain the missing data all without interrupting the work flow. There are downsides to this type of RAID: it requires more disks than any of the others and essentially loses the space of an entire disk to parity.


Figure 4. Raid-5

### 1.3.2 Master Boot Records and GUID Partition Tables

Now that the RAID background has been covered, it is time to move to a lower level on the disks, beginning with the Master Boot Record (or MBR) and the GUID Partition Table (or GPT). The MBR began back with IBM's DOS and has continued through until today. It is a small 512 byte section of code located at the beginning of the disk and contains the boot loader code needed to get the operating system to begin loading. It also contains 4 slots for the primary partition table; although if a disk has more then 4 partitions, one or more of those 4 slots can point to an extended partition table instead. If the disk is a dynamic disk, then it uses volumes instead of partitions, but within the MBR it will still list one partition with the partition type code as 42.[12] This will then instruct the computer to find the Logical Disk Manager (or LDM) located in the last 1 MB of the partition and will contain the information about what volumes are present and what type they are.

The GPT is far less common and normally only used on 64-bit server operating systems. The GPT was developed by IBM in the 1990's as shortcomings in the MBR were beginning to surface. The first such problem was: as drives became

larger, the Cylinder-Head-Sector (CHS) addressing used in the old MBR was no longer capable of addressing all the available space. For this reason the GPT was constructed to use the Logical Block Addressing (LBA) that would later be used in modern MBRs. The other major change was the number of partitions that could be addressed. While MBRs can address more then 4 partitions with the use of the extended partition table, the GPT can address up to 128 in its table. Finally, the GPT also allows for some redundancy by having a backup of the partition table, and if the stored checksum is invalid, it will automatically rewrite the primary partition table with the backup copy.[13] When using a GPT, it is a little different to find the volumes. The volumes themselves are stored in a partition with the GUID AF9B60A0-1431-4F62-BC68-3311714A69AD while the LDM database describing those volumes is in a separate partition with the GUID 5808C8AA-7E8F-42E0-85D2-E1E90434CFB3.

### 1.3.3 Logical Disk Manager

The LDM database is either located in the last 1 MB of the disk in an MBR disk, or a LDM metadata partition in a GPT disk. In either case the LDM contains all of the information needed about all of the volumes on the disk. It also contains information about the disk pack or group of disks combined in the RAID so that if one disk is missing, it can quickly figure out which one it is, and if using a RAID such as mirrored or RAID-5, it can quickly rebuild it onto another free disk.

The LDM database is made up of several different pieces: the TOCBLOCK, the VMDB, the VBLK, the KLOG and the PRIVHEAD. There are three copies of the PRIVHEAD on a disk; the first is located right after the partition table; the second is at the end of the LDM database, and the third is always in the last 512 bytes of the disk. The PRIVHEAD stores the starting location for the database and the number of logs, TOCs, and VBLKs that are present. It also contains the

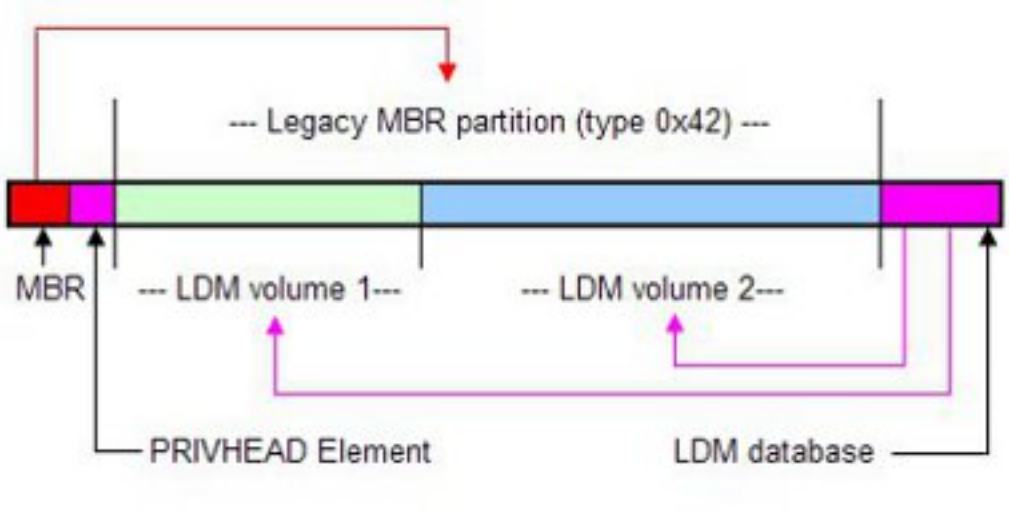



Figure 5. Dynamic Disk Layout

disk group name and the start and size of the logical disk, which is needed to calculate the start of the volumes. The TOCBLOCK simply gives you the start offset and size of the configuration section of the database and the log section. The VMDB contains how many volumes, components, partitions, and disks are in the database. The KLOG stores any changes that are being made to the database so in the event of a failure, it can be rolled back to a consistent state.

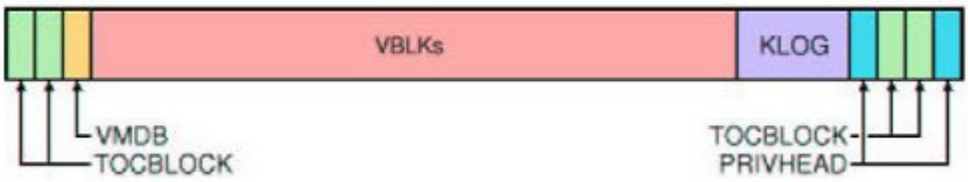



Figure 6. LDM Layout

There are several different types of VBLK and that is where the majority of needed information is stored in order to access the volumes. The first type is the Disk Group VBLK containing the group name and ID, which is only important for establishing which disks belong in the group. The next VBLK is the Volume

VBLK which contains the unique Volume ID identifying it as well as listing if it is currently an active volume. It also contains how many Component VBLKs there are, which is the next important one. The main important part of the Component VBLK is the stripe size which is needed in the event that there is a striped volume. It also contains how many Partition VBLKs there are. The Partition VBLKs are the most important to finding the volumes as they record both the start and the size of each volume. The start is actually an offset from the logical disk start referenced in the PRIVHEAD, but the size is the total size of the volume. It also contains how many Disk VBLKs there are which contain a unique identifier for each disk to help determine what disk is missing.[14]

### 1.3.4 Law Enforcement

Law Enforcement has strict procedures when dealing with a seized drive regardless of if it is in a RAID. The first step to any investigation is to attach the original seized disk to a write-blocker (which will be discussed later) and image the entire disk onto another copy. By separating the original from the copy they will work on, they can guarantee that the original has not been modified and can show the court that they could not have tampered with evidence. This copy is a byte for byte transfer of the data and can be stored in either an image file or be written right to another disk.

In either case once this has been done, the investigator moves on to processing the evidence. This involves using any tools at their disposal to find the evidence, including things such as Windows Search and Windows Explorer. There are far more powerful tools such as the ones Encase and AccessData produces, but it can be time consuming to process the case, and sometimes a quick search with Explorer will give enough evidence in order to take the case to court initially. This is especially true when grabbing a large number of disks from a server or when

executing a subpoena to look for a reason to seize the drives where those powerful tools are not at their disposal.

Regardless of tools used, the officers working the case need to be very diligent in ensuring that no data is changed on any of the disks they are investigating or else the case could get thrown out of court. To ensure that this doesn't happen, they use either a hardware or software write blocker. This is not the same thing as marking a disk as read-only, because while you cannot make changes to a read-only environment, the operating system is actually still making some changes which could potentially change time stamps of when files were accessed and change the hash of the disk.

The hash is the only proof that nothing was changed over the course of the investigation. At the beginning of the investigation, when the copy is being made, either an MD5 or SHA-1 hash is taken of the disk. These hashes pass all of the bytes through a mathematical formula which computes a 128-bit value(if using MD5) or a 160-bit value (if using SHA-1). In either case the change of even a single bit will have large changes in the resulting value; so if at the end of the investigation the hash still matches the one taken at the beginning, then it is pretty conclusive that nothing was altered over the course of the testing.

Up until recently the only method to ensure that the hash could not be changed was through the use of a hardware write blocker. Rather then plug the hard drive directly into the computer, the investigator would instead plug them into the write blocker and then attach that to the computer. To the computer there has been no change, and it still sees the disk as normal. Any read command or normal disk IO command (such as power down) will pass through the write blocker and be executed as if it was not there, but any command that would result in the changing of data (such as a write command) is blocked by the write blocker.

The biggest problem with a write blocker is the hardware needs to analyse every command that goes through, which slows down the commands. They are also expensive and can only be used on a fixed number of disks on a single computer at a time.

Starting in 2005, the University of Rhode Island began programming a software write blocker in-order to keep all of the protection of the hardware methods but to mitigate some of the downsides. They have succeeded and now Safe Block is in use in police stations around the world. Rather than build a large piece of hardware to analyse the commands, Safe Block works by inserting a filter driver directly above the disk driver. Because of the way the Windows driver stack works, they get all commands going to the disk right before the disk driver does. Because this is done in software, it is significantly faster then using a hardware blocker. Also because it is software, you can block as many different disks on the same computer as you want, and only licensing defines how many computers it can be used on at the same time.[15]

### 1.3.5 Windows Device Drivers

The Windows Architecture is made up of 3 portions that are required to work together in order for its proper operation. This discussion will begin with the first and third part first and then finish with the second part where the core of the thesis is located. Then it will delve in depth into the two types of Windows drivers.

The first part of the architecture is the core part of the system which is known as the kernel. There is a different kernel for each version of the operating system, and it performs the most basic operations of the computer. It is the first code that is loaded by the boot loader when the system is started and contains all of the code required in order to interact with the CPU and the bios. It also handles all of the memory management tasks for both the cache and the RAM. Outside of these

tasks it simply contains the required code in-order to run higher level programs known as drivers.[16, 17]

The third level of the operating system is the application layer which exists within an area known as user space. Anything running in user space is considered to be untrusted, while the code in kernel space is trusted, which means that there are a lot of restrictions on what an application can do. For many tasks this means that the applications need to deal with either the Windows Application Programming Interface(API) functions that are available in the kernel, or they need to deal with the drivers. This is the level that most users deal with on a day to day basis and is where all of the programs that contain user interaction are located. For the Software RAID Virtual Disk, the Graphical User Interface(GUI) that allows a user/investigator to find the RAID information and issue the command to mount the drive exists completely in the application layer and uses an API in order to pass the required information down to the next level in the discussion.

The second level of the operating system is the Windows device drivers, which exists in both the user space and kernel space. These drivers are designed in order to add functionality to the core kernel and allow the computer to interact with hardware that could not have been programmed at the time the kernel was written. These allow a user to install any new graphics card into a compatible system, and after simply installing the driver, to utilise all of the new API functions and hardware functions of the device.

#### 1.3.5.1 Windows Driver Model

The first drivers were created for Windows 95 and allowed for developers to extend the operability of the kernel. These initial drivers were built on the Windows Driver Model(WDM) framework which has evolved over the years and is present in every version of Windows after Windows 95[18]. Under this framework

the user could write bus, function or filter drivers, and Windows provided base port and bus drivers that could be called from within another WDM driver.

While this framework is quite powerful and does allow for a wide variety of device support, it also did not age well and needed to be replaced due to a large number of issues[19]. The first problem that developers ran into was that the WDM framework was designed to be very low level which led to a high level of complexity that had to be dealt with. Over time as more functionality got added to the framework, this complexity only got worse until it became quite tedious to develop new drivers. For instance simply supporting plug and play as well as power management could take over 2000 lines of code and support for hundreds of states. This got even worse if your driver was supposed to be multifunctional, making already complex drivers even more difficult to deal with.

The second major problem was that Microsoft had not anticipated third-party driver development and as such all of the driver interfaces were being exported directly from the kernel bypassing many of the security elements being integrated into newer operating systems. Most importantly this led to many of the common errors in the drivers to crash the system since they were not correctly separated from the kernel. This also was the reason for a lack of version support which meant that the developer had to create a different binary for every version of Windows that the developer supported which made the debugging process far longer than it needed to be.

The last big problem was the development of a number of different miniports in an attempt to make driver development easier. These miniports shield some of the operating system requirements, and while they are simpler to program then the full WDM drivers, the number of choices make it difficult to know when one should be used, and in many applications more than one is needed which means

needing to know how each of them works.

#### 1.3.5.2 Windows Driver Foundation

After working with WDM for years Microsoft finally designed a new framework when they came out with Windows 2000 which has persisted through Windows 7. This foundation simplified much of the coding required, allowing more developers to write faster and higher quality code. They also designed it so there was a clear separation between what was running in user space and what is in kernel space, which allows the developer to write certain drivers so that even were the driver to fail, the system could continue. Furthermore the system was also designed to be somewhat backwards compatible so that WDM drivers can still be developed and will continue to run on a newer system when required, but there are now options as to how the driver is programmed. Because of the number of features the new foundation offers the Software RAID Virtual Disk was built upon WDF.

There are two different levels within WDF. The first exists in user space, while the second exists within kernel space. The purpose of the user mode drivers is to provide easy accessibility and support to applications for filter drivers that don't need to be at as low a level as a kernel mode driver. These drivers exist as Component Object Model(COM) based Dynamic Link Libraries(DLLs) and are capable of performing many of the functions required by applications. Furthermore, because these drivers exist in user space, they don't have access to the kernel memory space and as such when they fail, the system is easily able to recover versus the kernel mode drivers, which will crash the system if an error is encountered. The problem with user mode drivers is that they are restricted in the different devices they can use. They are fine for USB drivers, display adaptors and other portable devices; however, they are not capable of interacting with the actual hardware and they don't have the ability to handle interrupts.

Drivers that exist within kernel space have access to all of the kernel memory as well as the kernel data structures. This allows them to create filter, function and bus drivers just like the older WDM foundation drivers. These drivers can directly interact with the hardware when necessary and have access to all of the underlying kernel APIs, allowing for them to handle interrupts and create uninterruptable sections of code. Because of this it is important that only trusted kernel drivers are loaded onto the system. For the Software RAID Virtual Disk, a kernel mode driver was required in order to interact with the Storport miniport as well as handling access to the underlying disk.

#### 1.3.5.3 Virtual Disk Service

There are several different methods in order to mount a disk and interact with a disk within a KMDF driver. You can use the Virtual Disk Service(VDS), the SCSI port or the Storport miniport. First to be discussed is VDS and then a discussion on the Storport miniport.

When the Windows Driver Foundation first came out with Windows 2000 there was only one option. Hard drive vendors had to install and manage their own applications and that meant the user also needed an application for every different storage medium that was attached to the computer. This changed with Windows 2003 when Microsoft introduced the Virtual Disk Service[20]. The VDS has two different providers that allow for the required functionality. First there is a hardware provider which is written by the vender of the hardware and exposes the APIs required in order to get the expected functionality. Second there is a software provider which allows the operating system and applications to utilize the functionality without the need to know the underlying hardware. This separation allows the operating system to manage a large number of different pieces of hardware without the need for hardware vendors to write a complete driver and

application for management.

VDS also heavily integrates with tools such as the volume shadow service(VSS) which keeps a copy of all of the important files on the system, so that a user can roll back the system to a time when it was stable if there are any problems. It also integrates with the logical disk manager(LDM)(the provider of dynamic disks and volumes). This is the service that the Software RAID Virtual Disk is meant to emulate.

#### 1.3.5.4 Storport Miniport

When Windows Server 2003 came out the developers realized that they needed to extend the existing SCSI port in order to get better performance in both throughput and system resources used for high-performance buses and RAID adaptors[21]. While the SCSI port was capable of handling all of the current buses, Windows Server 2003 introduced newer high-performance buses that required better performance at every level. Furthermore the newer Storport miniport is compatible with almost all of the original SCSI port drivers with only minimal changes. The only exceptions are when the device is missing features such as plug and play.

When Windows Vista SP1 and Windows Server 2008, came out Microsoft extended the Storport miniport even further with the Virtual Miniport which allows the drivers to implement disks that don't have underlying hardware[22]. This allowed for programs to begin using the same drivers and tools that existed for physical disks in order to utilize other things such as mounting files as a disk. This functionality is exactly what was needed for the Software RAID Virtual Disk, as it will allow the user to create a virtual disk that will act exactly as a regular disk acts, but it is not tied directly to the underlying hardware, allowing multiple disks to be combined to create this virtual disk.

### 1.3.6 Programming Language Considerations

When it came to choosing a language for programming the Software RAID Virtual disk, there were no real choices for the back end, but there were plenty of choices to make on the front end of the program. The WDF not only defines a group of specifications on how to write a driver and what functionality is available, it also states that the drivers must be written in C or C++.

#### 1.3.6.1 C++ Programming

The C++ language was designed in the early 80's by AT&T Bell Laboratories and Bjarne Stroustrup in order to add object oriented functionality to the C language[23]. It was designed to supersede the C language and to continue all of the original functionality by building upon the already established language. It took almost 10 years before the first standard came out, but it has been very succesful since then, showing up in almost every internet browser, many operating systems, and many other popular applications. The language has a high performance, but due to its complexity it can also be difficult in order to debug.

Most UMDF drivers are written in C++ utilizing the languages COM functionality in order to accomplish its tasks. KMDF drivers on the other hand are primarily written in C, but unlike most C programs they are still ended in .cpp because the C++ compiler of the Windows Driver Kit has much better error checking then the C compiler.

The front end program had far more options on what languages to be designed in and changed several times over the development of the program. The front end began as a C++ program, as the example from OSR Online contained the functions required to communicate with the driver already completed in C++. This was also the language I had the most experience in so the program was developed as a command line application. While this program worked throughout

the development phase, it was ill suited for the finished product as it could not correctly handle a large number of disks or allow a user to manually enter in information when the LDM database was corrupted.

#### 1.3.6.2 Visual C# and .NET Programming

The C# language was designed by Microsoft starting in 1997 in response to Sun Microsystems suing Microsoft over their use of the Java language in J++[24]. In the year 2000 Microsoft announced and released both the new C# language and the new Visual.NET development studio[25]. These brought Microsoft back to the forefront of programming languages and allowed it to compete with web languages such as Java.

The .NET programming framework introduces several new features to the visual languages to help them compete with the other solutions on the market[26]. The first is the use of a Common Intermediate Language(CIL) which is similar to the way Java compiles into bytecode. This code is completely platform neutral and will run the same on any computer that has the .NET platform installed, which now starts installed on all Windows operating systems after Vista. Microsoft has included installers for any other version of Windows that the user wishes to run a .NET program on, but it is still only possible to run it on Windows computers at this time; although, there is a Linux project called Mono trying to bring the .NET platform to the Linux operating systems[27].

The other big features that the .NET programming framework introduces are the Common Language Runtime(CLR) and the Common Type System(CTS), which allows a developer to write portions of the code in different languages that will all work together seamlessly to create the final project. This is because all of the major types that are implemented as external functions are all compiled to the same .NET system types regardless of language. They also are all compiled

into very similar CIL which allows for each of them at runtime to function almost identically. There is no requirement for the internal functions of a class to conform to the standard, however, which means that certain languages are easier to do certain functions then others. One example is F# which is a functional language that compiles into .NET. While it has the ability to handle object-oriented programming, it is also very bad at it.

It is for all of these reasons, along with the ease of programming a graphical user interface(GUI), that motivated the switch from C++ to C# for the front-end of the Software RAID Virtual Disk. The GUI code itself was written by Sean Alvarez and was based heavily on the original C++ command line code. The underlying code that reads the LDM database was simply converted from its C++ code to the corresponding C# equivalents without being modified in function.

**List of References**


[1] Wikipedia, "Raid — wikipedia, the free encyclopedia," 2011, [Online; accessed 13-December-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=RAID&oldid=465701597

[2] Zero Assumption Recovery. "Ldm / dynamic disks basics." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.z-a-recovery.com/art-dynamic-disks.htm

[3] X-Ways. "Winhex." [Online; accessed 13-December-2011]. Mar. 2010. [Online]. Available: http://www.winhex.com/winhex/

[4] Runtime Software. "Raid recovery for windows v1.01." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.runtime.org/raid-recovery-windows.htm

[5] Wikipedia, "Disk image — wikipedia, the free encyclopedia," 2011, [Online; accessed 13-December-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Disk_image&oldid=465613179

[6] K. Kato. "Virtual floppy drive 2.1." [Online; accessed 13-December-2011]. Feb. 2008. [Online]. Available: http://chitchat.at.infoseek.co.jp/vmware/vfd.html#top

[7] K. Kato. “Virtual disk driver version 3.” [Online; accessed 13-December-2011]. Apr. 2005. [Online]. Available: http://chitchat.at.infoseek.co.jp/vmware/vdk.html#top

[8] OSR Online. “Writing a virtual storport miniport driver.” [Online; accessed 13-December-2011]. Sept. 2009. [Online]. Available: http://www.osronline.com/article.cfm?article=538

[9] Kroll Ontrack. “Raid: History and information.” [Online; accessed 30-September-2011]. [Online]. Available: http://www.ontrackdatarecovery.co.uk/data-recovery-articles/raid-history-information/

[10] Microsoft. “What are dynamic disks and volumes?” [Online; accessed 13-December-2011]. Mar. 2003. [Online]. Available: http://technet.microsoft.com/en-us/library/cc737048(v=ws.10).aspx

[11] EUSSO Technologies, Inc. “4-bay sata gigibit network terabank nas.” [Online; accessed 30-September-2011]. [Online]. Available: http://www.eusso.com/Models/NAS/USS4500-RS4/USS4500-RS4.htm

[12] Eindhoven University of Technology. “List of partition identifiers for pcs.” [Online; accessed 1-October-2011]. [Online]. Available: http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

[13] Wikipedia, “Guid partition table — wikipedia, the free encyclopedia,” 2011, [Online; accessed 1-October-2011]. [Online]. Available: //en.wikipedia.org/w/index.php?title=GUID_Partition_Table&oldid=452606816

[14] R. Russon, “Home - ldm documentation,” 2002.

[15] ForensicSoft. “Software write blockers.” [Online; accessed 1-October-2011]. 2010. [Online]. Available: https://www.forensicsoft.com/sb_features.php

[16] The Linux Information Project. “Kernel definition.” [Online; accessed 13-December-2011]. May 2005. [Online]. Available: http://www.linfo.org/kernel.html

[17] P. Orwick, *Developing Drivers with the Windows Driver Foundation*. One Microsoft Way, Redmond, Washington 98052-6399: Microsoft Press, 2007.

[18] Microsoft. “Windows driver model (wdm).” [Online; accessed 9-March-2012]. Apr. 2002. [Online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463453

[19] Microsoft. “Introduction to the windows driver foundation.” [Online; accessed 9-March-2012]. Oct. 2003. [Online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463316

[20] Microsoft. "What is virtual disk service?" [Online; accessed 9-March-2012]. Mar. 2003. [Online]. Available: http://technet.microsoft.com/en-us/library/cc778187(v=ws.10).aspx

[21] Microsoft. "Storport driver." [Online; accessed 9-March-2012]. Feb. 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff567541(v=vs.85).aspx

[22] Microsoft. "History of storport." [Online; accessed 9-March-2012]. Feb. 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557249(v=vs.85).aspx

[23] cplusplus.com. "History of c++." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.cplusplus.com/info/history

[24] J. Kovacs. "C#/.net history lesson." [Online; accessed 9-March-2012]. Sept. 2007. [Online]. Available: jameskovacs.com/2007/09/07/cnet-history-lesson/

[25] CSharp-Online.NET. "C# overview." [Online; accessed 9-March-2012]. [Online]. Available: en.csharp-online.net/CSharp_Overview#A_Brief_History_of_C.23

[26] A. Troelsen, *Pro C# 2010 and the .NET 4 Platform, Fifth Edition.* 233 Spring Street, New York, New York 10013: Apress, 2010.

[27] Novell. "Cross platform, open source .net development framework." [Online; accessed 9-March-2012]. [Online]. Available: www.mono-project.com/Main_Page

# CHAPTER 2

## Review of Literature

In this chapter other potential solutions to the problem at hand will be analysed as well as the other work that was used. These tools all solve the problem at least partially, but in each case there is some inadequacy that renders the problem still unsolved.

### 2.1 EnCase

Of all of the software that can handle a dynamic disk, none handles it as well as *EnCase* by Guidance Software[1]. EnCase Forensic v7 comes with a Physical Disk Emulator that allows an investigator to mount the evidence file as a virtual disk in the same way as the Software RAID Virtual Disk. It can natively handle both hardware and software RAIDs including dynamic disks.[2]

In order to handle a RAID, the investigator must use EnCase to acquire the evidence. EnCase will then write the disks to a built-in case file. Once the case file has been created, EnCase will display the volume and allow the investigator to process the evidence from within EnCase. In order to use tools outside of EnCase, the investigator can simply right click on the drive under entries and select "Mount as Emulated Disk", and it will mount the disk as a virtual drive.[3]

There are a few issues in the way that EnCase handles the drive. First, if the investigator accidentally mounts a drive that is physically attached to the system, Windows will crash. Second, EnCase only works from the case file, which means that first the investigator must add the drives to EnCase and image them, at which point it will create a case file greater than the size of the disks, since it also stores the checksum and some metadata. Finally, EnCase caches and writes to the disk, and then allows it to be saved to the case file. This process suffers because if an

investigator is not careful he/she can actually alter the evidence.

## 2.2 Linux NTFS

The second related application is an NTFS driver which was being developed for Linux to support dynamic disks.[4, 5] The documentation for this project clearly shows how the LDM database of the dynamic disk is formatted as well as the locations of the private header and the LDM database. This information is invaluable as it outlines the amount of information stored in each of the LDM records that the Software RAID Virtual Disk must parse out. The Linux driver itself is not useful for a Windows platform nor was it designed to meet forensics requirements. It implements both reading and writing to the disks as well as requiring that the LDM database and MBR be in a consistent state. Thus, the driver itself is not applicable for the Windows Software RAID Virtual Disk tool, but as a guide to the LDM data structure, it is very valuable.

## 2.3 ProDiscover

Another forensic tool capable of handling dynamic disks is ProDiscover Forensics by Technology Pathways. It can handle all of the Windows dynamic disks and, unlike EnCase, does not force a case file to be created. The biggest problem with ProDiscover is that the RAID can only be analyzed within the program. However, it can create an image file that can be used by other programs that handle those.[6, 7]

## 2.4 RAID Recovery for Windows

RAID Recovery for Windows is a new tool by Runtime Software (the creators of RAID Reconstructor) with the sole purpose of recovering NTFS-formatted hardware and Windows-software RAIDs. It is capable of handling RAID 0 and RAID 5 and can take either the physical disks or image files as input. As output it provides

an image file of the logical RAID, which can then be analyzed with other tools. The problem with this program is similar to ProDiscover: the only programs that can utilize the RAID after it is reconstructed are programs capable of handling image files.[8]

### 2.5 SMART Linux

SMART Linux is a Linux boot disk that is capable of imaging a software RAID or rebuilding a RAID from image files. It is one of the few programs that is capable of handling RAID 4 and is capable of figuring out what the RAID header information is even if the LDM is corrupted. Its biggest problem is, like Linux NTFS, it is built on Linux and most investigators are more comfortable working with Windows. Another problem is that, like many of the other programs described above, the only programs that can utilize the RAID after it is reconstructed are programs capable of handling image files.[9, 10]

### 2.6 X-Ways Forensics

X-Ways Forensics and WinHex, both by X-Ways Software Technology AG, can handle virtually mounting RAIDs in RAM, allowing instant access to the RAID. This makes WinHex much faster than any of the other programs reviewed. It can handle the RAID in image files or physical disks and can either be used to do the full analysis or to create an image file. If a disk is missing from the RAID 5, WinHex is also capable of reconstructing the RAID from the parity information. However, like ProDiscover, it is better at handling the RAID within the program.[11, 12]

| Features | SRVD | EnCase | Linux NTFS | ProDiscover |
|---|---|---|---|---|
| Works in Windows | Y | Y | N | Y |
| Reads Physical Disks | Y | Y | Y | Y |
| Forced to make a disk image | N | Y | N | N |
| Mounts the disk | Y | Y | Y | N |
| Allows the use of third party tools | Y | Y | N | N |
| Simple Dynamic Disk | Y | Y | Y | Y |
| Spanned RAID | Y | Y | Y | Y |
| Striped RAID | Y | Y | Y | Y |
| Mirrored RAID | Y | Y | Y | Y |
| RAID 5 | N | Y | Y | Y |
| Protects from writes | Y | N | N | Y |

| Features | SRVD | RAID Recovery | SMART Linux | X-Ways Forensics |
|---|---|---|---|---|
| Works in Windows | Y | Y | N | Y |
| Reads Physical Disks | Y | Y | Y | Y |
| Forced to make a disk image | N | Y | N | N |
| Mounts the disk | Y | N | Y | N |
| Allows the use of third party tools | Y | N | N | N |
| Simple Dynamic Disk | Y | N | Y | Y |
| Spanned RAID | Y | N | Y | Y |
| Striped RAID | Y | Y | Y | Y |
| Mirrored RAID | Y | N | Y | N |
| RAID 5 | N | Y | Y | Y |
| Protects from writes | Y | Y | N | N |

Table 1. Feature Comparison

### 2.7 Linux-NTFS Project

The Linux NTFS project was designed to bring the Windows dynamic disks to Linux in order for them to enjoy the same functionality and to allow for users to

run both operating systems on the same system and still have access to the same data[4]. This project was implemented in a Linux driver which was useless for this program, but they also heavily documented their research which was pivotal in being able to parse the very complicated data structure of the LDM database.

### 2.8 Virtual Storport Miniport Driver

The final program that Software RAID Virtual Disk builds on is the example virtual Storport miniport driver by OSR Online. Their driver is designed to mount a file as a disk and provides for both read and write operations. With a few modifications it became capable of mounting a physical drive although more changes were required in order to properly mount the RAID. This sample driver was designed with all of the generic functions and communication methods already defined so that only the important functions that operate differently on a RAID from a file needed to be modified.[13]

### List of References


[1] Digital Intelligence. "Encase forensic v7." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.digitalintelligence.com/software/guidancesoftware/encase7/

[2] GuidenceSoftware. "Encase forensic." [Online; accessed 1-October-2011]. 2011. [Online]. Available: http://www.guidancesoftware.com/forensic.htm

[3] "Encase version 6.12 modules manual," Guidance Software.

[4] R. Russon, "Home - ldm documentation," 2002.

[5] Linux-NTFS. "Linux-ntfs." [Online; accessed 1-October-2011]. Feb. 2009. [Online]. Available: http://www.linux-ntfs.org/doku.php

[6] "Prodiscover forensics," pdf, Technology Pathways, Aug. 2009.

[7] Technology Pathways. "Prodiscover forensics." [Online; accessed 13-December-2011]. 2010. [Online]. Available: http://www.techpathways.com/prodiscoverdft.htm

[8] Runtime Software. "Raid recovery for windows v1.01." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.runtime.org/raid-recovery-windows.htm

[9] S. D. Dickerman, "Raid rebuilding," pdf, 2007.

[10] ASR Data. "Smart linux." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.asrdata.com/forensic-software/smart-linux/

[11] "X-ways forensics/winhex," pdf, X-Ways, 2011.

[12] X-Ways. "Winhex." [Online; accessed 13-December-2011]. Mar. 2010. [Online]. Available: http://www.winhex.com/winhex/

[13] OSR Online. "Writing a virtual storport miniport driver." [Online; accessed 13-December-2011]. Sept. 2009. [Online]. Available: http://www.osronline.com/article.cfm?article=538

# CHAPTER 3

# Methodology

## 3.1 Computer Setup

There are several phases of testing that were done on different machines. All of the tests, outside of the speed testing and hashing of the spanned and striped RAIDs, were done in a virtual machine running in VMWare Workstation[1] version 8.0.2 build-591240 running on an ASUS N61J-XV1[2] laptop with 8 GB of ram and an i7 processor. The speed testing was done on a Dell OptiPlex 760[3] with an Intel Core 2 Duo E7300[4] running Windows 7 Enterprise SP1[5]. The disks for the RAID were setup through an Adaptec AHA-2940U/W SCSI-3 Controller[6] and the 2 disks were 36.7 GB Quantum Atlas10k2-TY367L[7] SCSI drives.

### 3.1.1 Hardware Configuration

Hardware setup on the physical system was quick and simple, first install the Adaptec SCSI card and attach the 2 Quantum Atlas SCSI drives. Next go to Adaptec's Driver Download page[1]and download AIC78xx and AIC78U2 Driver for Windows 7 x86 and Server 2008R2 x86. Once that is done, restart the computer and then go to Start, right click on Computer, and select Manage. Select Device Manager, expand Disk drives, and ensure that the two Quantum Atlas drives are in the list.

### 3.1.2 Disk Configuration

There are several different configurations of disks that can be accomplished. Most of them are very similar, and as such, I will walk through how to configure a Striped software RAID on an MBR disk and will just mention the other options.

[1]The download can be found at http://www.adaptec.com/en-us/downloads/ms/ms_win_7/productid=aha-2940uw&dn=aha-2940uw.html

All of these options assume that you are running on any version of Windows 7 though they are similar in other versions of Windows. First, regardless of what options you are going to be selecting, all of the options begin the same, click on Start and then Right click on Computer and select Manage.

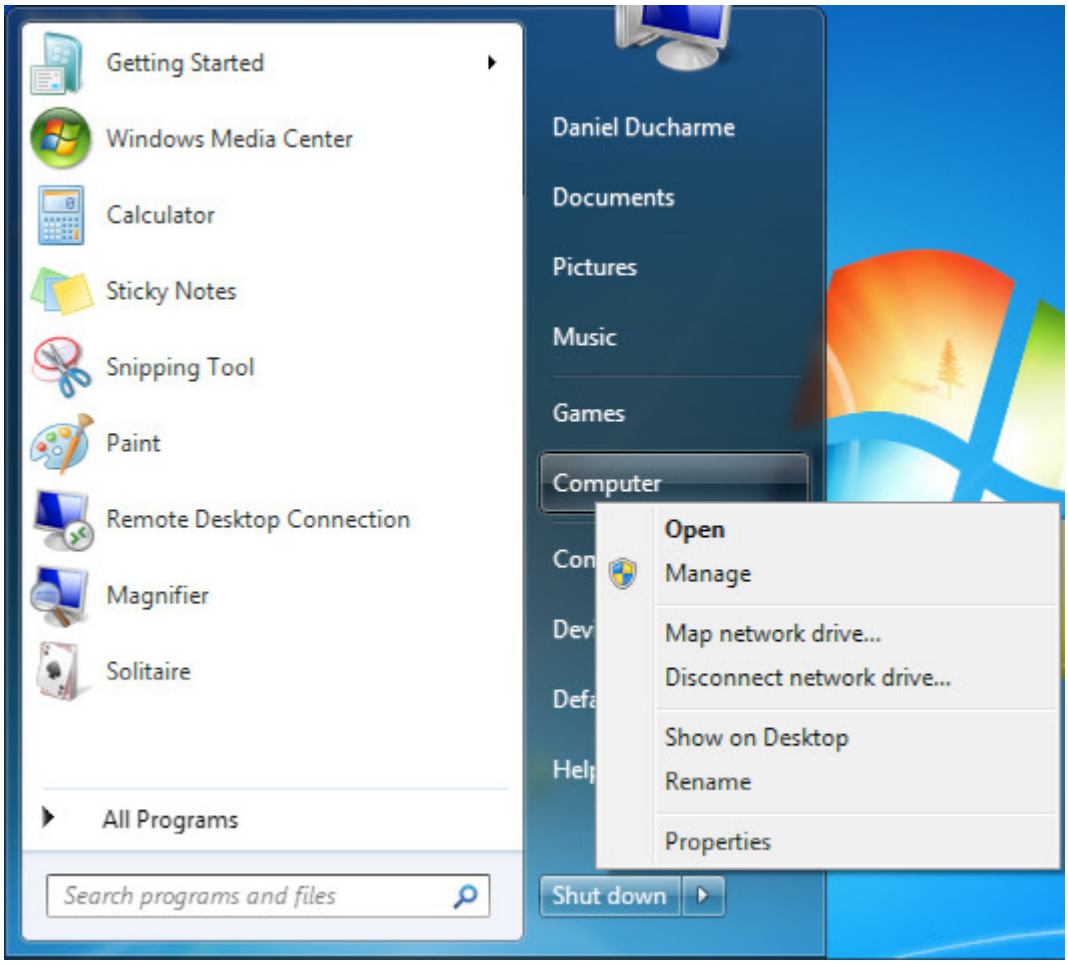



Figure 7. Disk Configuration Start Menu

This will open the Computer Management dialog box where the Disk Management tool under Storage should be selected.

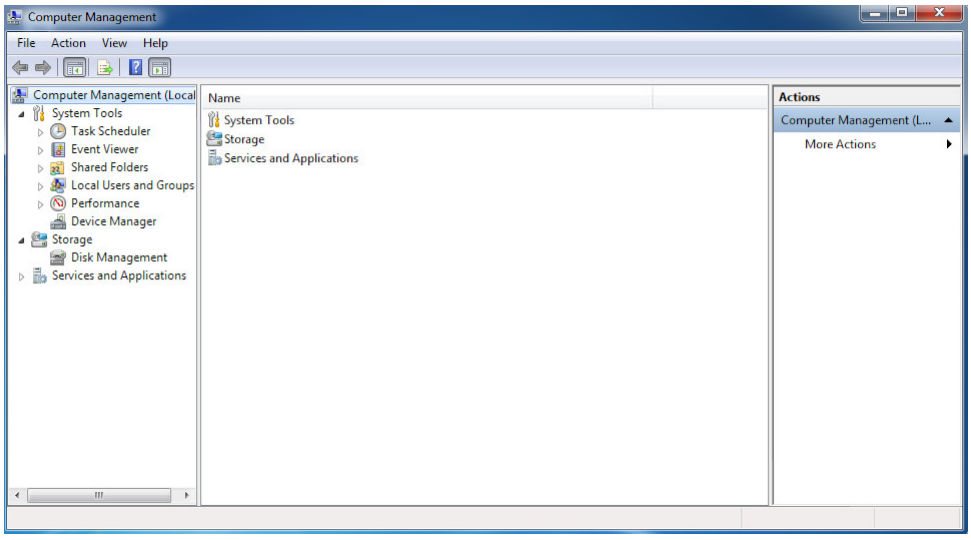



Figure 8. Disk Configuration Computer Manager

Since you should have just added 2 new disks that have not yet been configured, a pop-up box as seen in the next figure should show up. If this does not show up, don't worry, it just means that the disks are already initialized. For most of the tests, the defaults will be used and you can just hit OK. If, on the other hand, you wish to replicate the GPT test, then ensure that the GPT radial is selected. If you wish to test the GPT functionality later, then you can convert it so don't worry. When you are satisfied, just press OK and move onto the next step.

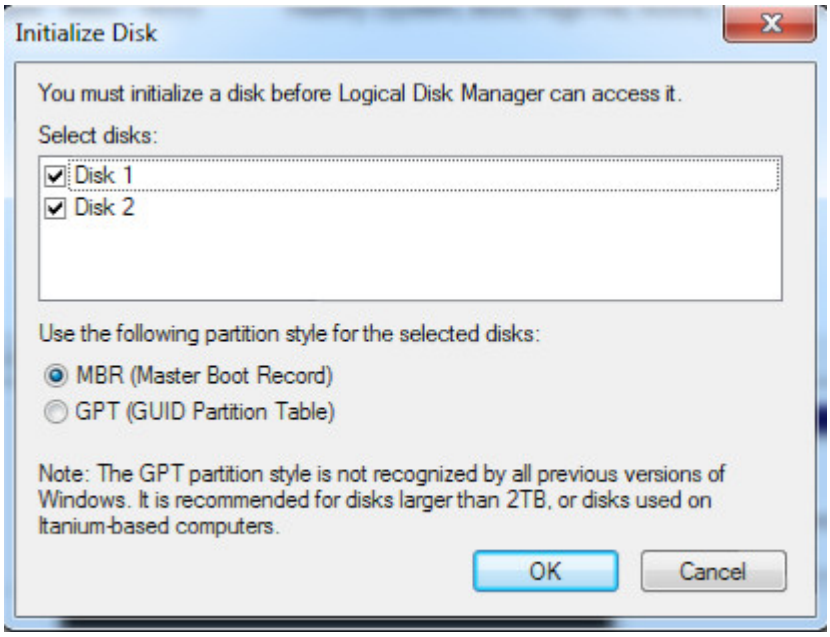



Figure 9. Disk Configuration Initialize Disk

Once the disks have been initialized, right click on one of the new disks, and you will get the choices shown in the figure below. This is where you need to decide what type of RAID you are going to be testing. For the purposes of this setup the New Striped Volume will be chosen, however, the next steps will be identical with any of the choices; only the underlying RAID will be different as well as the eventual hard drive size.

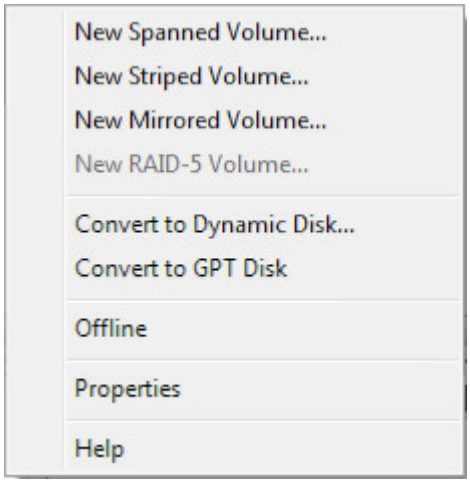



Figure 10. Disk Configuration Disk Selection

After making your selection, you will be shown the following screen. Just press next to begin setting up the dynamic disk.

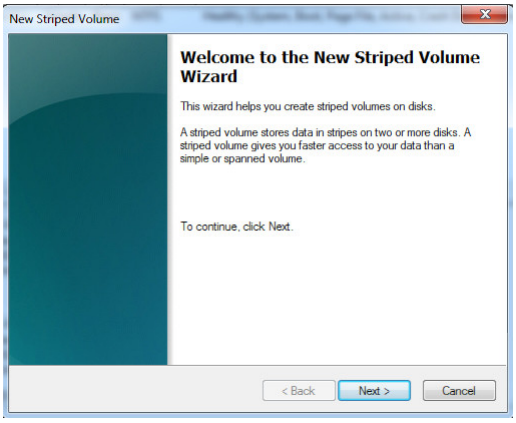



Figure 11. Disk Configuration Striped Volume Main Menu

The next screen will show you any disks that have free space available and that can be converted to a dynamic disk. For the purposes of testing, the only 3 disks were the main disk with the operating system and the two test disks, so the only options available will be the two test disks. In any case select any disks you want to be in the RAID, and click add to ensure they are in the right pane. Make sure there are at least 2 disks in right pane, and select how much space you want to use on each disk; for this testing we simply selected the entire disk. Once you are satisfied, click next.

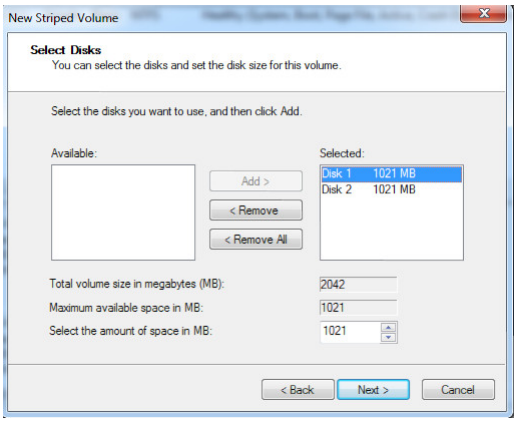



Figure 12. Disk Configuration Striped Volume Main Disk Selection

The options on the next screen don't really matter, we can just leave the defaults selected. Just remember where you mount it if you change it, in order to put any files onto the disk. When you are satisfied, just hit next.

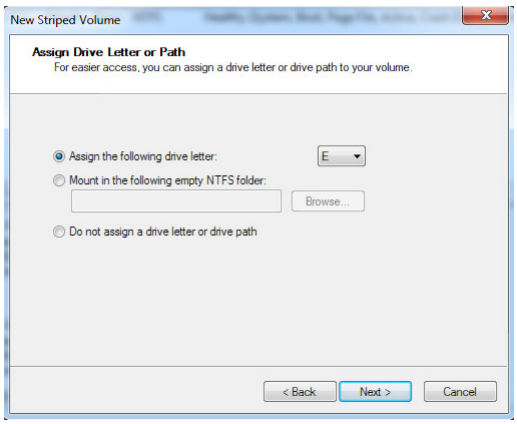



Figure 13. Disk Configuration Striped Volume Mount Point

Just leave the defaults alone on this page. While this will work with any file system that Windows will understand, we can simply leave it as NTFS for testing as the driver does not deal with file systems. Just click on Next to move on.

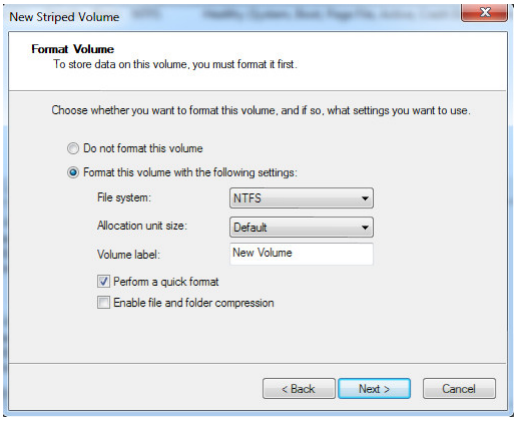



Figure 14. Disk Configuration Striped Volume Formatting Options

Finally, double check to make sure that all of the options you selected are correct, then click finish.

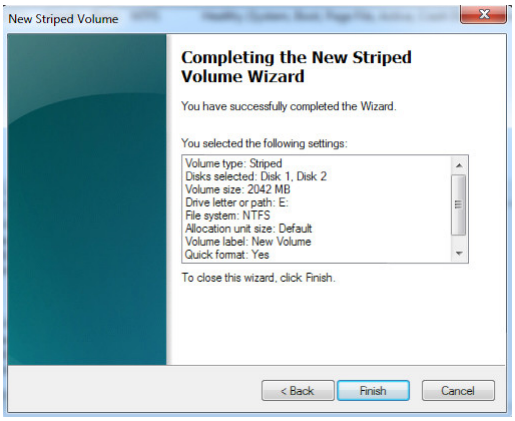



Figure 15. Disk Configuration Striped Volume Confirmation

A warning should pop up warning you that the disk will be converted to a dynamic disk. This is expected and is just warning you that you should not be doing this on the operating system disk or the system will be unable to function. Just press Yes, and it will finish the setup.

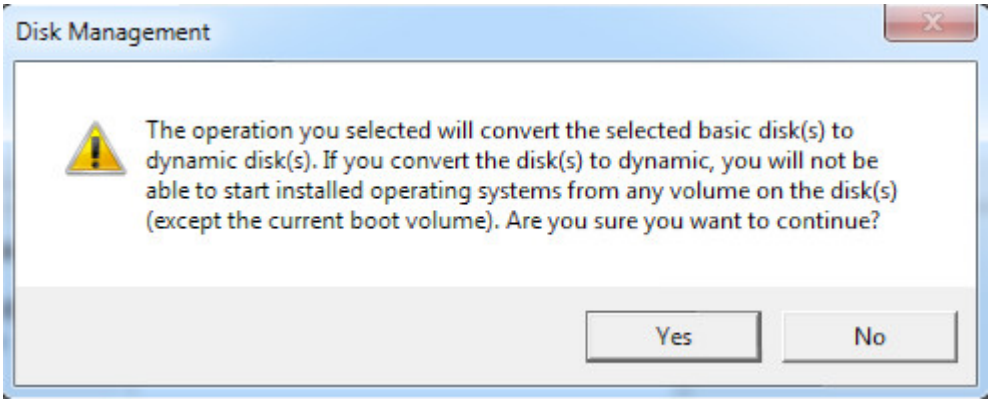



Figure 16. Disk Configuration Striped Volume Main Menu

To confirm that the disk was setup, just open up Computer, and check that there is a new volume on the system mounted to the letter you specified. As long as you can open it up, then this setup is complete, and you can move on.

### 3.1.3 Software Installation

There are two separate programs that need to be installed in order for the Software RAID Virtual Disk to work.

There is no automated installer for the driver installation, and as such it needs to be done manually. This process is slightly different for each version of Windows so the instructions will be included for each, though it will begin with the Windows 7 x86 instructions. Please note that this driver will only run on Windows Vista and Windows 7 as it requires the Storport service introduced in Vista.

#### 3.1.3.1 Driver Installation on Windows 7 x86

First open the start menu, type in run, and select the executable at the top called Run.

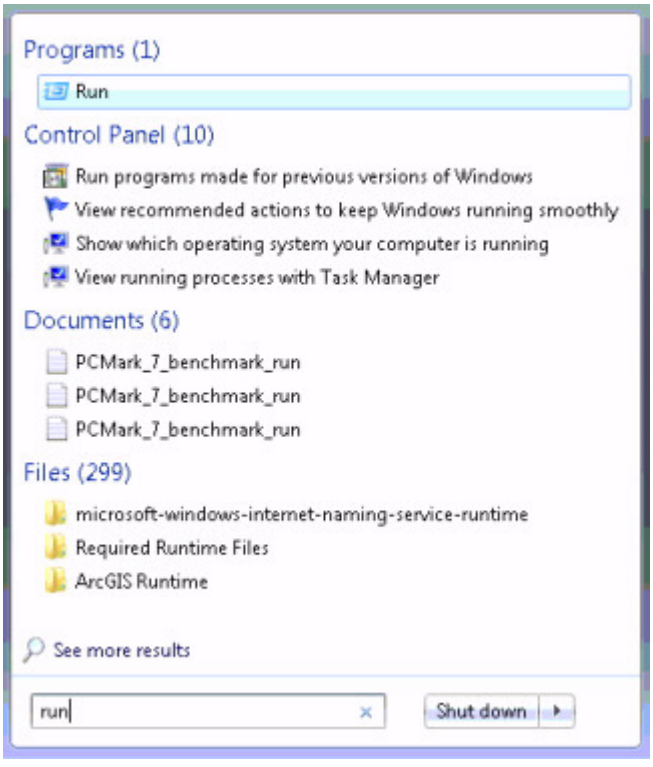



Figure 17. Windows 7 x86 Start Menu

This will open the Run dialog box where you need to type in hdwwiz as shown

in the figure below in order to start the hardware wizard.

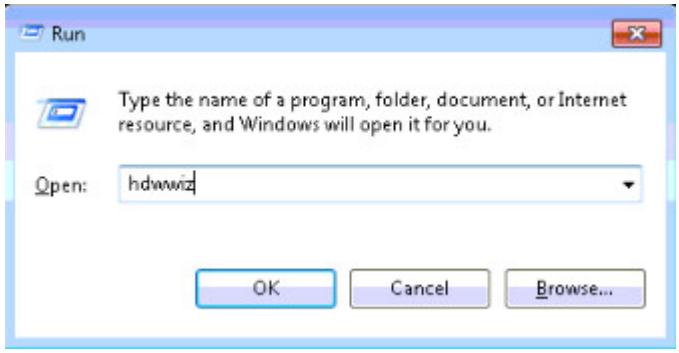



Figure 18. Windows 7 x86 Run Dialog

When the wizard starts, you will be presented with the screen seen in the following figure. Just click Next to move on to the next screen

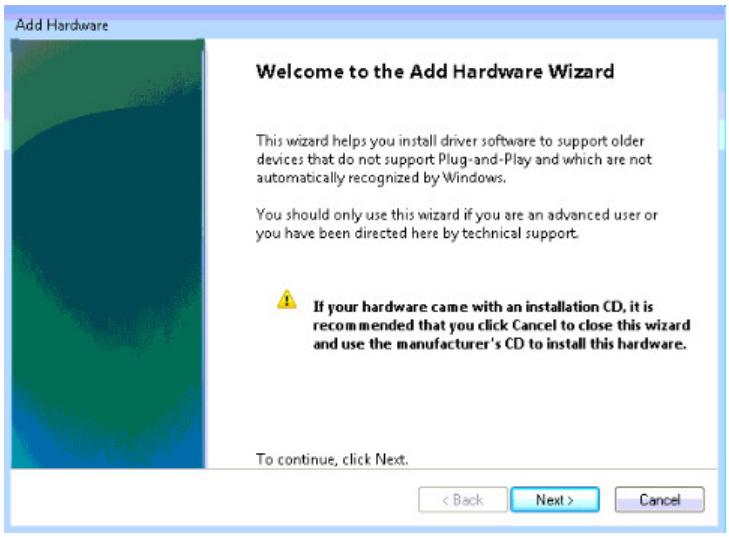



Figure 19. Windows 7 x86 Hardware Wizard Starting Screen

On the next screen, make sure you select the second radial in order to choose the installation file since no real hardware was added to the system.

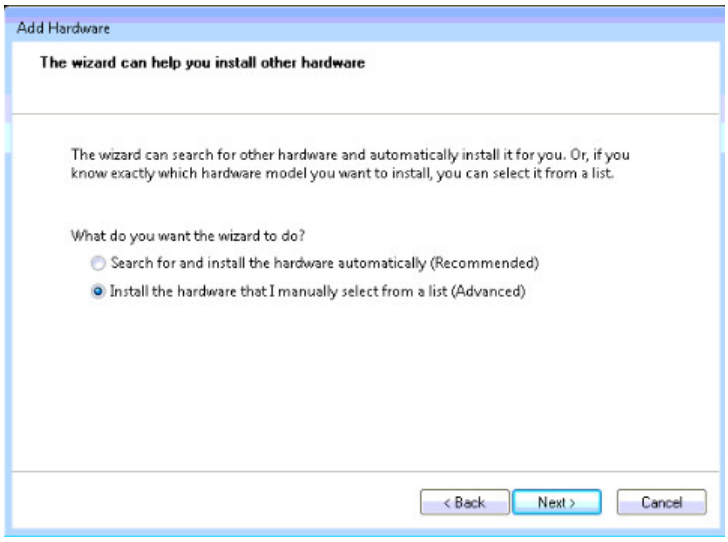



Figure 20. Windows 7 x86 Hardware Wizard Select Advanced

Because the driver we are installing is not for actual hardware but is instead a virtual storage controller, we simply select to Show All Devices because we are going to supply the location of the driver anyway.

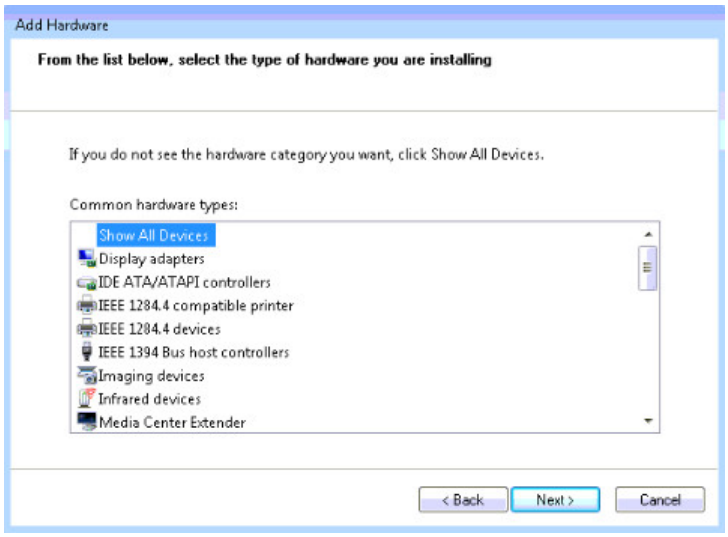



Figure 21. Windows 7 x86 Hardware Wizard Select Show All Devices

On the next page, just click the Have Disk... button which will allow you to specify the location of the inf file.

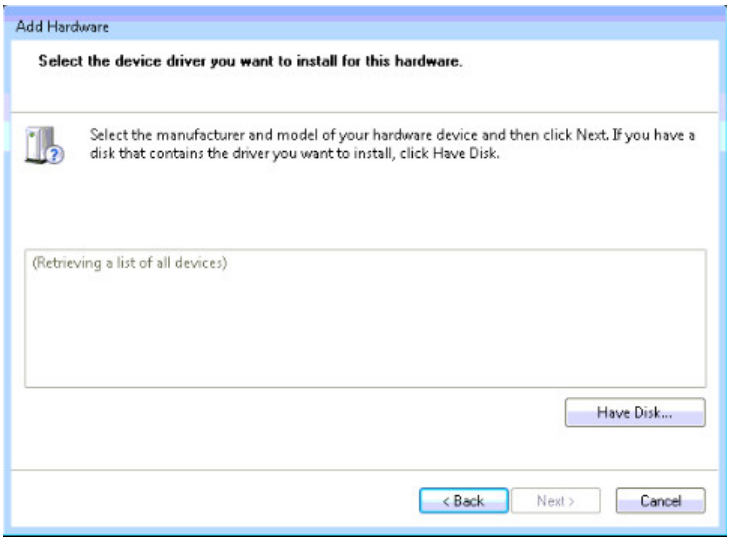



Figure 22. Windows 7 x86 Hardware Wizard Select the Devices

Browse to the location you have stored the inf file and installation files (uriSRVDstor.sys), and select the inf file. Note the inf file should be just outside of the i386 folder as seen in the figure below.

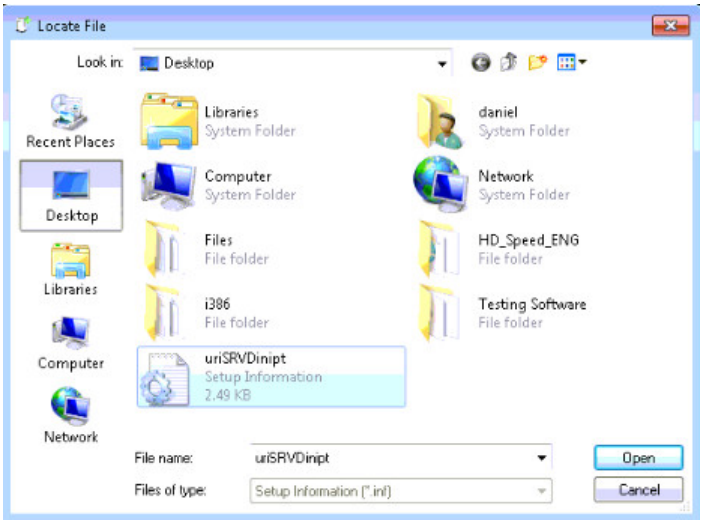



Figure 23. Windows 7 x86 Hardware Wizard Select uriSRVDinipt.inf

Just make sure that the path is correct, then select OK.

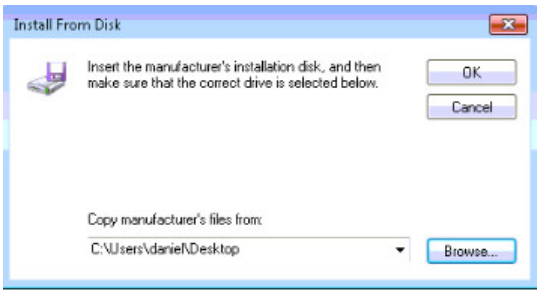



Figure 24. Windows 7 x86 Hardware Wizard Confirm the Path

The next screen should look exactly like the figure below, as long as it does, simply click Next. Otherwise make sure you are selecting the correct inf file and that the i386 folder is at the same location.

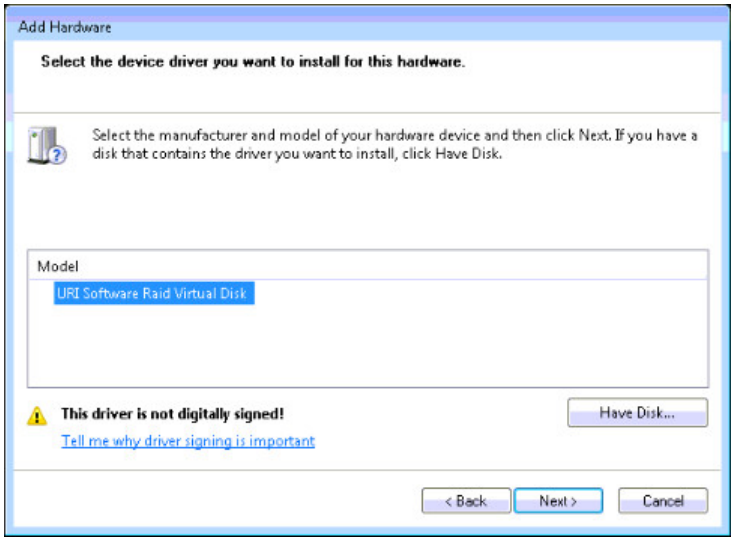



Figure 25. Windows 7 x86 Hardware Wizard Ensure the Correct Model

Finally it will just confirm the device you want to install, just click next, and the install will commence.

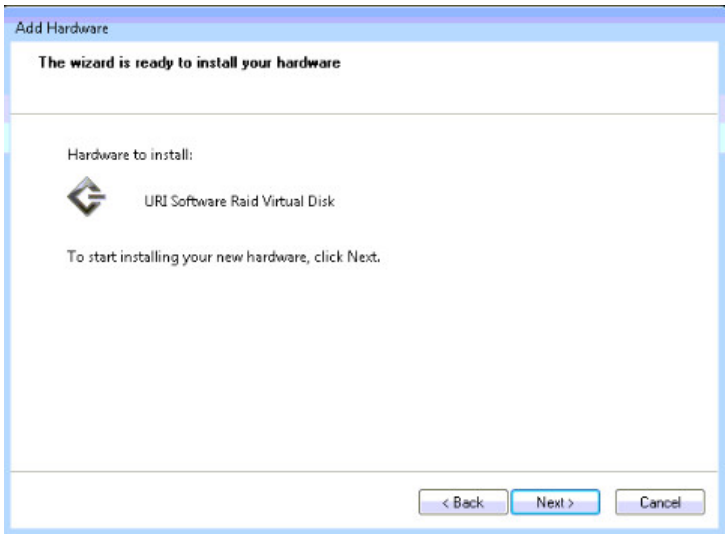



Figure 26. Windows 7 x86 Hardware Wizard Confirm the Install

After you confirm the install, you will get a Windows Security message seen in the figure below. This is because the driver is not signed by Microsoft which is expensive and difficult to pass. Because of this just select to Install this driver software anyway, and the installation will continue.

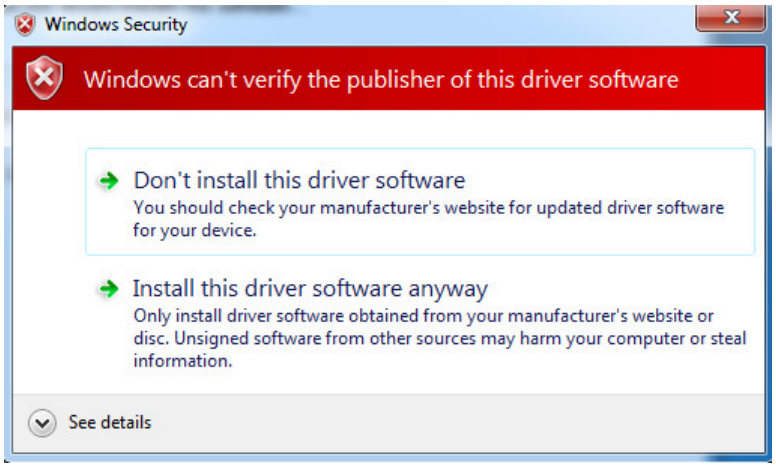



Figure 27. Windows 7 x64 Windows Security Warning

The install should not take too long, and once it is complete, you will get the figure below. If the dialog says anything other than what is shown below, then the installation was not successful.

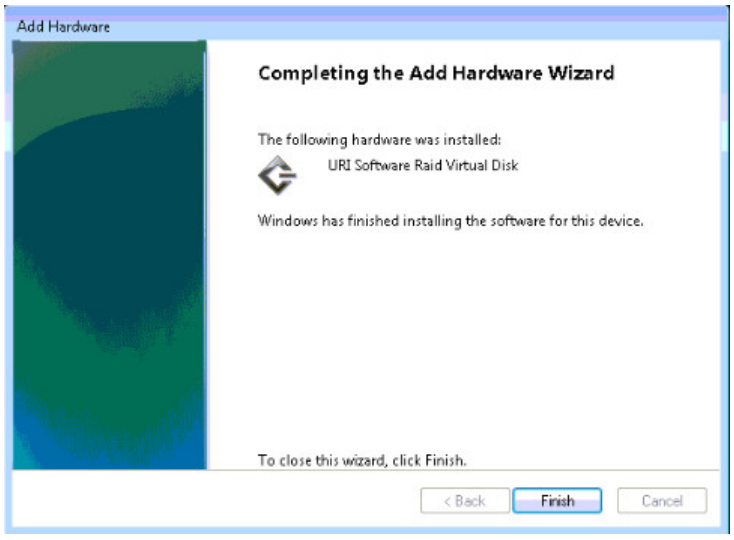



Figure 28. Windows 7 x86 Hardware Wizard Install Complete

To confirm that the install worked, open up the control panel, and select Device Manager. Expand the Storage Controllers section, and ensure that URI Software Raid Virtual Disk is in the list[2].

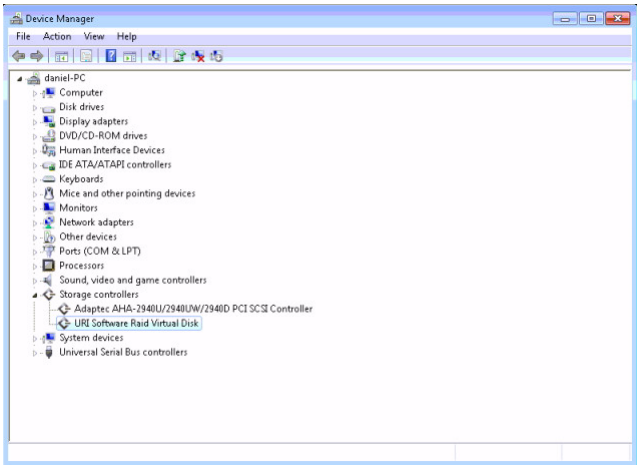



Figure 29. Windows 7 x86 Device Manager

#### 3.1.3.2 Driver Installation on Windows 7 x64

The installation on a 64 bit system is almost identical to the 32 bit one with only a few exceptions. Because of the increased driver security on a 64-bit system,

[2]Note: Your list may look different from the one shown depending on other hardware and software installed on the system.

the hardware wizard completion will warn you that the file is incorrectly signed and may not work correctly. As long as it looks like the figure below, continue to the next step.

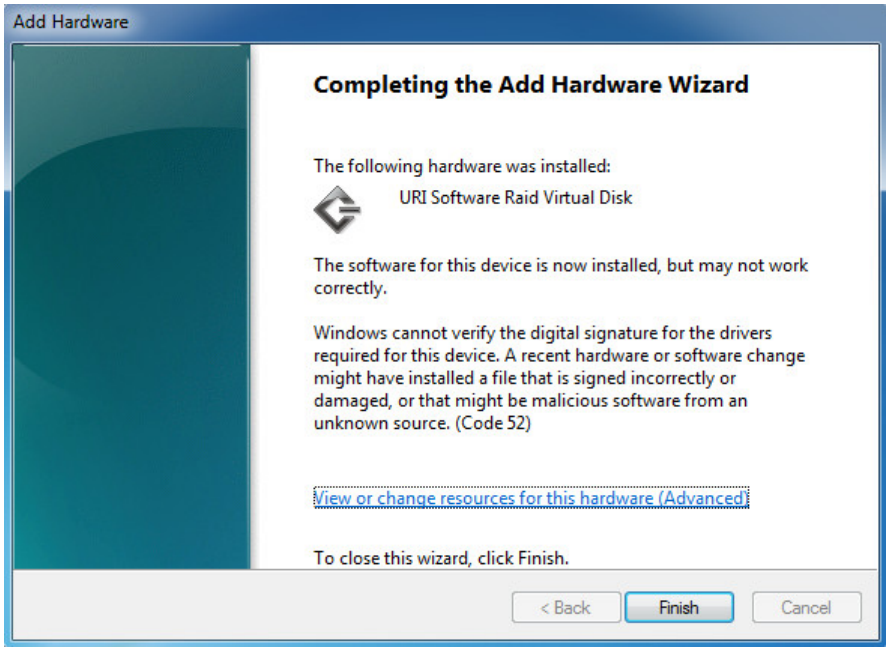



Figure 30. Windows 7 x64 Hardware Wizard Install Complete

You may also receive a program compatibility assistant warning letting you know that the installed driver is not signed and will not function correctly. You can safely ignore this warning, and just hit Close.

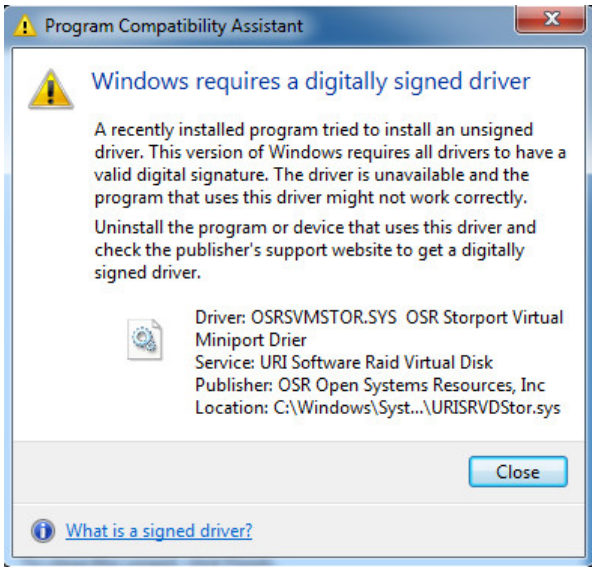



Figure 31. Windows 7 x64 Program Compatibility Assistant Warning

There are additional steps required in order to allow Windows to load the driver since it is unsigned. The first thing that needs to be done is to open an elevated command prompt. In order to do this, open up Start, and type cmd. On the only file that comes up, right click, and select Run as Administrator as seen in the figure below.

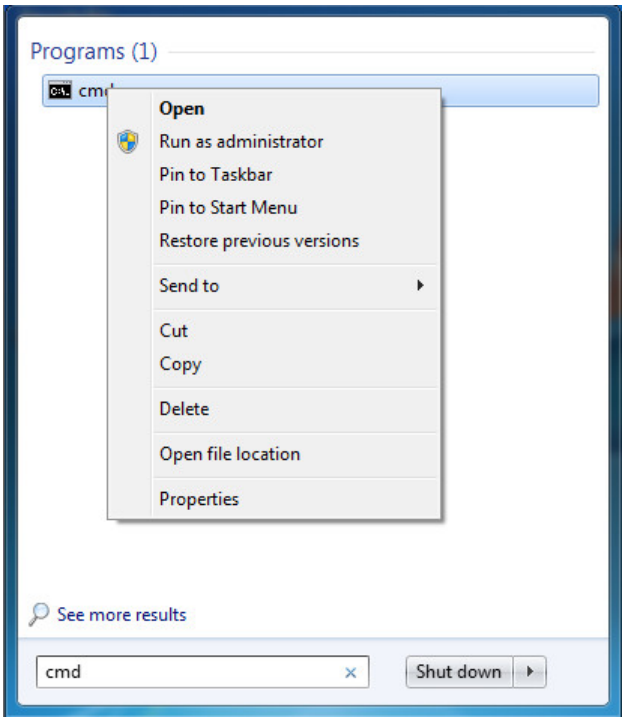



Figure 32. Windows 7 x64 Elevated Command Prompt

Once you have the elevated command prompt, you must type in the following command (without the quotes) in order to put the system into test mode: "bcdedit.exe /set TESTSIGNING ON". Then press the enter key, and you should see the confirmation as in the figure below. Once that is done, simply restart the computer, and you will see the test mode information in the bottom right of the desktop.

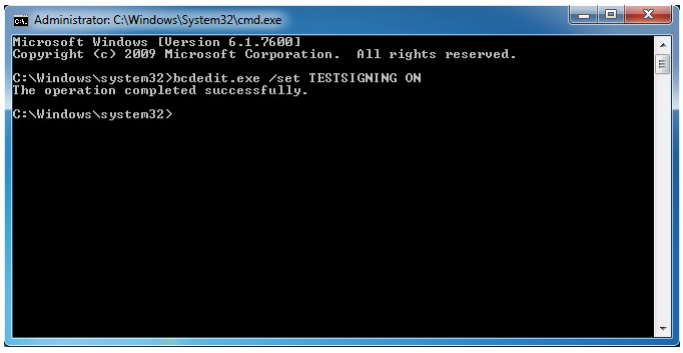



Figure 33. Windows 7 x64 Test Mode Command

#### 3.1.3.3 Driver Installation on Windows Server 2003 x86

There are a few differences to earlier instructions, first to get to add hardware, simply go to the Control Panel, and select Add Hardware. Next it will search for your hardware then display the screen in the figure below. Make sure you select the radial that says Yes, I have already attached this hardware.

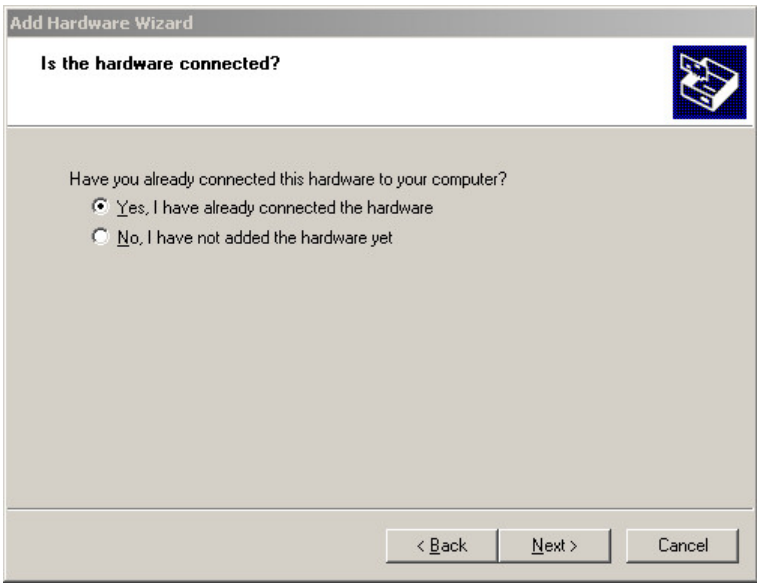



Figure 34. Windows 2003 x86 Hardware Wizard Hardware Attached

On the next screen, scroll all the way to the bottom, and select to Add a new hardware device.

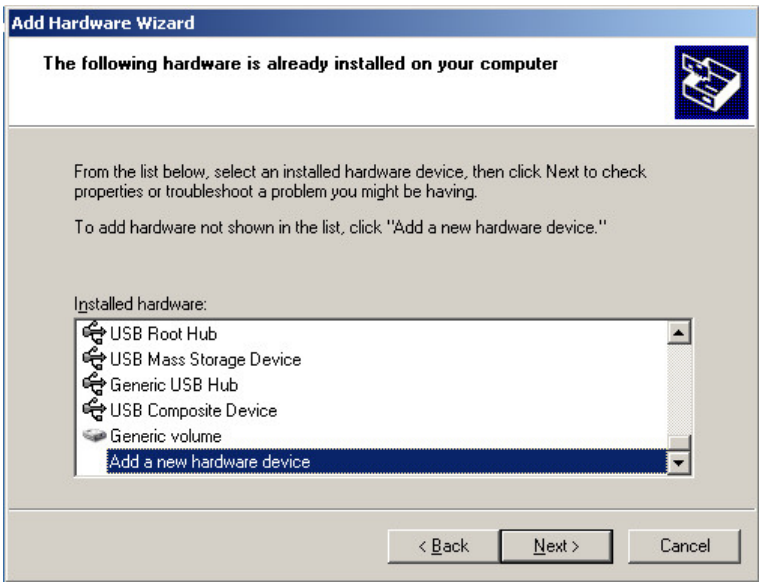



Figure 35. Windows 2003 x86 Hardware Wizard New Hardware Device

After that it follows the same instructions as Windows 7 x86, until at the end you get the conformation screen seen in the figure below.


Figure 36. Windows 2003 x86 Hardware Wizard Install Complete

#### 3.1.3.4 Driver Installation on Windows Server 2003 x64

There is no difference between the 32 and 64 bit installations in Windows Server 2003 so just view the instructions above.

#### 3.1.3.5 Driver Installation on Windows Vista x86

There is no difference from the Windows 7 x86 installation so just view the instructions above.

#### 3.1.3.6 Driver Installation on Windows Vista x64

There is no difference from the Windows 7 x64 installation so just view the instructions above.

#### 3.1.3.7 Driver Installation on Windows Server 2008 x86

There is no difference from the Windows 7 x86 installation so just view the instructions above.

#### 3.1.3.8 Driver Installation on Windows Server 2008 x64

There is no difference from the Windows 7 x64 installation so just view the instructions above.

#### 3.1.3.9 Front-End Installation

First ensure that you have the newest version of .NET 4.0 installed on your system. To do that go to the Microsoft Download Center, and search for the .NET Framework 4.0 (Web Installer)[3]. Once that is installed, you must also ensure that the Visual Studio C++ 2010 redistributable (x86) is installed on the system, which can also be found in the Microsoft Download Center[4]. Follow the instructions on the website in order to download and install the framework. Once that is installed, all you need to do is copy SoftwareRAIDMount.exe and call.dll of the front-end onto the target computer; no installation is needed. To ensure everything is all

set, just run the executable and ensure you get the same screen as in the following figure.

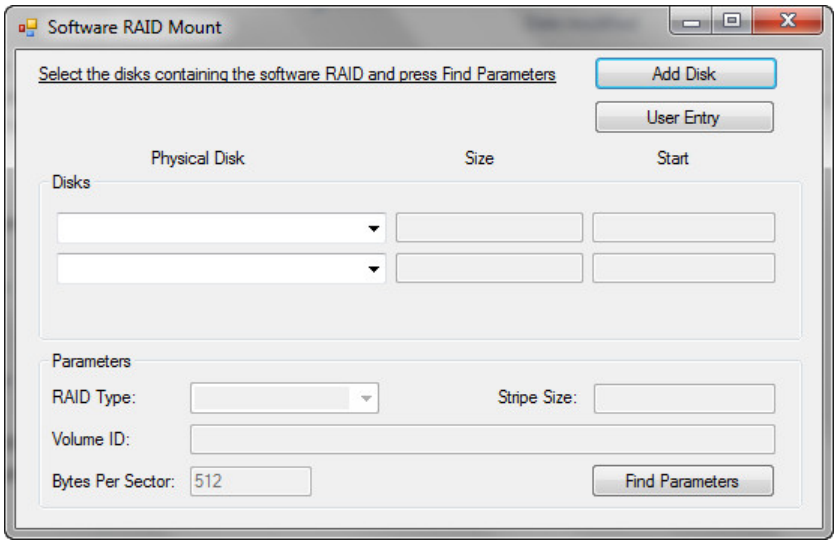



Figure 37. Software RAID Mount

## 3.2 Software RAID Testing

The Software RAID virtual disk was tested in several different phases. The first test is the speed testing which was done on a physical machine first using spanned disks and then striped. Once the speed testing finished the hash testing was done to ensure that the contents of the volume were tested to ensure accuracy. Once that was tested, the next test was to show that the Software RAID Virtual Disk was tested on a variety operating systems, ensuring that it can be used on different systems. Then each configuration of the software RAID were tested along with testing if the driver can handle a small level of corruption. Finally, the last test showed that the Software RAID Virtual Disk can handle opening files and makes no modifications to the system. These tests were done using VMWare Workstation as the underlying hardware was not important.

[3]The link as of the writing is http://www.microsoft.com/download/en/details.aspx?id=17851
[4]The link as of the writing is http://www.microsoft.com/download/en/details.aspx?id=5555

### 3.2.1 Speed Testing

The purposes of the speed testing is to verify that the implementation of the driver is not significantly slower than the Windows implementation. It is also to test and ensure that the slowdown expected from attaching any write blocker is consistent with the base drive slowdown. For this reason several tools were considered and tested, however, only three tools were able to get speed values from the RAID.

The first tool that was tested and was unable to be used was FutureMark's PCMark[8] which failed to run correctly even on Windows implementation of the RAID. The error simply stated Init Error and after contacting FutureMark and supplying all requested information they stopped all communications, and as such the program was never able to run.

The next tool tried was Crystal Disk Mark[9] which performed flawlessly on Windows implementation. When run on the Software RAID Virtual Disk, however, you simply get the error Failed Create File, which is believed to be caused by the software attempting to store a known file on the drive and read that file back for speed testing.

After that the next tool to try was eXibition Software's Drive Speed Checker[10] which again performed great on Windows implementation. Again it failed on the Software RAID Virtual Disk, this time with Permission Denied even when run as an administrator. Again it is believed that the program requires write permission to the disk in order to store a test file.

The last tool that was tried and failed to perform was Open Source Development Lab's Iometer[11] (formerly developed by Intel) which I only tested on the Software RAID Virtual Disk in order to see if it would even be useful. After configuring it to only do read tests, it hung for over 30 minutes on initializing disks

and as such was not used.

After all of the testing, there are three different tools that were used for testing the speed. The first of these only checks the speed of the physical disks and allowed for a baseline with which to compare the speed degradation of write-blocking the disks. The other two tools were run on the logical volume and were able to get good results.

The first tool is EFD Software's HD Tune Pro 5.00[12] which is able to do a variety of tests on the physical disks themselves. The test that was run is the Benchmark tool which had the Read radial selected and the Transfer rate checkbox checked along with the Access time and Burst rate checkboxes. For these tests, a screenshot is provided in the results section.

The second tool that was used was steel byte's HD_Speed[13]. For each configuration the tool was run for 120 minutes with a block size of auto, in read mode, and with results logged to file. After each test a screenshot of the final results was taken and provided in the results section along with the data that was logged.

The next tool that was used was simplisoftware's HD Tach[14] which was setup with the long test option. A screenshot of the completed test is provided in the results section below.

The final tool that was used was a commercially available software write blocker marketed by ForensicSoft, called SafeBlock[15]. This software has been tested to be forensically sound with a minimum of impact on the system.

Each of the tools was used on the following configurations for spanned and then for striped:

1. Windows Mounted, Software RAID Virtual Disk Unmounted, No Write Blocker

2. Windows Mounted, Software RAID Virtual Disk Mounted, No Write Blocker

3. Windows Unmounted, Software RAID Virtual Disk Mounted, No Write Blocker

4. Windows Unmounted, Software RAID Virtual Disk Mounted, SafeBlock Write Blocking

### 3.2.2 Hash Testing

In order to ensure that the logical volume of both the Windows version and Software RAID Virtual Disk are identical, the hash of each should be identical. In order to test this WinHex[16] was used in order to hash the drives. To do this first click on Tools, then Open Disk.

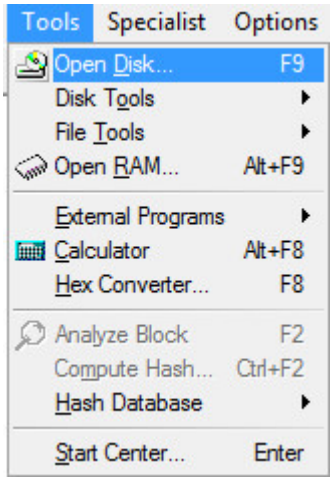



Figure 38. WinHex Open Disk

Next select the volume you wish to hash. For this example I am using a 2 GB thumb drive that is mounted to G.

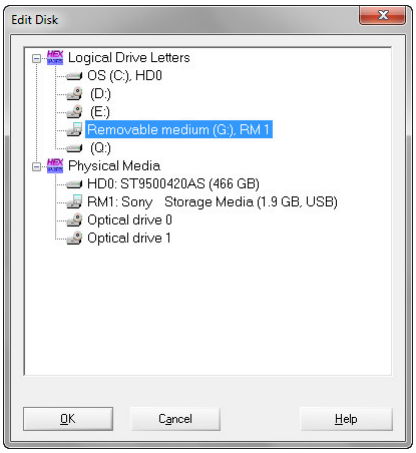



Figure 39. WinHex Disk Choice

Once the disk has opened, you need to start the hashing by again going to the tools dropdown and selecting Compute Hash.

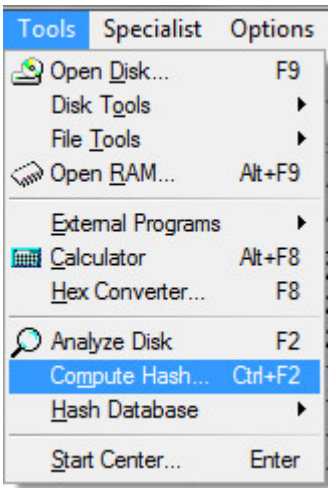



Figure 40. WinHex Compute Hash

There is a choice of hashes; for the test the MD5 hash was used as that is a hash still used extensively by law enforcement.

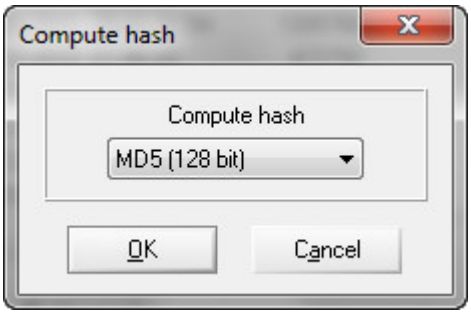



Figure 41. WinHex Hash Choice

Once you have selected the hash and pressed OK, the hashing will begin, and you will see the screen in the figure below. Please allow this to run as it can take over 1 hour depending on the speed and size of the disks.

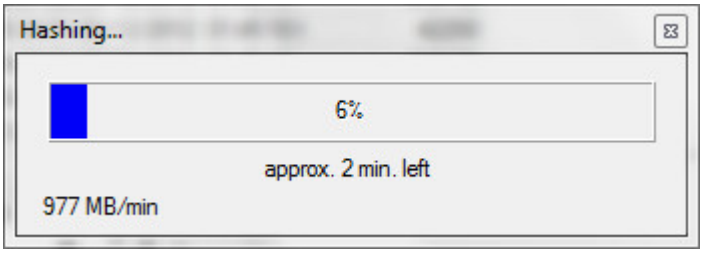



Figure 42. WinHex Hash Computing

Once it has finished, the hash of the drive will be shown as in the following figure. For the purposes of testing, a screenshot of the hash along with the Computer dialog box showing what drive was hashed is provided in the results section. Make sure you do both hashes immediately one after another because Windows will change time stamps if you continue to work or reboot the computer between hashes.

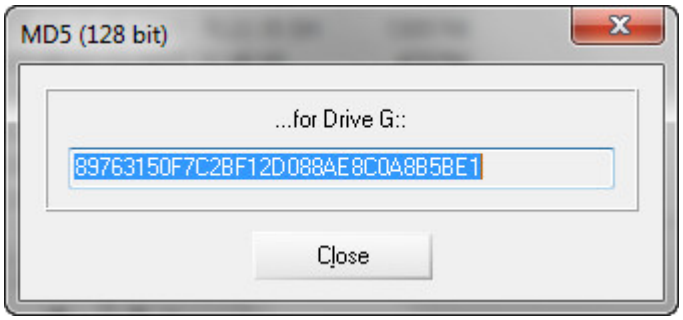



Figure 43. WinHex Hash

### 3.2.3 Operating System Compatibility Testing

In order to test if the driver worked on each operating system it was only necessary to mount a single spanned RAID to show that the driver functioned. This was done in a virtual environment where two 1 GB virtual SCSI disks were added and configured into a spanned RAID using the earlier instructions. In each case this was done on a clean install of the operating system with fresh installs of .NET 4.0 and the Visual Studio 2010 C++ Redistributable. A screen shot is provided of the SoftwareRAIDMount.exe finding all of the parameters successfully then of Computer showing both the Windows mounted version and the Software RAID Virtual Disk mounted version.

### 3.2.4 Configuration Compatibility Testing

There are several different configurations that needed to be tested. The easiest to test was the ability to mount each type of RAID; for that one of each type of RAID was added to a Windows Server 2008 x86 clean installation. For proof screenshots of the Disk Manager setup, each SoftwareRAIDMount.exe parameter set, and finally Computer showing all of the mounted drives are in the results section.

The next configuration that needed to be tested was the ability for the program to mount a disk that has had its LDM database and MBR information erased.

This was accomplished by using WinHex and editing the hex of the spanned disk drives to 00 across the partition table (screenshot will be provided). Now that the partition table is blank SoftwareRAIDMount.exe was used to automatically mount the disks and a screenshot will be provided. After that the LDM database, which is the last 1 MB of the disk, was also be overwritten with 00 to remove all traces of the software RAID (screenshot will be provided). Once that has been done the SoftwareRAIDMount.exe will attempt to automatically mount the disks and after that fails the information was hand entered and a screenshot was provided of the mounted disk.

After proving that a corrupted disk can be mounted as long as the underlying file system is intact, a GPT disk spanned RAID was created and WinHex was used to find the start of the file system and other pertinent information. Once that was found, it was plugged into the SoftwareRAIDMount.exe and a screenshot of both the Disk Configuration and Computer are provided below.

The last configuration to test was the ability to even mount an NTFS partition from inside of a Linux software RAID. The software RAID was setup using Ubuntu 12.04 LTS x64 server install CD[17]. Before starting ensure that you have 2 blank drives attacked to the system for the RAID. Boot into the CD and then select to Install Ubuntu to the hard disk, as seen in the next figure.

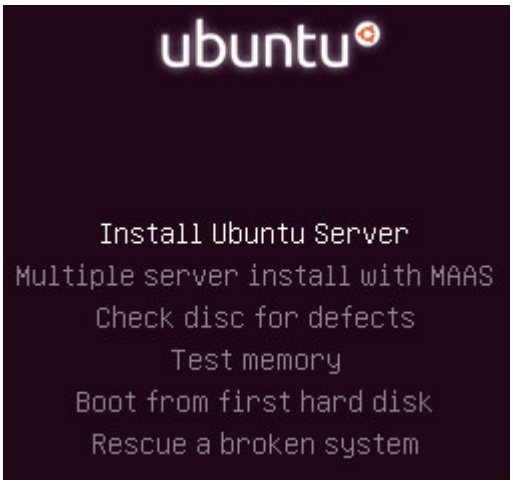



Figure 44. Ubuntu Starting Screen

At each step through the partition screen simply accept the default options. You can choose anything for a hostname, user name and password. Once the partition screen comes up choose to manually partition the disks.

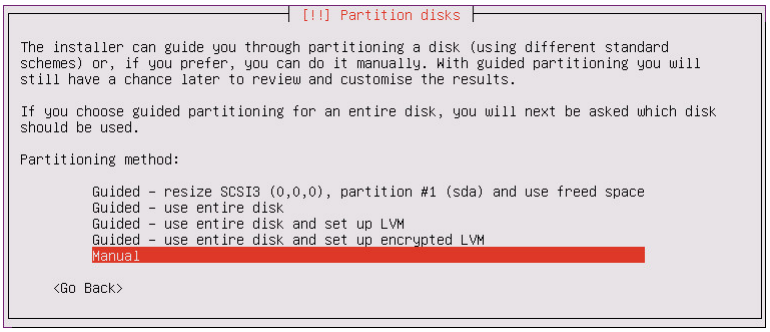



Figure 45. Ubuntu Manually Partition Disks

Next scroll down to the first of the unpartitioned drives added for the RAID and select it.

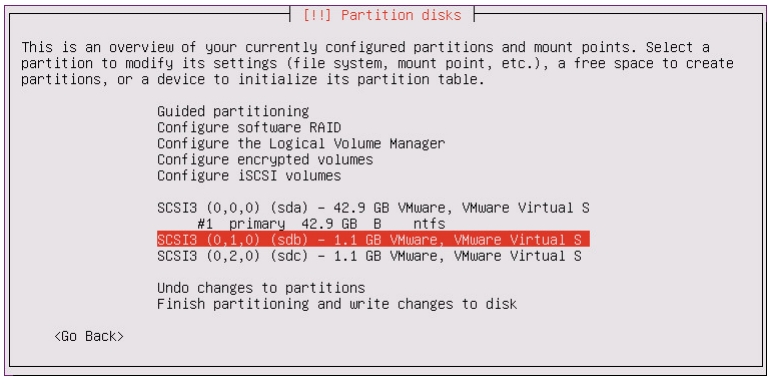



Figure 46. Ubuntu First RAID Drive

It will ask if you want to create a new empty partition table on this device, select yes.

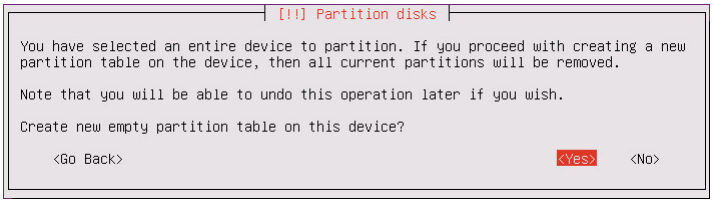



Figure 47. Ubuntu Create Partition Table

Next there will be a new line under the drive labeled as FREE SPACE, select the line to partition the drive.

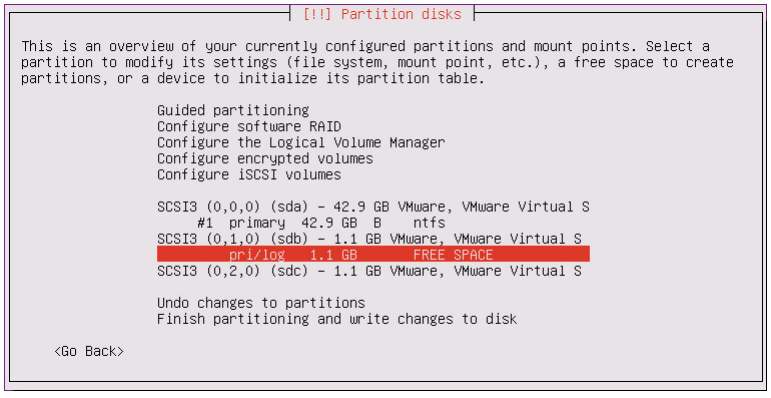



Figure 48. Ubuntu Partition Free Space

In the next message box just select to Create a new partition.


Figure 49. Ubuntu Create Partition

Choose the max space which is the default option, so just hit enter.

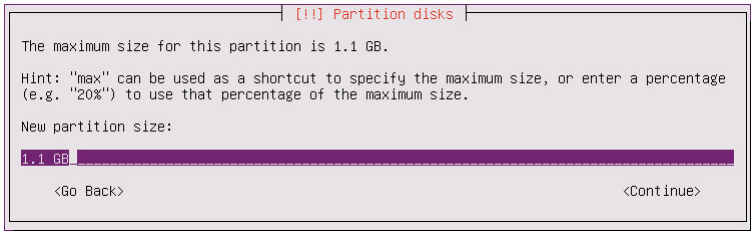



Figure 50. Ubuntu Partition Size

This is the basis of the software RAID so at this point just chose to make this a primary partition.


Figure 51. Ubuntu Partition Type

By default the partition begins as an Ext4 partition, this needs to be changed in order to use this as a software RAID so select the Use as line to change the formatting.

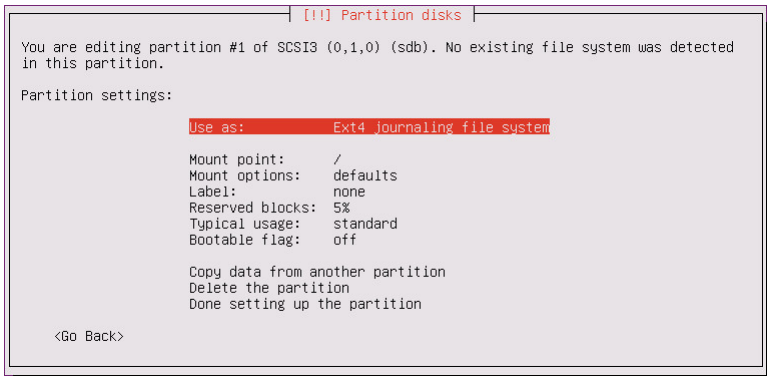



Figure 52. Ubuntu Partition Use Type

Now select physical volume for RAID in order to make this an empty partition we can use.

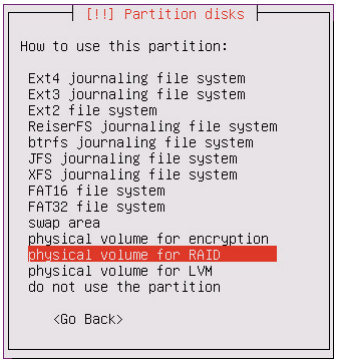



Figure 53. Ubuntu Physical Volume for RAID Partition

Finally select Done setting up the partition in order to finish setting up the first drive.

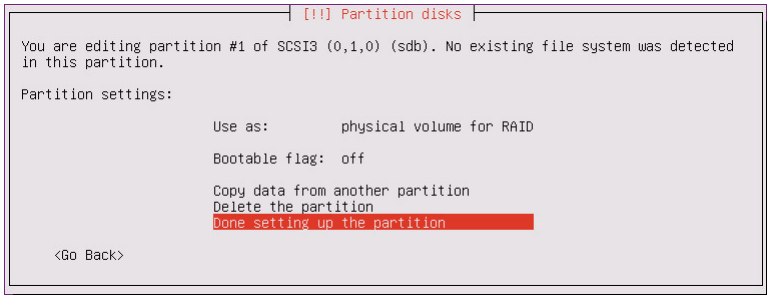



Figure 54. Ubuntu Finish Drive Setup

Now repeat those same steps on the second drive so that they are both formatted for the RAID. Once they are both done go up to Configure software RAID to begin setting up the RAID.

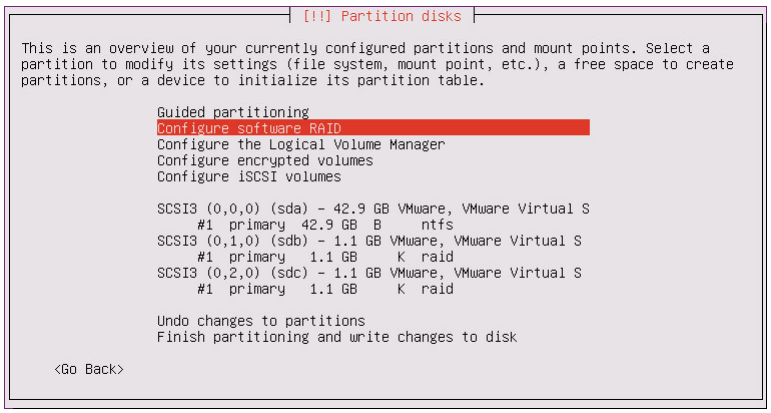



Figure 55. Ubuntu Configure Software RAID

It will ask to write all the changes that have been made so far to the disks, select yes to continue on to configure the RAID.

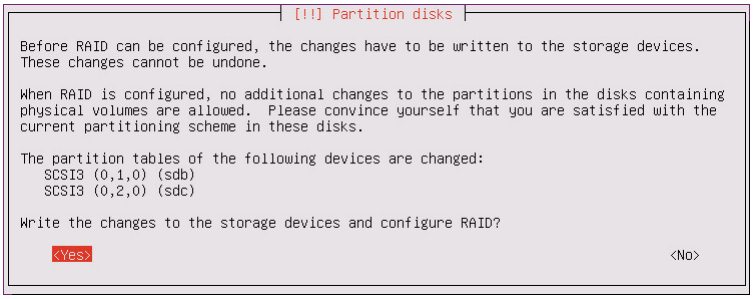



Figure 56. Ubuntu Write Changes

To start creating the software RAID select Create MD device to create a new multiple disk or RAID device.


Figure 57. Ubuntu Create MD "RAID" Device

For this test we used RAID0 in order to fully test the capabilities, so simply select RAID0 and continue on.

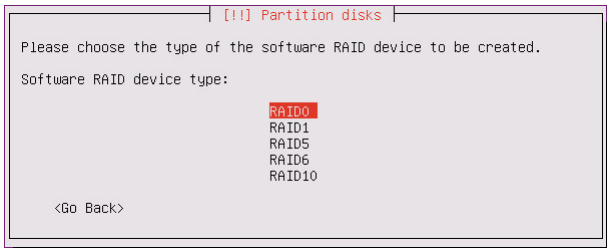



Figure 58. Ubuntu RAID Type

Next the disks in the RAID need to be selected, so scroll to each of the RAID disks and hit the spacebar to select the disks. Once both disks are selected hit enter to continue.

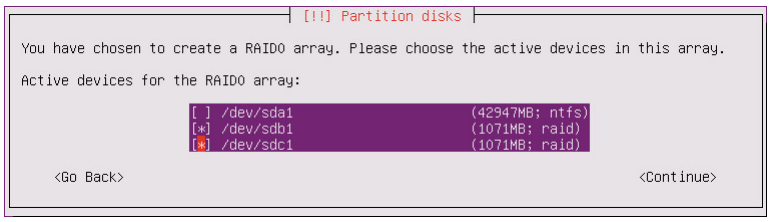



Figure 59. Ubuntu RAID Disk Select

Now select Finish in order to go on and format the RAID.


Figure 60. Ubuntu Finish RAID Setup

Now at the top of the disk list there should be a new RAID0 device with 1 unpartitioned space and some unusable space. Select the unpartitioned space to create and format the postition.

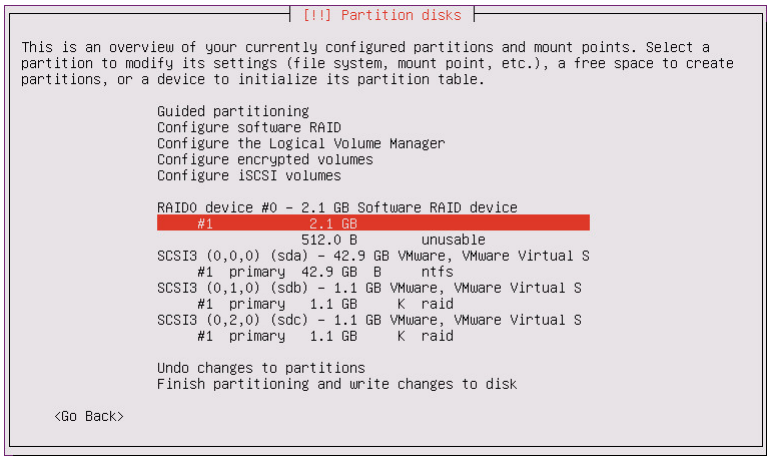



Figure 61. Ubuntu Partition RAID

The partition is currently set as do not use, select the Use as line in order to change that.

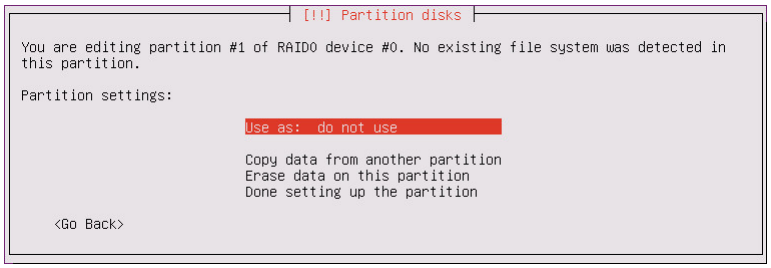



Figure 62. Ubuntu RAID Use Type

Since there is no choice for NTFS, choose FAT32 so that the RAID is formatted with a partition that Windows will be capable of mounting.

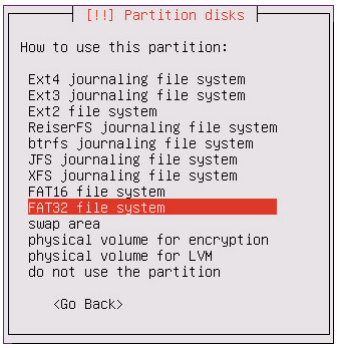



Figure 63. Ubuntu FAT32 RAID Partition

At this point the partition is all set so select Done setting up the partition to finish up.

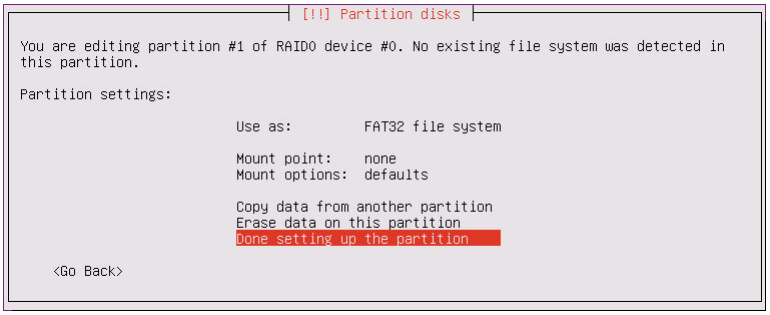



Figure 64. Ubuntu Finish RAID Partition

Because we are not actually installing Ubuntu onto the system we cannot select finish partitioning or it will just keep sending us back here. Because of this select Configure Software RAID as above and then select yes to write the changes to the disk. At this point stop the system either by powering off the VM or by holding the power button on a physical machine and then boot into Windows.

Once booted into Windows a tool such as WinHex will need to be used in-order to find the RAID information such as the starting sector, size, and stripe size.

### 3.2.5 Content Testing

In order to test that content works correctly on the Software RAID Virtual Disk, several files were stored on a striped disk, on a clean install of Windows 7 x86, all of which were large enough to span several stripes. These files include an image, a text file, a pdf, a md4 movie and a MPEG-4 song. After those files were placed on the drive, the Windows implementation was unmounted, and the drive was mounted with Software RAID Virtual Disk. A starting hash of the drive was taken immediately in order to ensure that nothing that would be tested would change any data.

First in order to test that attempting to write to the drive will not change anything, the files that are stored on the drive are attempted to be copied onto the drive again. This should fail and an error will show up on the Windows tool bar. After this the hash will be taken to ensure that the attempt did not change anything.

Next each of the files will be opened and run, for each of them a screenshot will be provided showing that the file opened correctly and that the Windows implementation was offline. Once all of the files have been opened, the hash will be checked again to ensure that none of the time stamps were modified.

Finally the text file will be opened, and the text inside will be changed. The file will attempt to be saved which again should fail, and a final hash was taken to ensure that nothing had been modified.

### List of References

[1] VMware, Inc. "Vmware workstation 8." [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.vmware.com/products/workstation/

[2] ASUS. "N61jq." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.asus.com/Notebooks/Multimedia_Entertainment/N61Jq/

[3] Dell. "Optiplex 760 desktop." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.dell.com/us/dfb/p/optiplex-760/pd

[4] intel. "Intel®core™2 duo processor e7300."

[5] Microsoft. "Windows enterprise." [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.microsoft.com/en-us/windows/enterprise/products-and-technologies/windows-7/default.aspx

[6] adaptec. "Aha-2940uw." [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.adaptec.com/en-us/support/scsi/2940/aha-2940uw/

[7] Seagate. "Quantum®atlas 10k ii." [Online; accessed 9-March-2012]. 2000. [Online]. Available: http://www.seagate.com/staticfiles/maxtor/en_us/documentation/data_sheets/atlas_10k_ii_datasheet.pdf

[8] Futuremark. "Pcmark pc performance testing." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.pcmark.com/

[9] Crystal Dew World. "Crystaldiskmark." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://crystalmark.info/software/CrystalDiskMark/index-e.html

[10] eXibition Software. "Drive speed checker." [Online; accessed 13-December-2011]. 2004. [Online]. Available: http://www.exibitionsoftware.com/products/drivespeedchecker/details.asp

[11] Open Source Development Lab. "Iometer." [Online; accessed 13-December-2011]. July 2006. [Online]. Available: www.iometer.org

[12] EFD Software. "Hd tune." [Online; accessed 13-December-2011]. Aug. 2010. [Online]. Available: http://www.hdtune.com/

[13] steel bytes. "Hd_speed." Jan. [Online]. Available: http://www.steelbytes.com/?mid=20

[14] simplisoftware. "Hd tach." [Online; accessed 13-December-2011]. [Online]. Available: http://www.simplisoftware.com/Public/index.php?request=HdTach

[15] ForensicSoft. "Safe block." [Online; accessed 1-October-2011]. 2010. [Online]. Available: https://www.forensicsoft.com/safeblock.php

[16] "X-ways forensics/winhex," pdf, X-Ways, 2011.

[17] Ubuntu. "Download ubuntu." [Online; accessed 2-May-2012]. 2012. [Online]. Available: http://www.ubuntu.com/download/server

# CHAPTER 4

# Findings

## 4.1 Speed Testing

### 4.1.1 HD Pro Results

The first results generated were from HDTune Pro in order to establish a baseline of what the maximum disk speed for each setting should be for each disk. Here are the screenshots of the initial tests done: first is the results when only Windows is mounted, then the results of when both Windows and the Software RAID Virtual Disk is mounted. Next is when just the Software RAID Virtual Disk is mounted and finally when the disks are write blocked and the Software RAID Virtual Disk is mounted.

#### 4.1.1.1 Windows Mounted

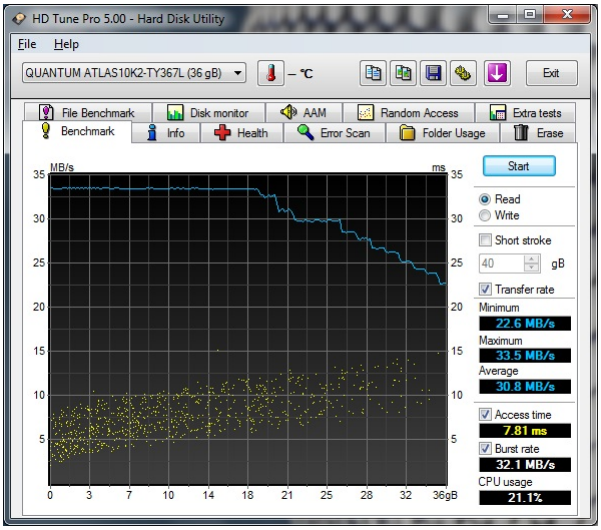



Figure 65. HD Tune Windows Disk 1 Baseline

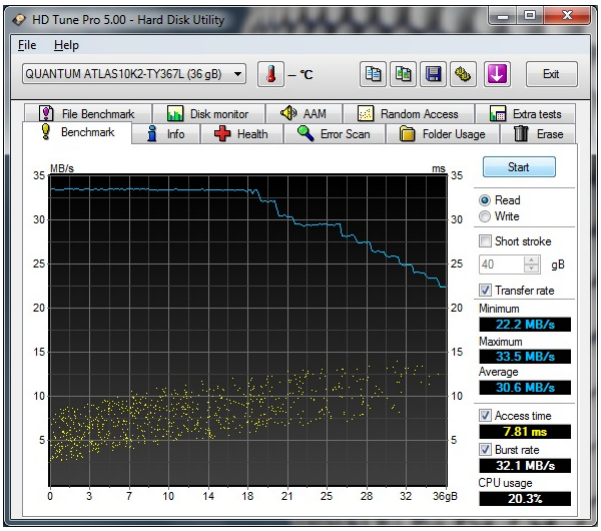



Figure 66. HD Tune Windows Disk 2 Baseline

#### 4.1.1.2 Windows and Software RAID Virtual Disk Mounted

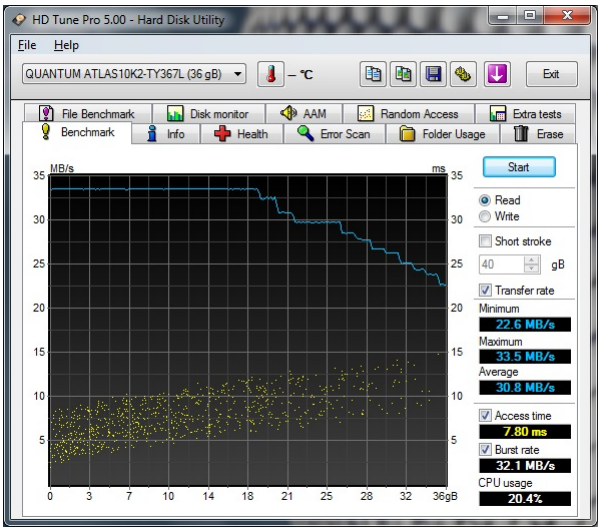



Figure 67. HD Tune Online Disk 1 Baseline

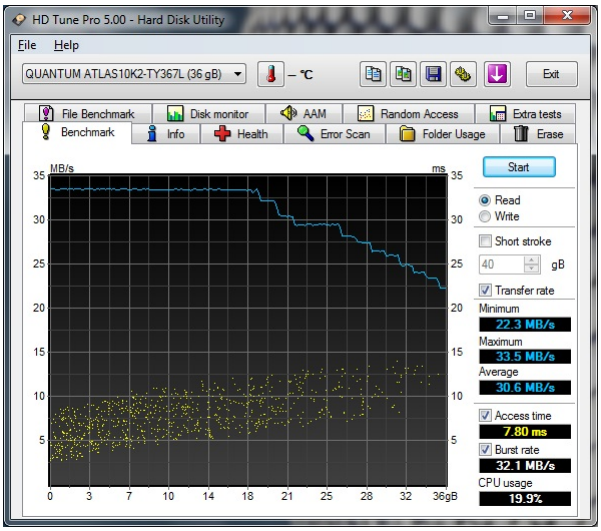



Figure 68. HD Tune Online Disk 2 Baseline

#### 4.1.1.3 Software RAID Virtual Disk Mounted

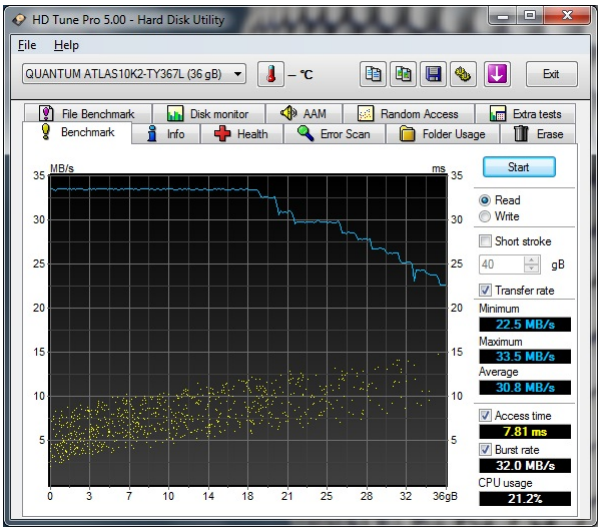



Figure 69. HD Tune Offline Disk 1 Baseline

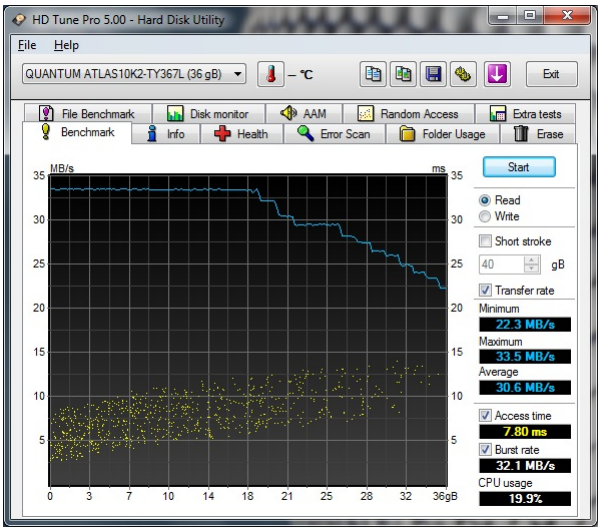



Figure 70. HD Tune Offline Disk 2 Baseline

#### 4.1.1.4 Write Blocked

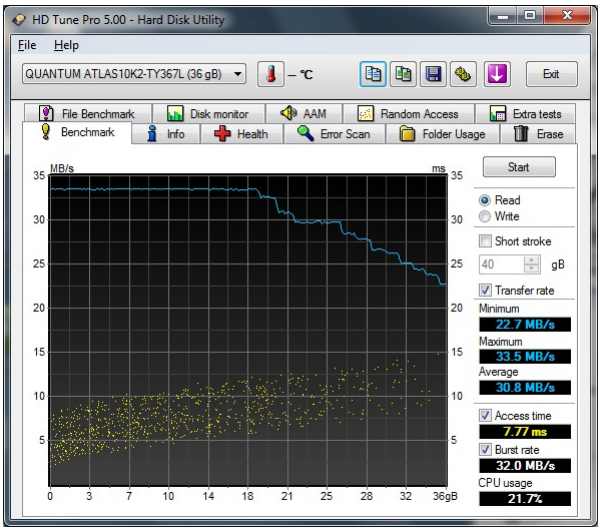



Figure 71. HD Tune SafeBlock Disk 1 Baseline

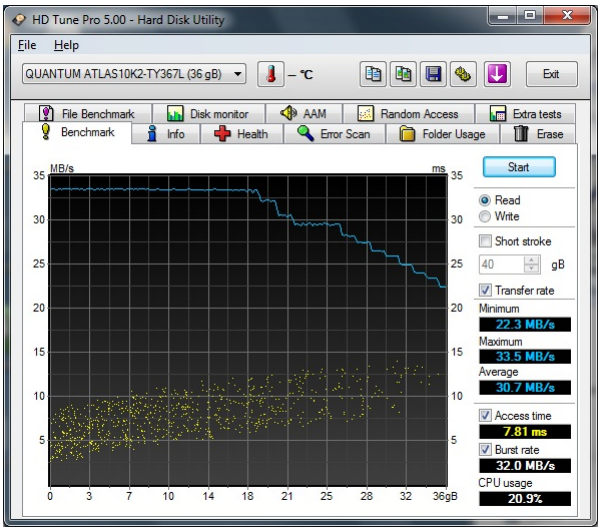



Figure 72. HD Tune SafeBlock Disk 2 Baseline

### 4.1.2 HD Speed Results

Here are the screenshots of the final result for HD Speed on each of the tests after running for 120 minutes.

#### 4.1.2.1 Windows Mounted

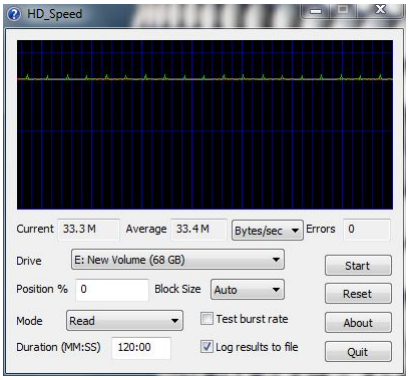



Figure 73. HD Speed Windows Spanned Results

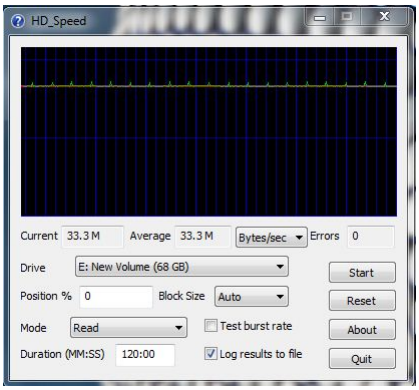



Figure 74. HD Speed Windows Striped Results

#### 4.1.2.2 Windows and Software RAID Virtual Disk Mounted

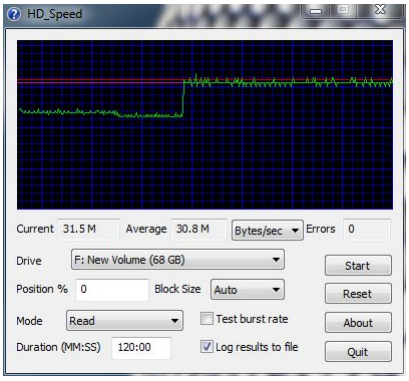



Figure 75. HD Speed Online Spanned Results

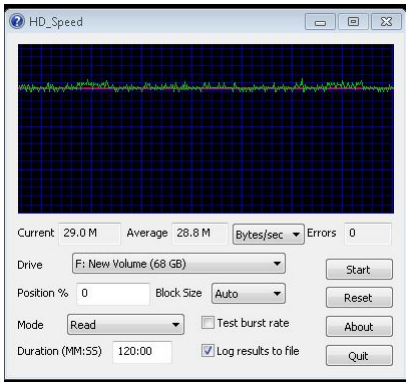



Figure 76. HD Speed Online Striped Results

#### 4.1.2.3 Software RAID Virtual Disk Mounted

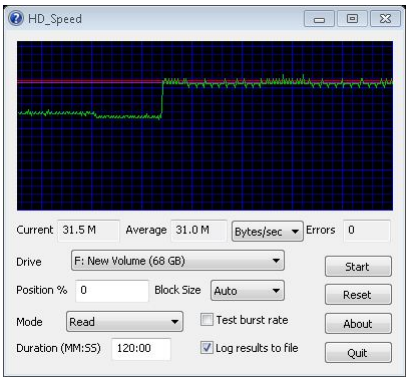



Figure 77. HD Speed Offline Spanned Results

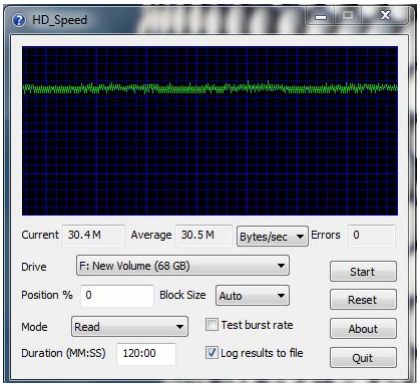



Figure 78. HD Speed Offline Striped Results

#### 4.1.2.4 Write Blocked

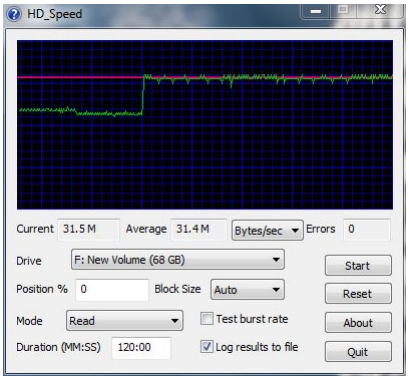



Figure 79. HD Speed SafeBlock Spanned Results

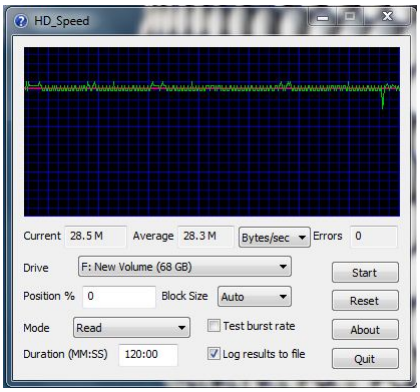



Figure 80. HD Speed SafeBlock Striped Results

### 4.1.3 HD Tach Results

Here are the screenshots of the results for HD Tach on each of the tests. There are no results for the Windows implementation because as it is programmed HD Tach can only speed test drives, however, because of the way the Software RAID Virtual Disk is presented to the system it was able to be utilized.

#### 4.1.3.1 Windows and Software RAID Virtual Disk Mounted

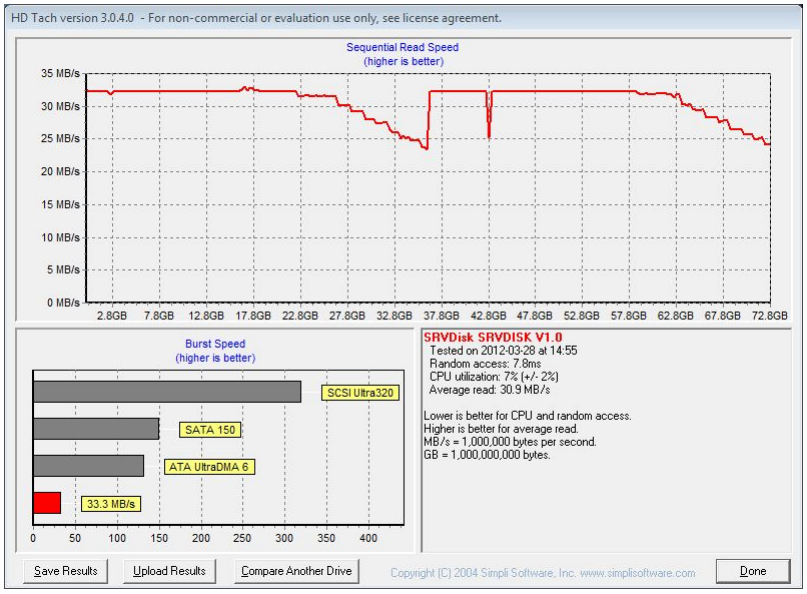



Figure 81. HD Tach Online Spanned Results

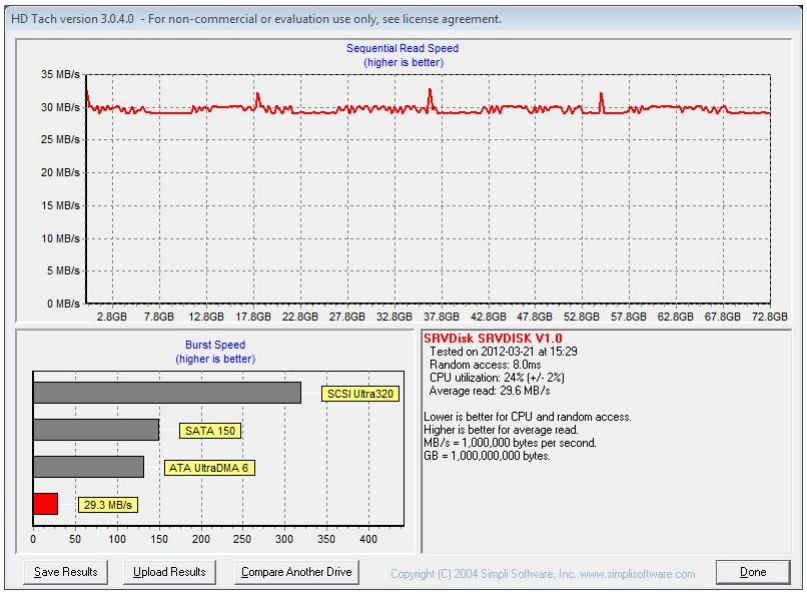



Figure 82. HD Tach Online Striped Results

#### 4.1.3.2 Software RAID Virtual Disk Mounted

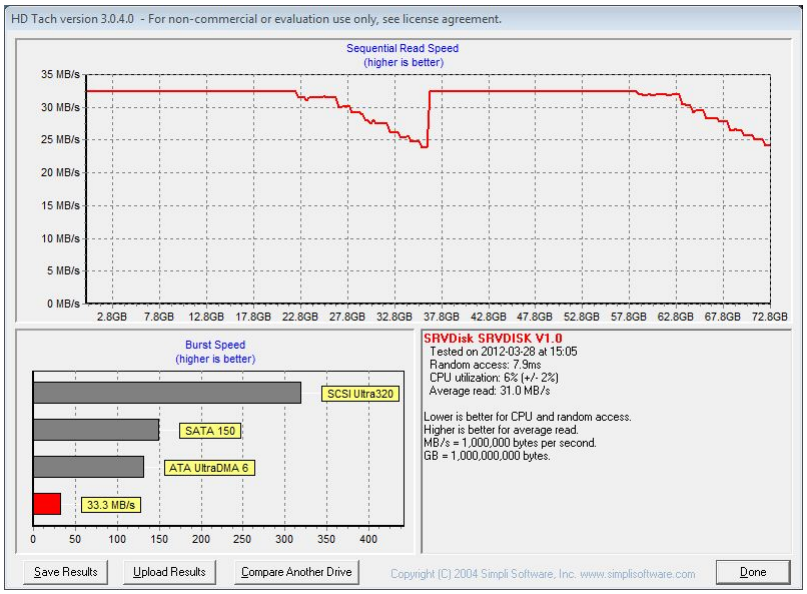



Figure 83. HD Tach Offline Spanned Results

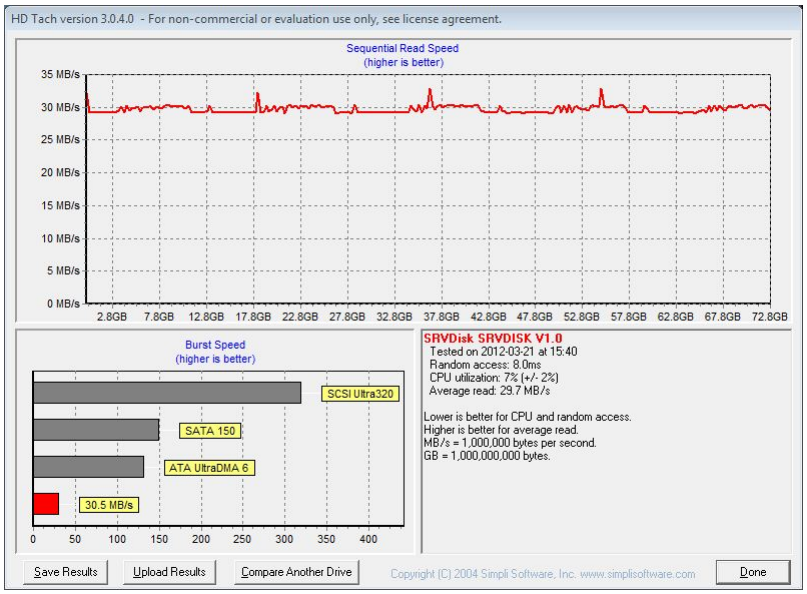



Figure 84. HD Tach Offline Striped Results

#### 4.1.3.3 Write Blocked

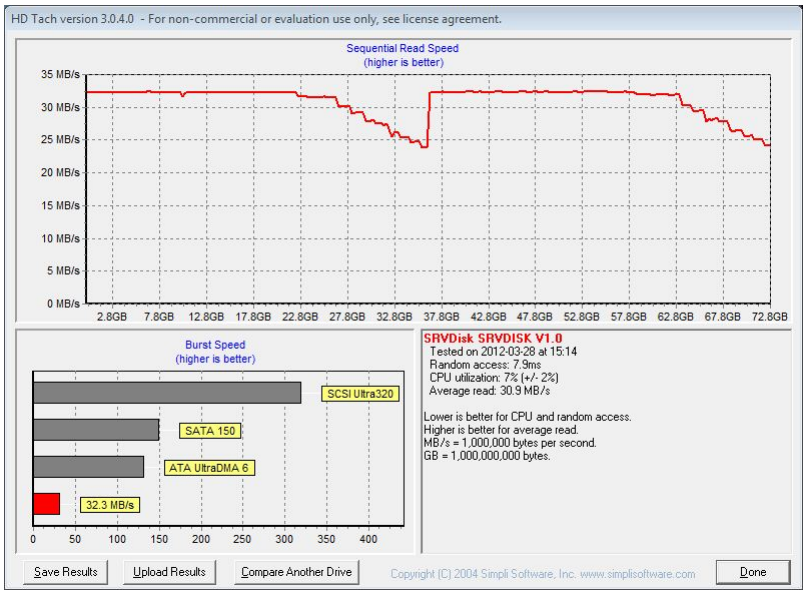



Figure 85. HD Tach SafeBlock Spanned Results

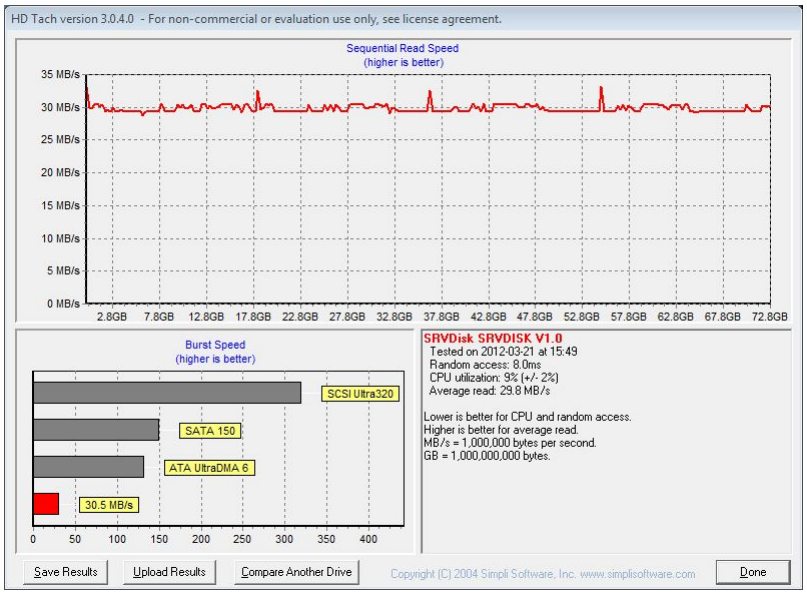



Figure 86. HD Tach SafeBlock Striped Results

### 4.1.4 Charted Results

Charting the above data to put the Maximum, Minimum and Average speed of each of the disks as well as from the RAIDs gives the following chart. Please note that these averages result from calculating the average from the underlying data supplied by HD Speed and not from the average displayed on the image.

| | Windows | Online | Offline | Write Blocked |
|---|---|---|---|---|
| **Disk 1 Minimum** | 22.6 | 22.6 | 22.5 | 22.7 |
| **Disk 2 Minimum** | 22.2 | 22.3 | 22.3 | 22.3 |
| **Spanned Minimum** | 10 | 12 | 22 | 12 |
| **Striped Minimum** | 10 | 24.3 | 24.25 | 24 |
| **Disk 1 Maximum** | 33.5 | 33.5 | 33.5 | 33.5 |
| **Disk 2 Maximum** | 33.5 | 33.5 | 33.5 | 33.5 |
| **Spanned Maximum** | 35 | 33 | 33 | 33 |
| **HDTach Spanned Burst** | | 33.3 | 33.3 | 32.3 |
| **Striped Maximum** | 35 | 32.25 | 32.25 | 31 |
| **HDTach Striped Burst** | | 29.3 | 30.5 | 30.5 |
| **Disk 1 Average** | 30.8 | 30.8 | 30.8 | 30.8 |
| **Disk 2 Average** | 30.6 | 30.6 | 30.7 | 30.7 |
| **Spanned Average** | 30.6 | 29.4 | 29.4 | 29.5 |
| **HDTach Spanned Average** | | 30.9 | 31 | 30.9 |
| **Striped Average** | 33.2 | 30.1 | 30.2 | 28.4 |
| **HDTach Striped Average** | | 29.6 | 29.7 | 29.8 |

Table 2. Chart of Data of Disk Speed with RAID Speeds

The next graph shows all of that charted data graphed. As you can see it is a little busy so it will be broken down further in the following graphs.

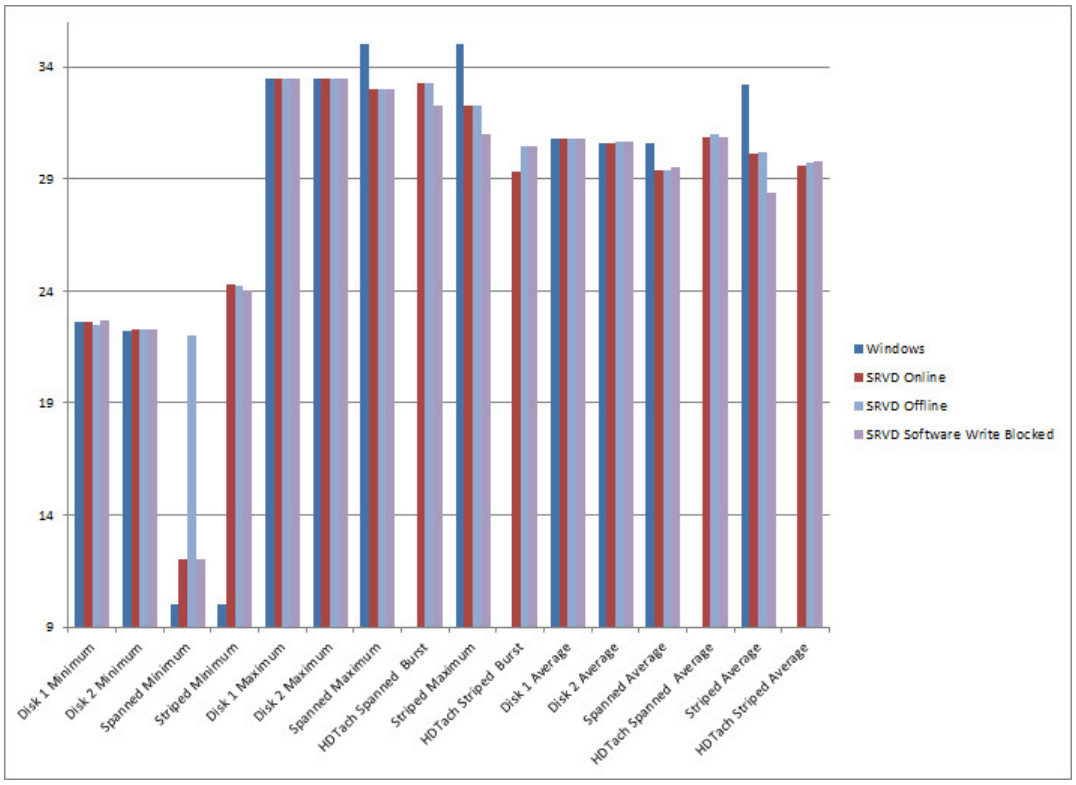



Figure 87. Graph of Data of Disk Speed with RAID Speeds

First the maximum speeds achieved were plotted. As seen in the following graph the Windows implementation outperformed every other implementation, including the disk baselines which are overlapped. Furthermore, it should be noted the HD Tach is not really showing the maximum speed as the others are but instead the burst speed which is why it is lower then the other results.

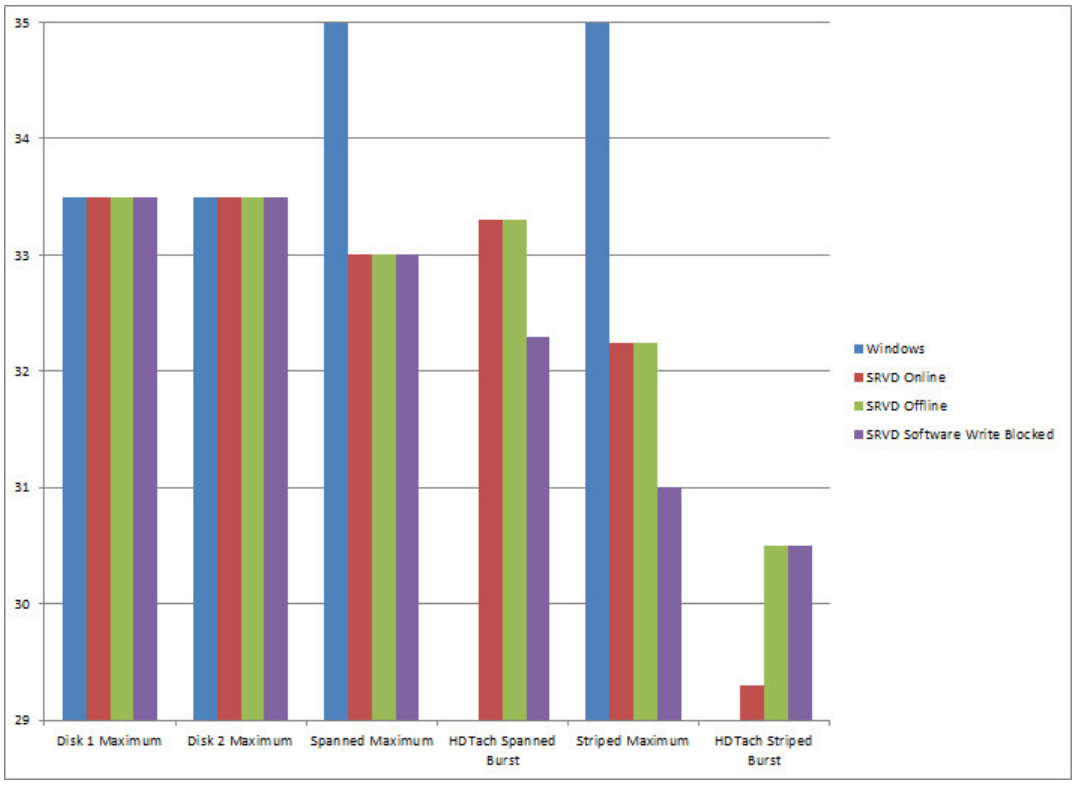



Figure 88. Graph of Data of Maximum Disk Speed with RAID Speeds

Strangely, while Windows had the highest maximum, it also has the lowest minimum which means that it had the largest standard deviation of the tested methods. Spanned was effected the most on the low end which was expected as striped is the option chosen for performance.

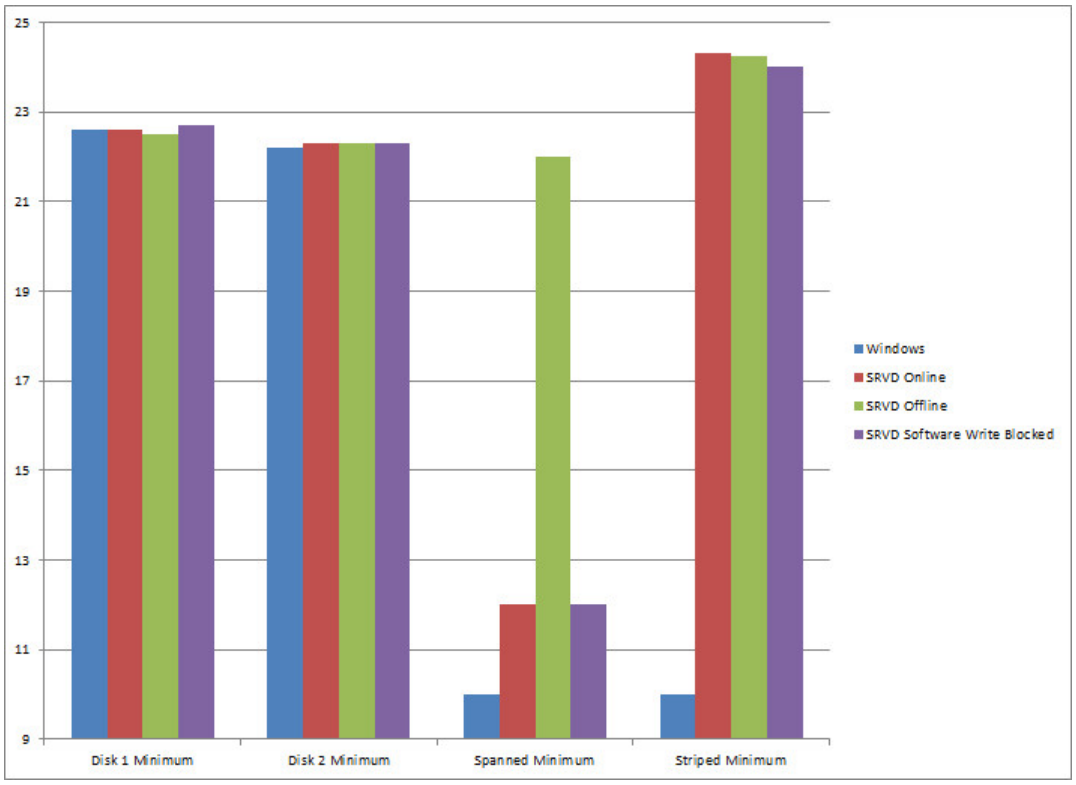



Figure 89. Graph of Data of Minimum Disk Speed with RAID Speeds

Finally the average is the most telling graph of them all, while Windows may have had the lowest minimum, on average it does outperform the Software RAID Virtual Disk implementation.

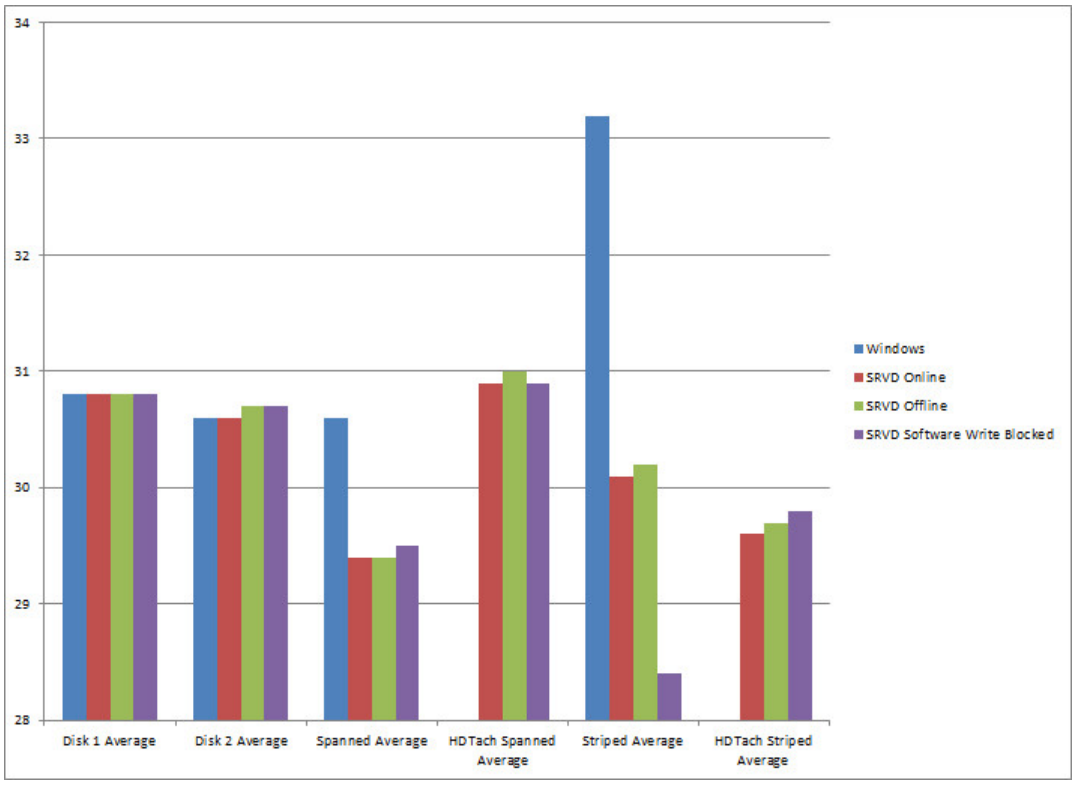



Figure 90. Graph of Data of Average Disk Speed with RAID Speeds

## 4.2 Hash Testing

### 4.2.1 Spanned RAID

Both implementations of the RAID arrived at the hash BBF1E92B007D8B536FCED844649B1C18.

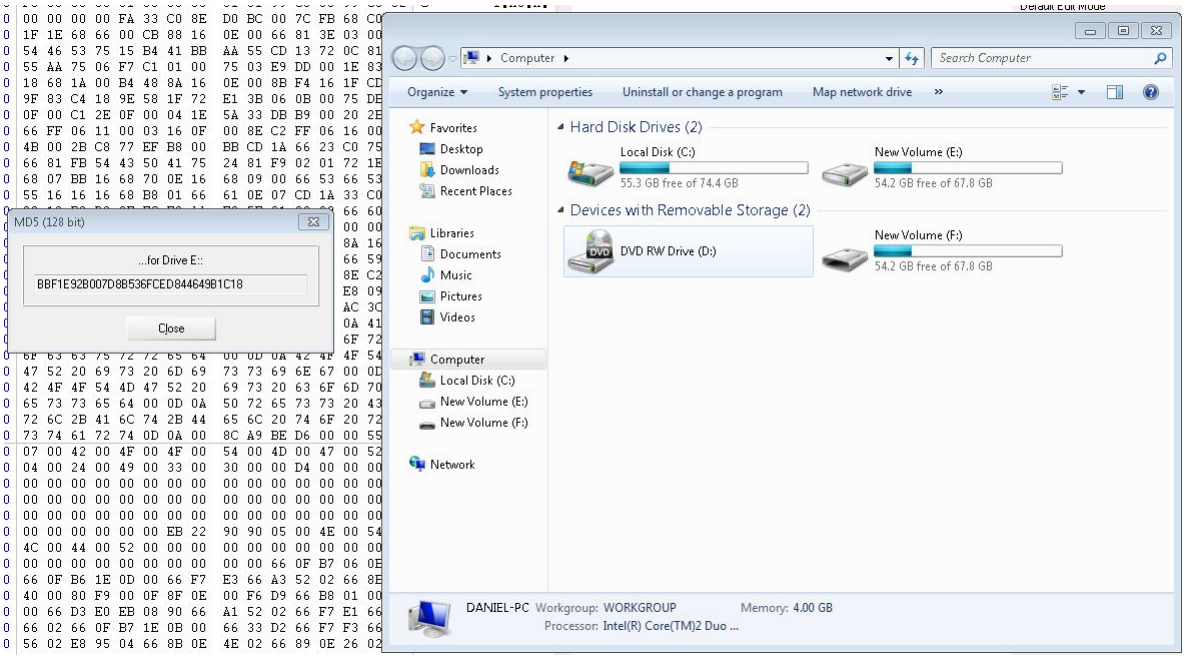



Figure 91. Windows Spanned Raid Hash

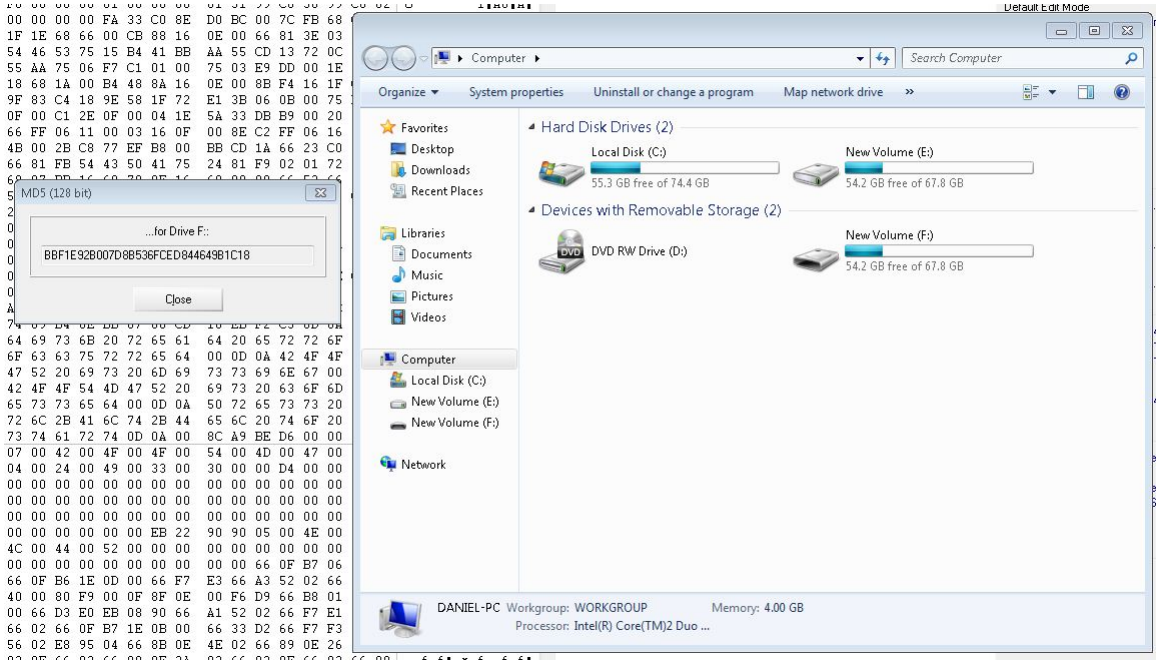



Figure 92. Software RAID Virtual Disk Spanned Raid Hash

### 4.2.2 Striped RAID

Both implementations of the RAID arrived at the hash F0CCB75A695EA870BDA3AA1B39727425.

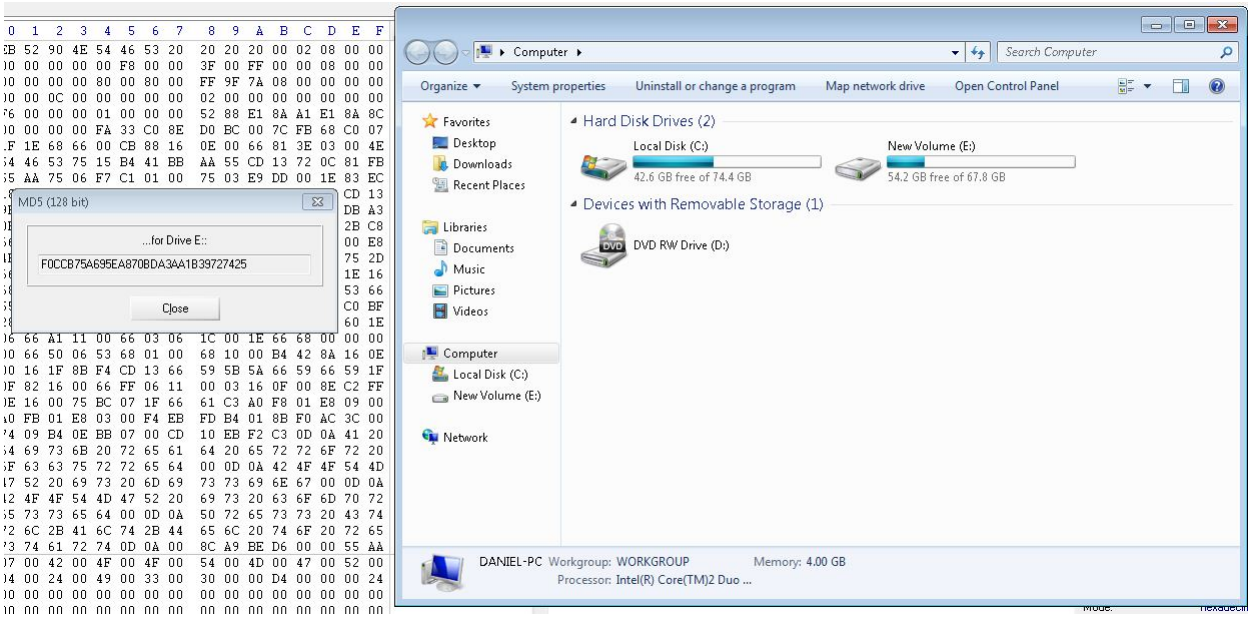



Figure 93. Windows Striped Raid Hash

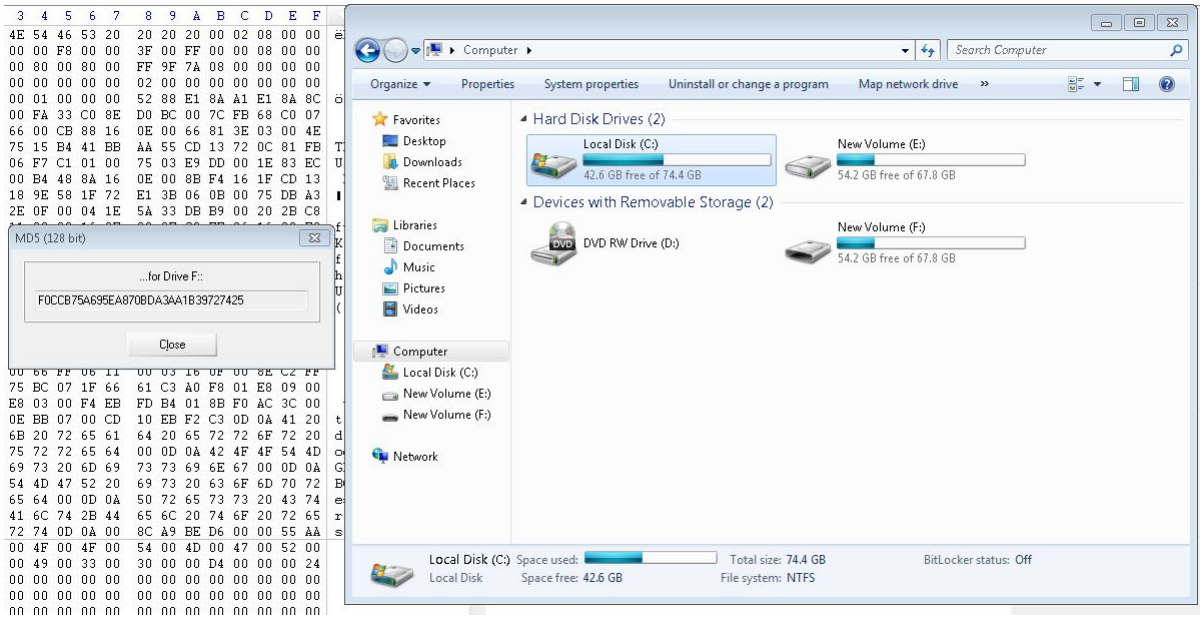



Figure 94. Software RAID Virtual Disk Striped Raid Hash

## 4.3 Operating System Compatibility Testing

### 4.3.1 Windows Server 2003 x86

While trying to test this configuration, it was quickly discovered that the automated parameter finding does not work on this operating system. For this reason the values you see in the figure below were hand entered in order to test if the driver itself works.

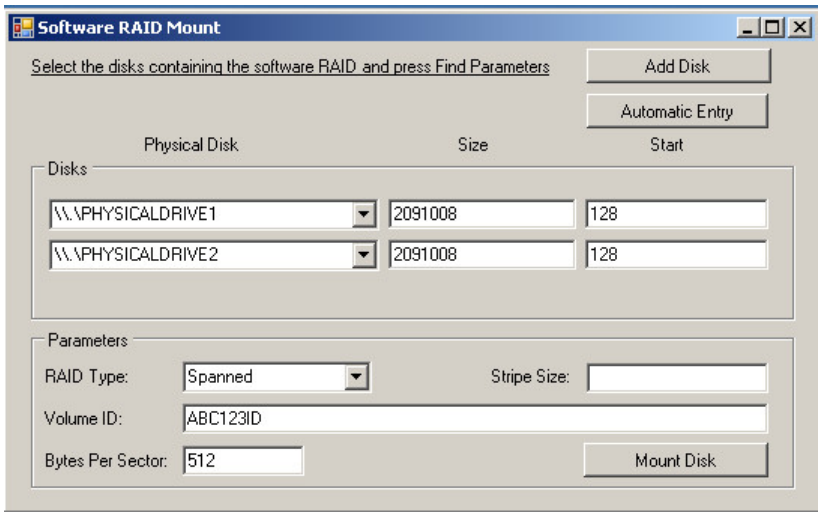



Figure 95. Windows Server 2003 x86 SoftwareRAIDMount.exe

After clicking on Mount Disks, nothing happened: no error and no mounted disk. This was tried three times but never managed to function.

### 4.3.2 Windows Server 2003 x64

While trying to test this configuration, it was quickly discovered that the automated parameter finding does not work on this operating system. For this reason the values you see in the figure below were hand entered in order to test if the driver itself works.

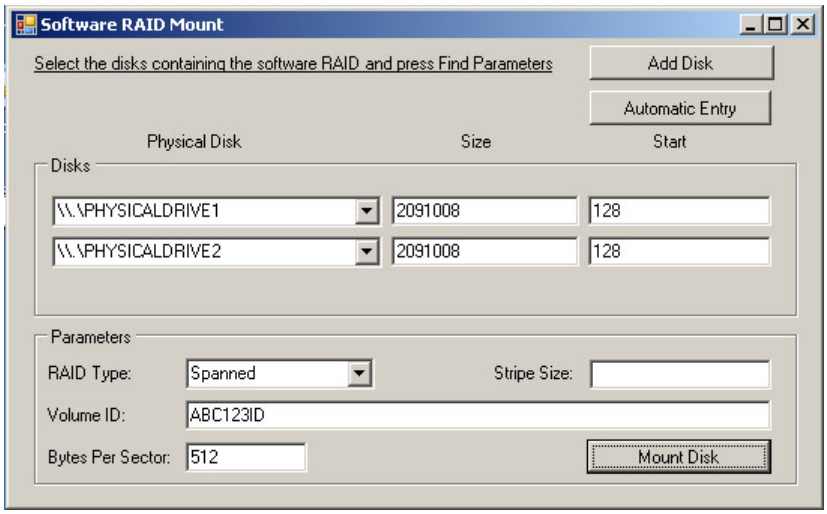



Figure 96. Windows Server 2003 x64 SoftwareRAIDMount.exe

After clicking on Mount Disks, nothing happened: no error and no mounted disk. This was tried three times but never managed to function.

### 4.3.3 Windows Vista x86

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran. After pressing Find Parameters immediately the volume information was found as seen in the figure below.

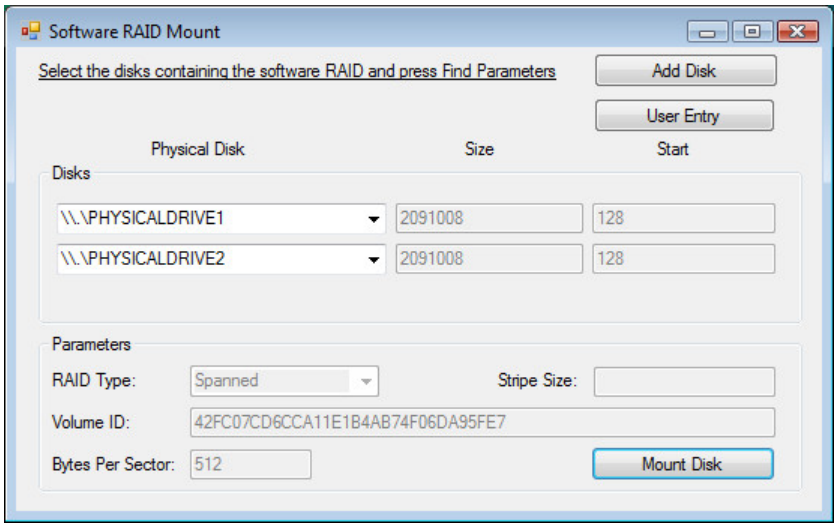



Figure 97. Windows Vista x86 SoftwareRAIDMount.exe

After clicking Mount Disk, there was no error message or confirmation as expected. To verify that the disk was properly mounted, Computer was opened, and as you can see in the next figure, there is now a F: disk under removable disks with the same Volume Name as the Windows RAID.

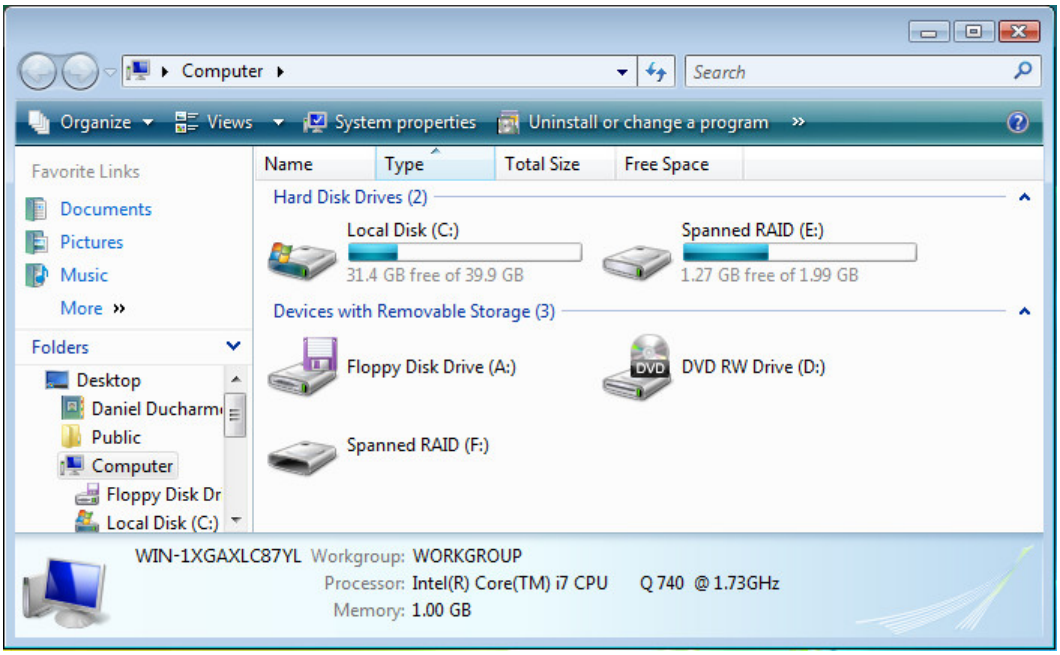



Figure 98. Windows Vista x86 Computer Drives

Finally here are two screenshots showing that all of the files from the Windows implementation of the RAID are also present on the Software RAID Virtual Disk implementation.

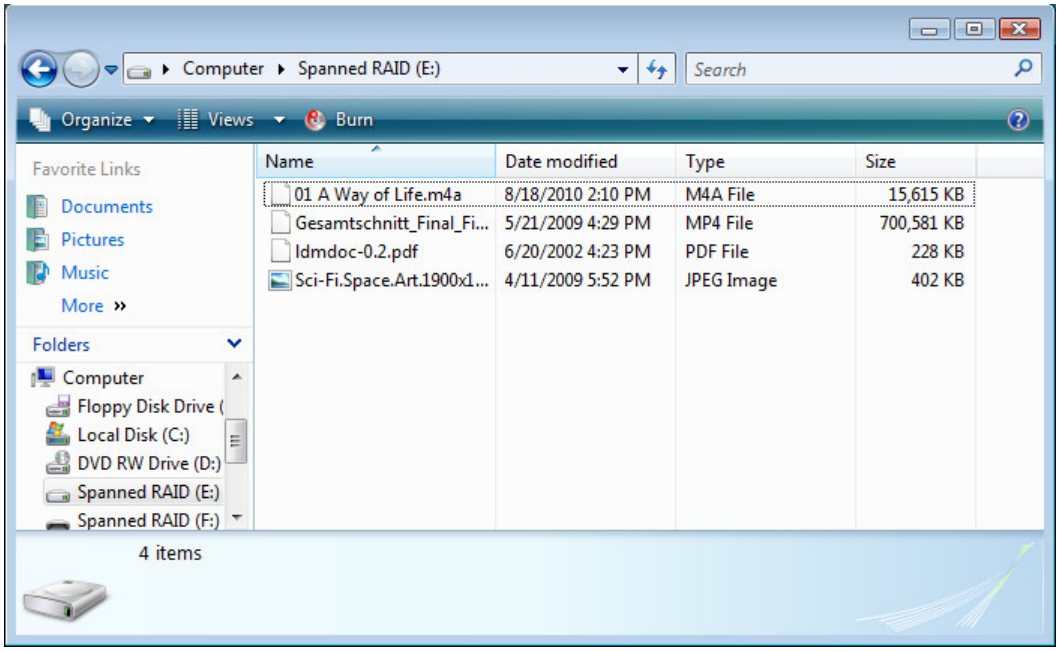



Figure 99. Windows Vista x86 Windows Files

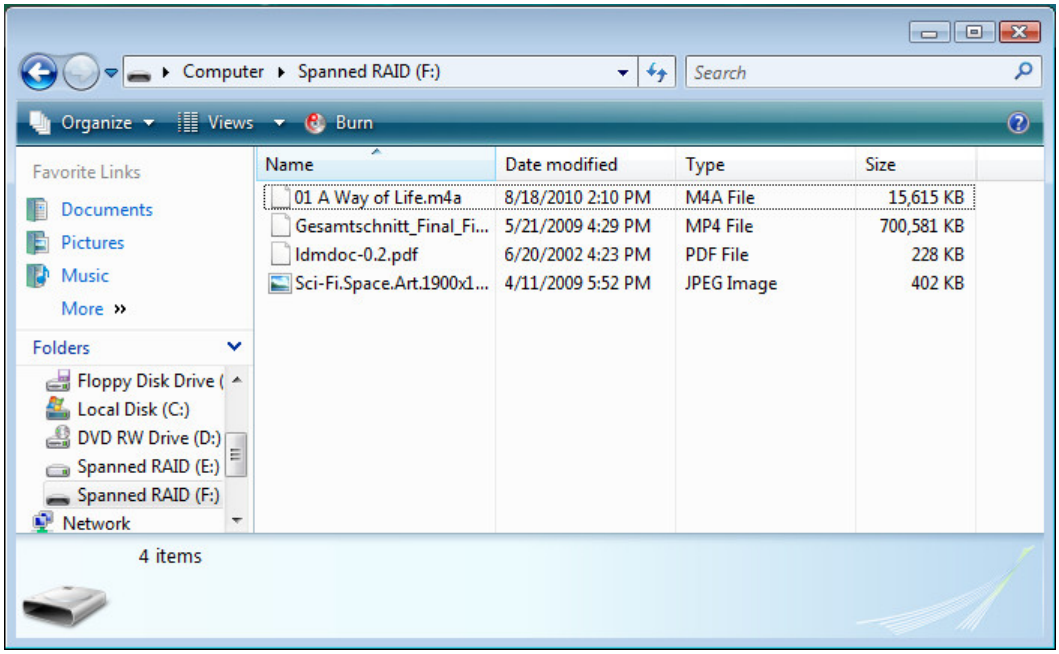



Figure 100. Windows Vista x86 Software RAID Virtual Disk Files

### 4.3.4 Windows Vista x64

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran and after pressing Find Parameters immediately found the volume information as seen in the figure below.

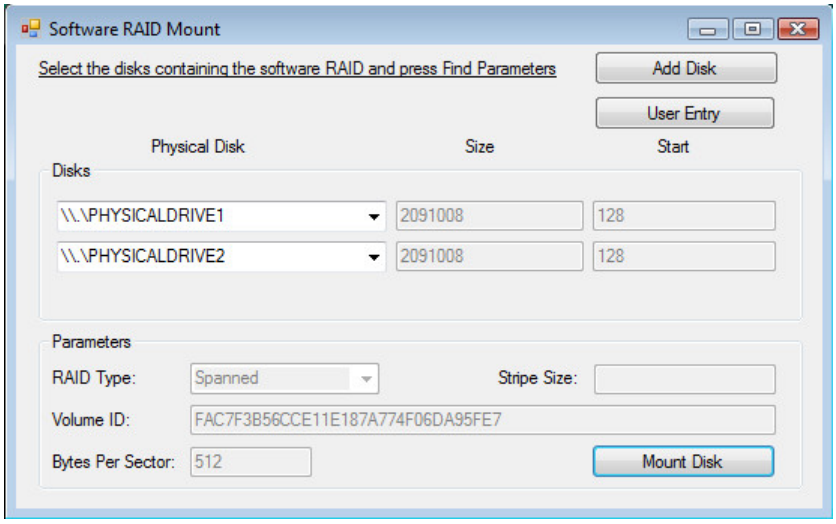



Figure 101. Windows Vista x64 SoftwareRAIDMount.exe

When Mount Disk was pressed, however, nothing happened as the system was not in test mode. After switching into test mode, SoftwareRAIDMount.exe was rerun and it still did not function.

### 4.3.5 Windows Server 2008 x86

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran and after pressing Find Parameters immediately found the volume information as seen in the figure below.

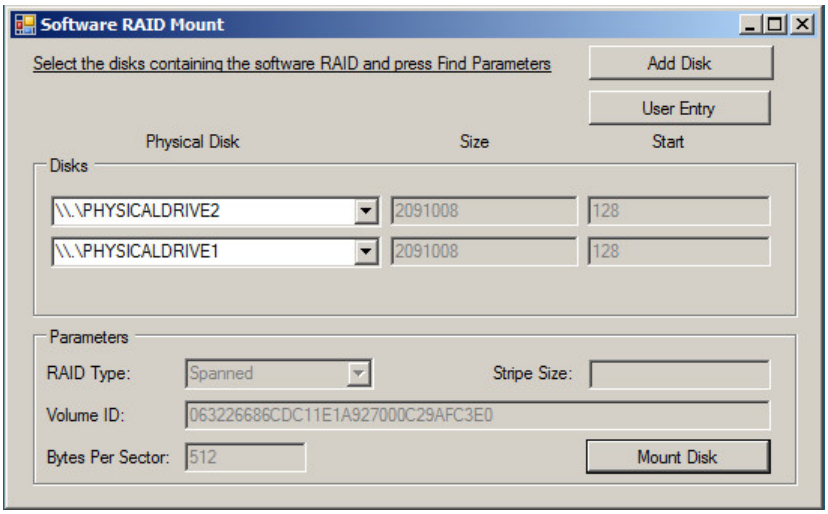



Figure 102. Windows Server 2008 x86 SoftwareRAIDMount.exe

After clicking Mount Disk, there was no error message or confirmation as expected. To verify that the disk was properly mounted, Computer was opened, and as you can see in the next figure, there is now a E: disk under removable disks with the same Volume Name as the Windows RAID.

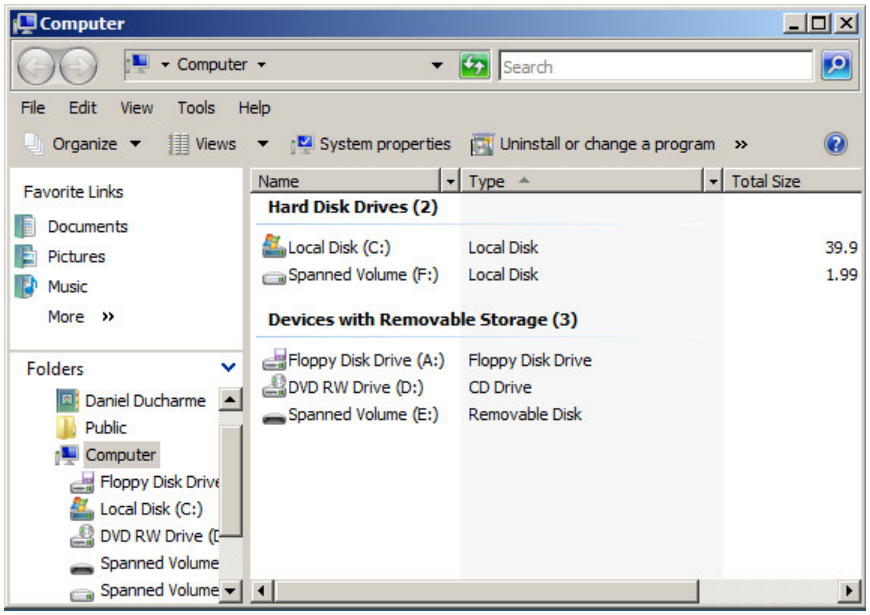



Figure 103. Windows Server 2008 x86 Computer Drives

Finally here are two screenshots showing that all of the files from the Windows

implementation of the RAID are also present on the Software RAID Virtual Disk implementation.

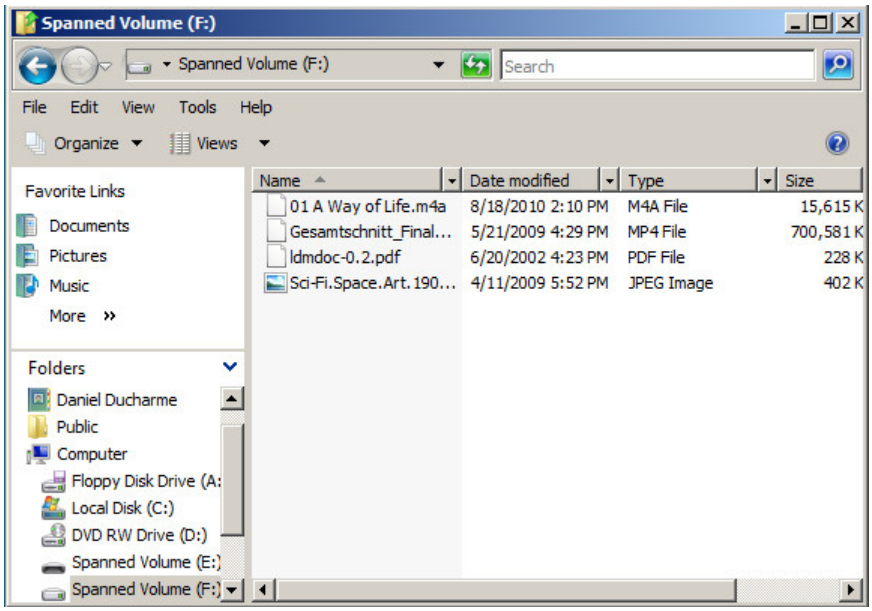



Figure 104. Windows Server 2008 x86 Windows Files

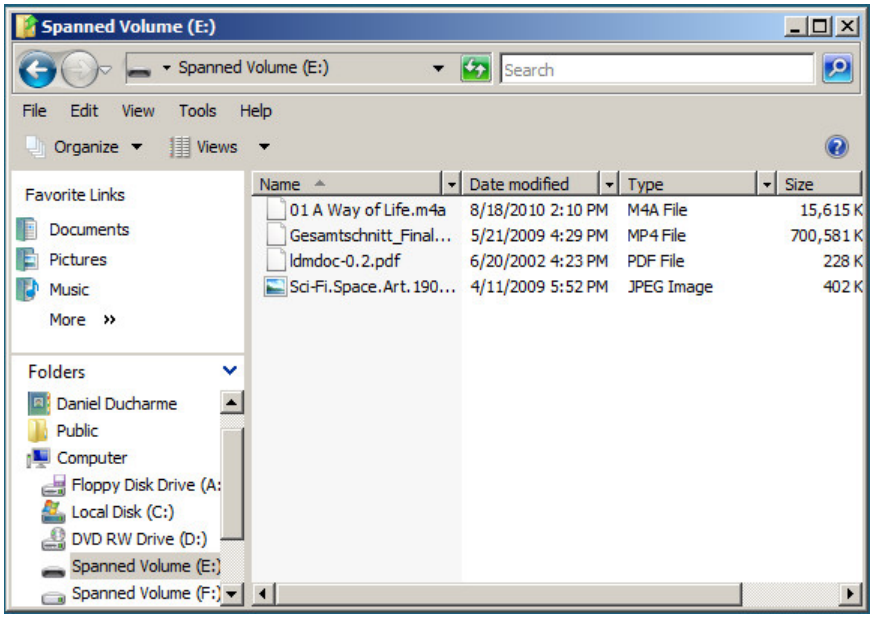



Figure 105. Windows Server 2008 x86 Software RAID Virtual Disk Files

### 4.3.6 Windows Server 2008 x64

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran and after pressing Find Parameters immediately found the volume information as seen in the figure below.

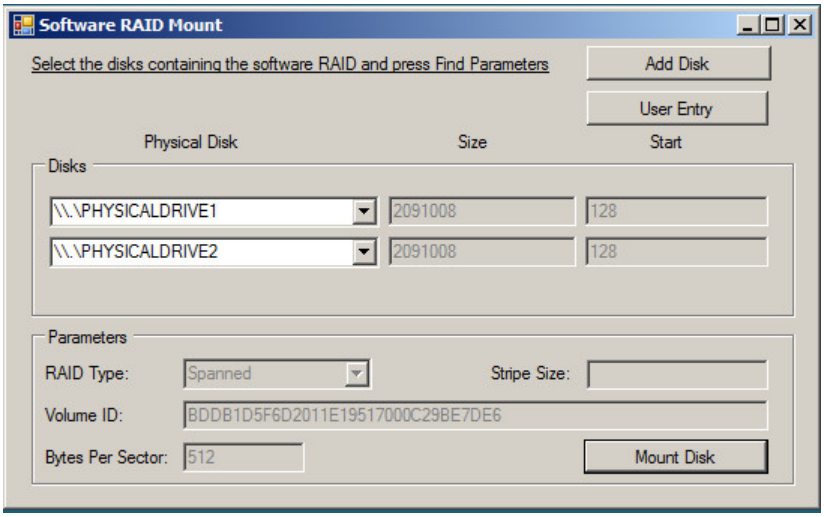



Figure 106. Windows Server 2008 x64 SoftwareRAIDMount.exe

When Mount Disk was pressed, however, nothing happened as the system was not in test mode. After switching into test mode, SoftwareRAIDMount.exe was rerun and it still did not function.

### 4.3.7 Windows 7 x86

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran and after pressing Find Parameters immediately found the volume information as seen in the figure below.

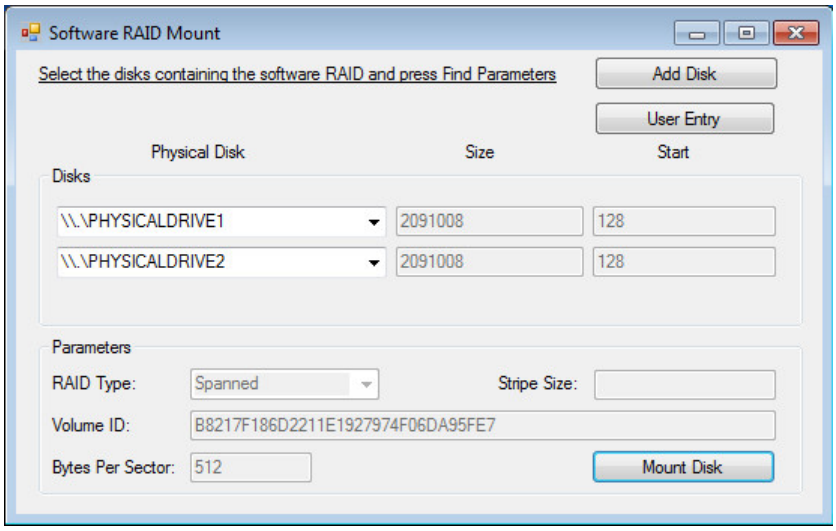



Figure 107. Windows 7 x86 SoftwareRAIDMount.exe

After clicking Mount Disk, there was no error message or confirmation as expected. To verify that the disk was properly mounted, Computer was opened, and as you can see in the next figure, there is now a E: disk under removable disks with the same Volume Name as the Windows RAID.

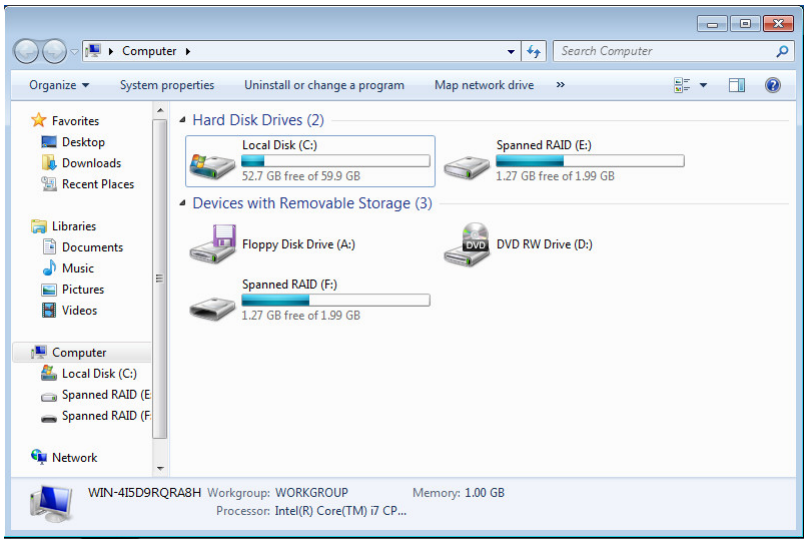



Figure 108. Windows 7 x86 Computer Drives

Finally here are two screenshots showing that all of the files from the Windows implementation of the RAID are also present on the Software RAID Virtual Disk

implementation.

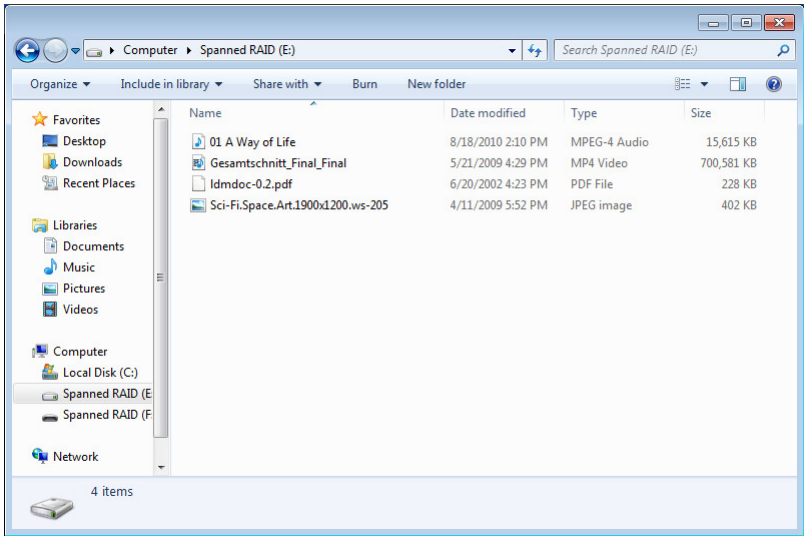



Figure 109. Windows 7 x86 Windows Files

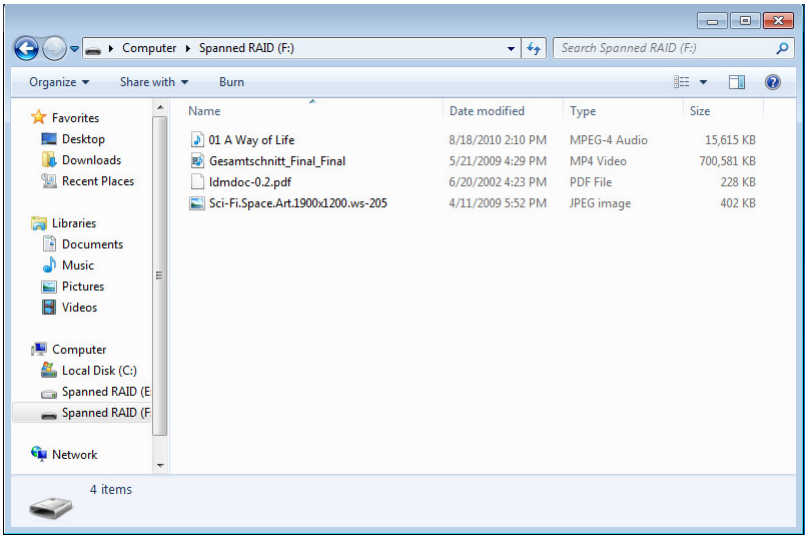



Figure 110. Windows 7 x86 Software RAID Virtual Disk Files

### 4.3.8 Windows 7 x64

After completing setup and placing some test files onto the disk to ensure they were properly mounted, SoftwareRAIDMount.exe was ran and after pressing Find Parameters immediately found the volume information as seen in the figure below.

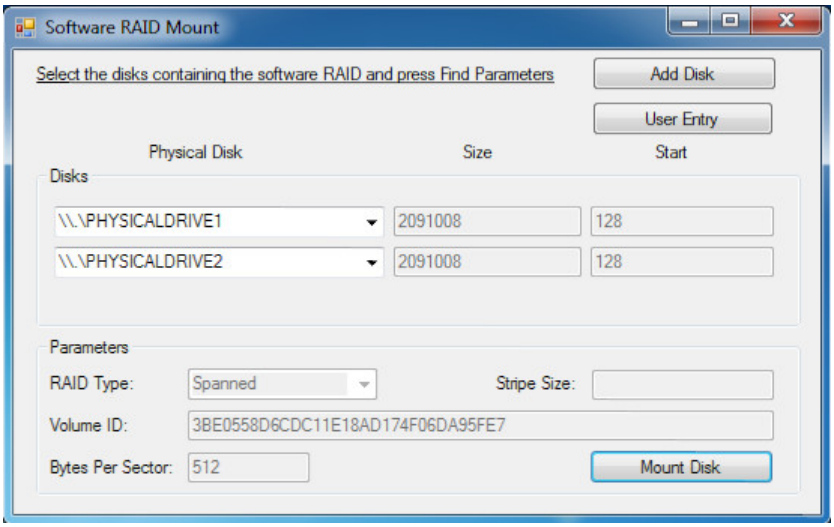



Figure 111. Windows 7 x64 SoftwareRAIDMount.exe

When Mount Disk was pressed, however, nothing happened as the system was not in test mode. After switching into test mode, SoftwareRAIDMount.exe was rerun, and there was no error message or confirmation as expected. To verify that the disk was properly mounted, Computer was opened, and as you can see in the next figure, there is now a G: disk under removable disks with the same Volume Name as the Windows RAID.

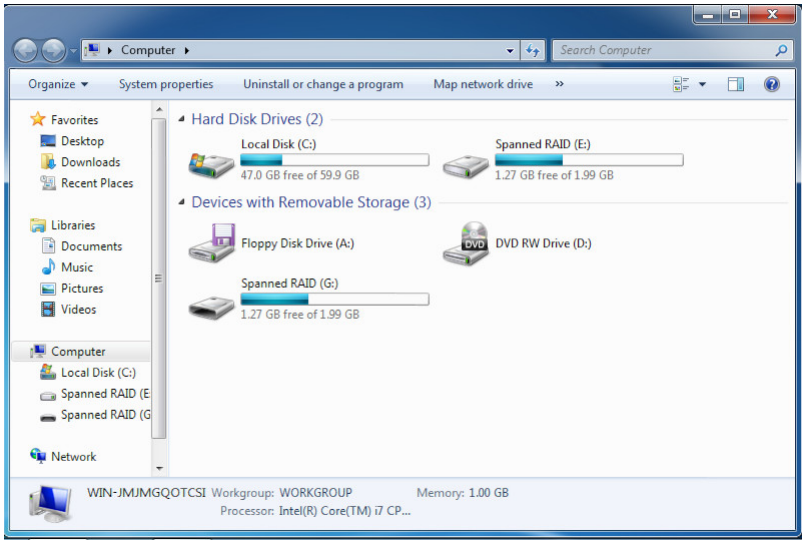



Figure 112. Windows 7 x64 Computer Drives

Finally here are two screenshots showing that all of the files from the Windows implementation of the RAID are also present on the Software RAID Virtual Disk implementation.

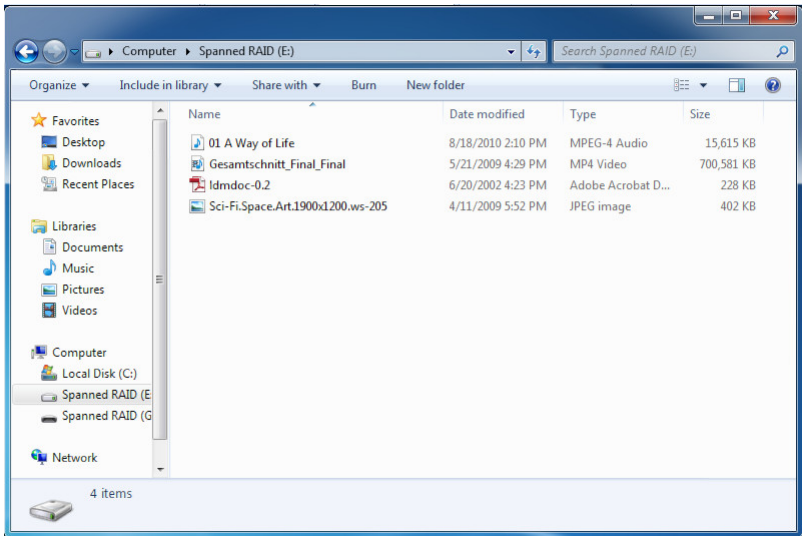



Figure 113. Windows 7 x64 Windows Files

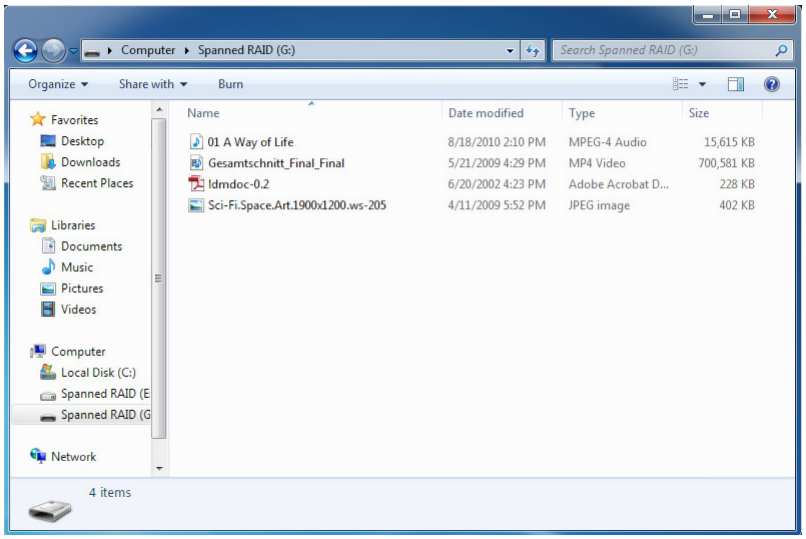



Figure 114. Windows 7 x64 Software RAID Virtual Disk Files

## 4.4 Configuration Compatibility Testing

### 4.4.1 Spanned RAID

The first RAID to test was the Spanned RAID which has been mounted numerous times in other tests. To set the RAID up, 2 disks were added and configured as a Spanned RAID as seen in the next figure.

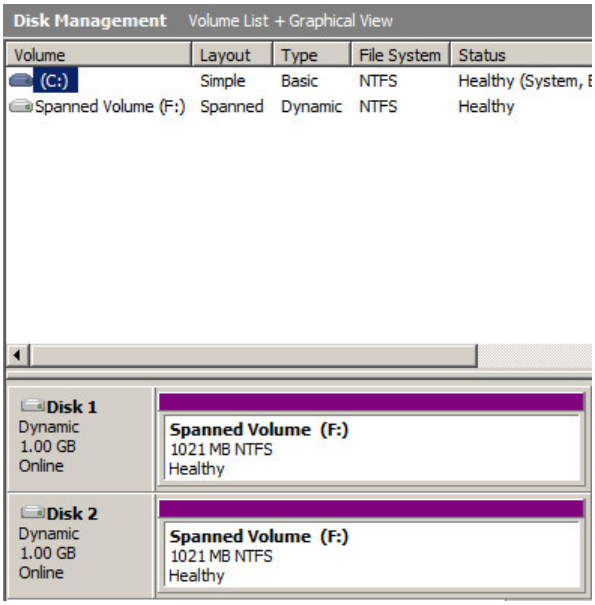



Figure 115. Spanned Disk Setup

Once it was setup, the SoftwareRAIDMount.exe was run and found all of the information.

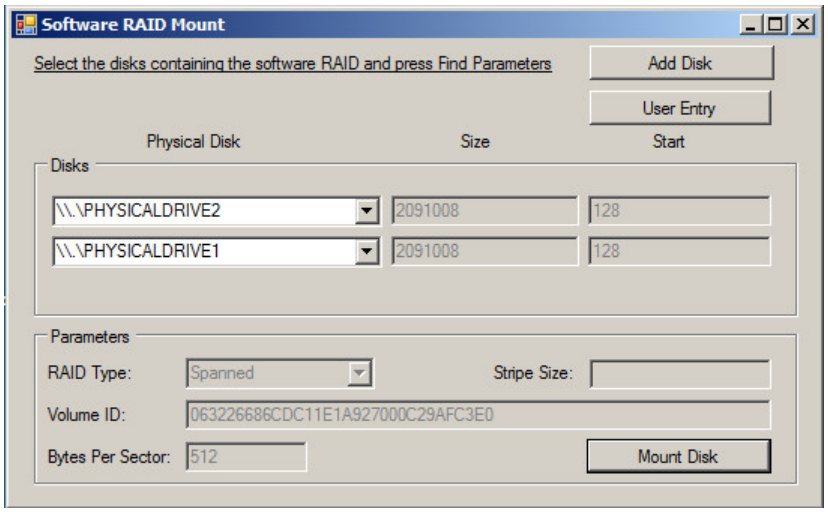



Figure 116. Spanned Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

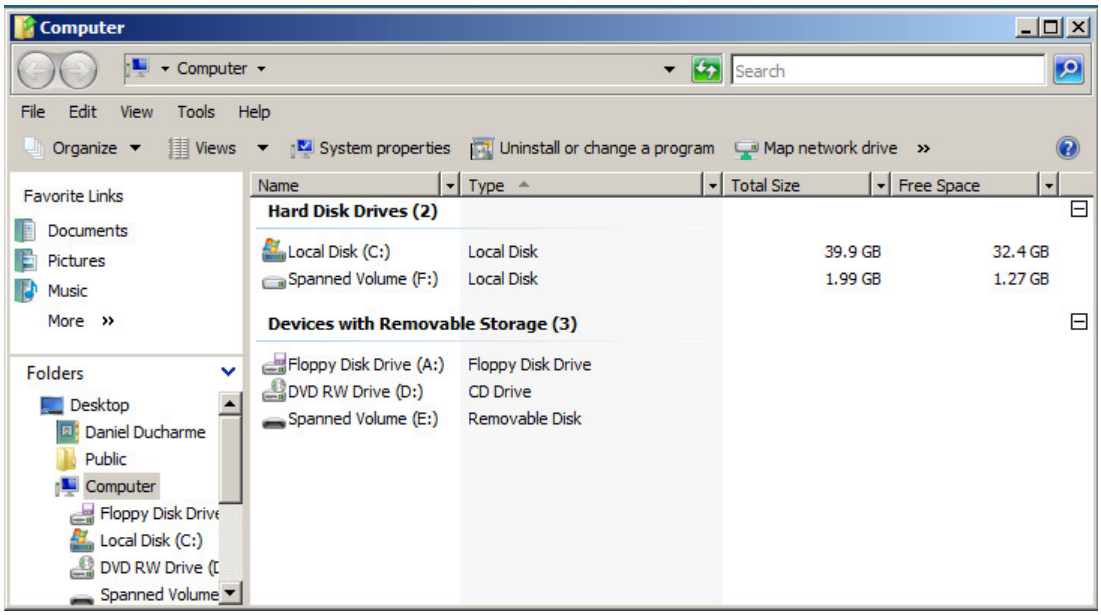



Figure 117. Spanned Disk Mounted in Computer

### 4.4.2 Corrupted Spanned RAID

Next in order to see what corruption the RAID can endure and still mount, the MBR partition table was removed from both disks while the LDM was left

intact.

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000060 | 26 | 66 | 68 | 00 | 00 | 00 | 00 | 66 | FF | 76 | 08 | 68 | 00 | 00 | 68 | 00 |
| 00000070 | 7C | 68 | 01 | 00 | 68 | 10 | 00 | B4 | 42 | 8A | 56 | 00 | 8B | F4 | CD | 13 |
| 00000080 | 9F | 83 | C4 | 10 | 9E | EB | 14 | B8 | 01 | 02 | BB | 00 | 7C | 8A | 56 | 00 |
| 00000090 | 8A | 76 | 01 | 8A | 4E | 02 | 8A | 6E | 03 | CD | 13 | 66 | 61 | 73 | 1E | FE |
| 000000A0 | 4E | 11 | 0F | 85 | 0C | 00 | 80 | 7E | 00 | 80 | 0F | 84 | 8A | 00 | B2 | 80 |
| 000000B0 | EB | 82 | 55 | 32 | E4 | 8A | 56 | 00 | CD | 13 | 5D | EB | 9C | 81 | 3E | FE |
| 000000C0 | 7D | 55 | AA | 75 | 6E | FF | 76 | 00 | E8 | 8A | 00 | 0F | 85 | 15 | 00 | B0 |
| 000000D0 | D1 | E6 | 64 | E8 | 7F | 00 | B0 | DF | E6 | 60 | E8 | 78 | 00 | B0 | FF | E6 |
| 000000E0 | 64 | E8 | 71 | 00 | B8 | 00 | BB | CD | 1A | 66 | 23 | C0 | 75 | 3B | 66 | 81 |
| 000000F0 | FB | 54 | 43 | 50 | 41 | 75 | 32 | 81 | F9 | 02 | 01 | 72 | 2C | 66 | 68 | 07 |
| 00000100 | BB | 00 | 00 | 66 | 68 | 00 | 02 | 00 | 00 | 66 | 68 | 08 | 00 | 00 | 00 | 66 |
| 00000110 | 53 | 66 | 53 | 66 | 55 | 66 | 68 | 00 | 00 | 00 | 00 | 66 | 68 | 00 | 7C | 00 |
| 00000120 | 00 | 66 | 61 | 68 | 00 | 00 | 07 | CD | 1A | 5A | 32 | F6 | EA | 00 | 7C | 00 |
| 00000130 | 00 | CD | 18 | A0 | B7 | 07 | EB | 08 | A0 | B6 | 07 | EB | 03 | A0 | B5 | 07 |
| 00000140 | 32 | E4 | 05 | 00 | 07 | 8B | F0 | AC | 3C | 00 | 74 | FC | BB | 07 | 00 | B4 |
| 00000150 | 0E | CD | 10 | EB | F2 | 2B | C9 | E4 | 64 | EB | 00 | 24 | 02 | E0 | F8 | 24 |
| 00000160 | 02 | C3 | 49 | 6E | 76 | 61 | 6C | 69 | 64 | 20 | 70 | 61 | 72 | 74 | 69 | 74 |
| 00000170 | 69 | 6F | 6E | 20 | 74 | 61 | 62 | 6C | 65 | 00 | 45 | 72 | 72 | 6F | 72 | 20 |
| 00000180 | 6C | 6F | 61 | 64 | 69 | 6E | 67 | 20 | 6F | 70 | 65 | 72 | 61 | 74 | 69 | 6E |
| 00000190 | 67 | 20 | 73 | 79 | 73 | 74 | 65 | 6D | 00 | 4D | 69 | 73 | 73 | 69 | 6E | 67 |
| 000001A0 | 20 | 6F | 70 | 65 | 72 | 61 | 74 | 69 | 6E | 67 | 20 | 73 | 79 | 73 | 74 | 65 |
| 000001B0 | 6D | 00 | 00 | 00 | 00 | 62 | 7A | 99 | 03 | C2 | 10 | FB | 00 | 00 | 00 | 00 |
| 000001C0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 000001D0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 000001E0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 000001F0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 55 | AA |

Figure 118. Corrupted MBR Setup

The system was then restarted in order to cause the disks to fail. The SoftwareRAIDMount.exe was still able to find the information as seen in the next figure.

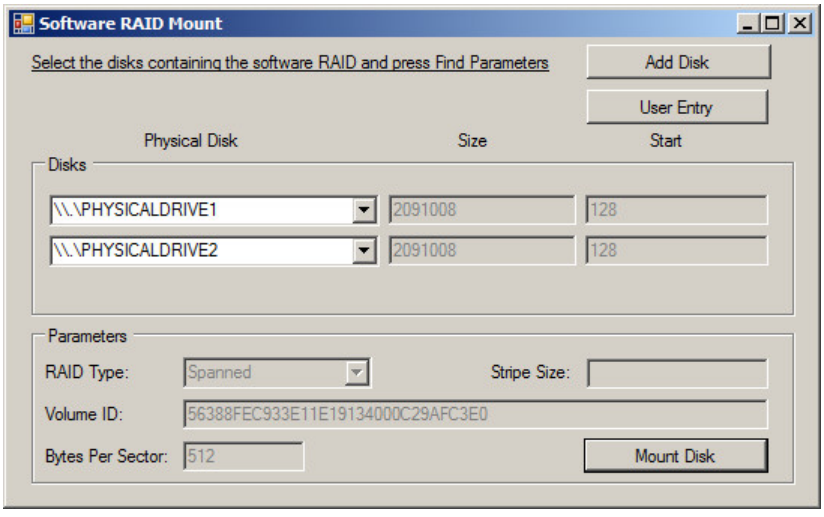



Figure 119. Corrupted MBR Disk Mount Information

After running the mounting program, the volume showed up under Computer.

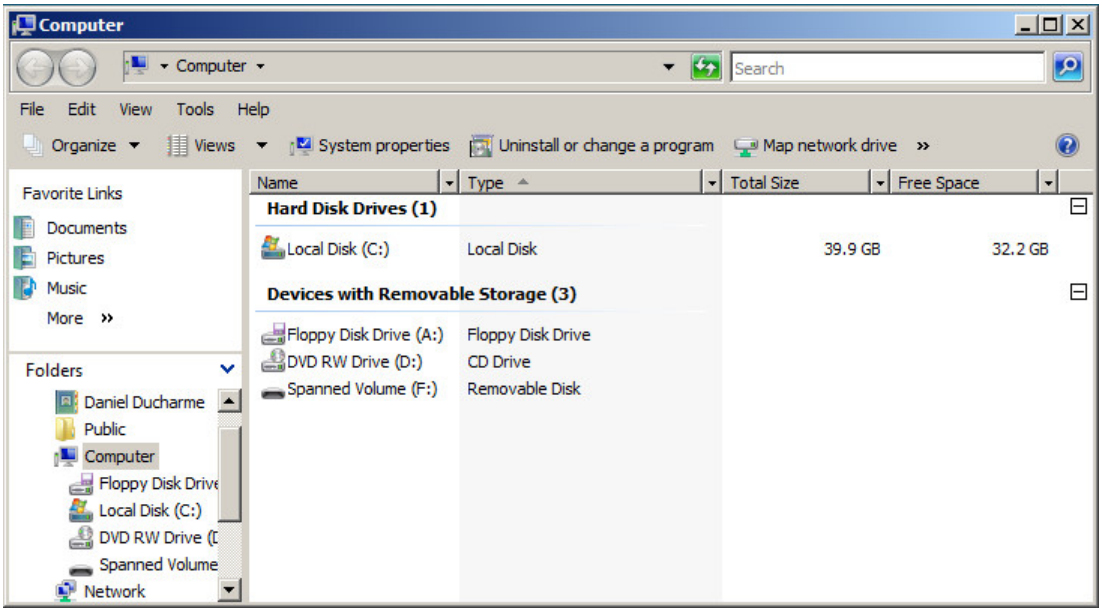



Figure 120. Corrupted MBR Disk Mounted in Computer

The next corruption to test is the removal of the LDM database. This was done by removing the information from the TOCBLOCK and the PRIVHEAD as seen in the next 2 figures.

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3FFFFA00 | 54 | 4F | 43 | 42 | 4C | 4F | 43 | 4B | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | TOCBLOCK |
| 3FFFFA10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA20 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA50 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA60 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA70 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA80 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFA90 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAA0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAB0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAC0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAD0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAE0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFAF0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB20 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB50 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB60 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB70 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB80 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFB90 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |

Figure 121. Corrupted Disk TOCBLOCK Removed

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3FFFFE00 | 50 | 52 | 49 | 56 | 48 | 45 | 41 | 44 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | PRIVHEAD |
| 3FFFFE10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE20 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE50 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE60 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE70 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE80 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFE90 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFEA0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFEB0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFEC0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFED0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFEE0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFEF0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF20 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF30 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF50 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF60 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF70 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF80 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 3FFFFF90 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |

Figure 122. Corrupted Disk PRIVHEAD Removed

Since the LDM database was no longer present, the mounting information was entered into the SoftwareRAIDMount.exe as shown in the next figure.

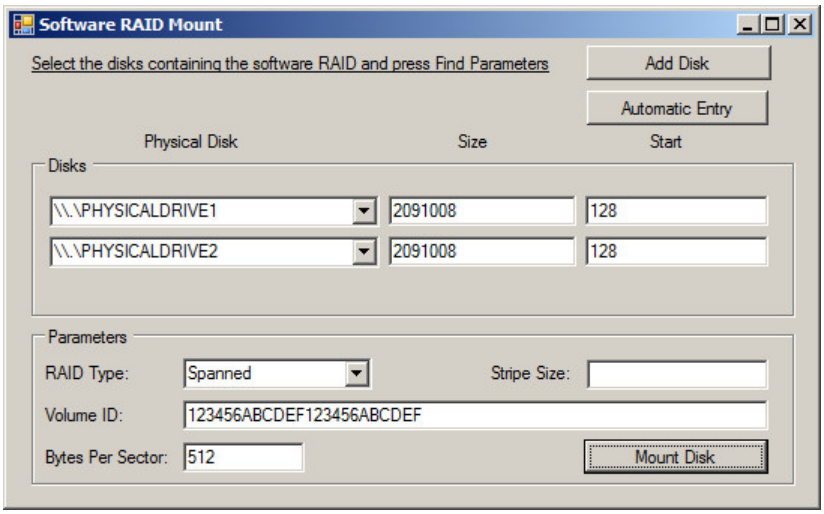



Figure 123. Corrupted LDM Disk Mount Information

After running the mounting program, the volume showed up under Computer.

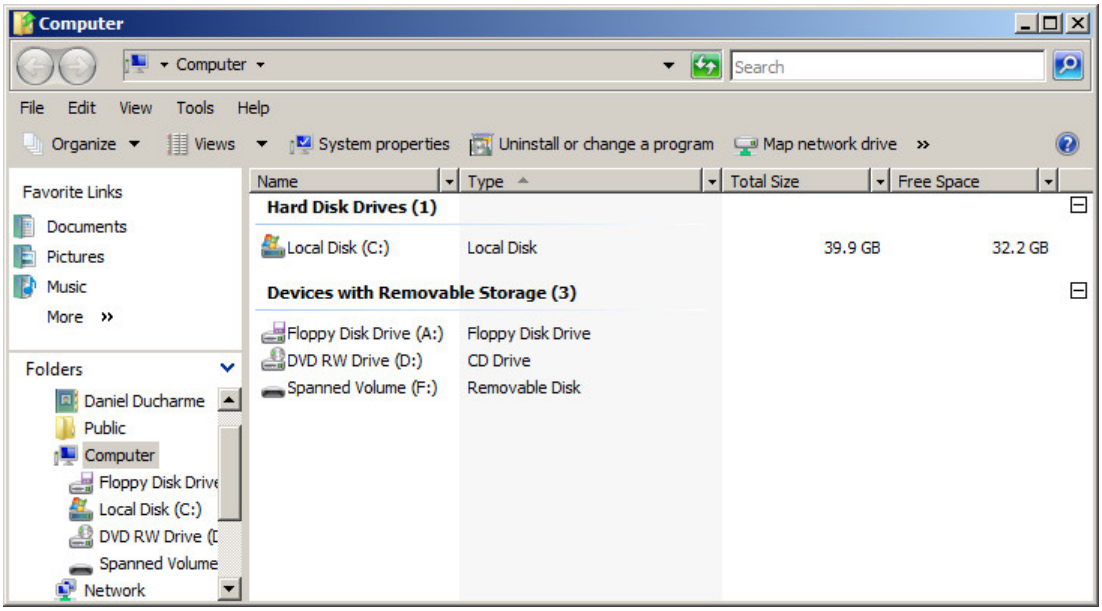



Figure 124. Corrupted LDM Disk Mounted in Computer

### 4.4.3 GPT Spanned RAID

After the corrupted RAID the next test was to see if the driver could handle a GPT disk. To set the RAID up, first 2 disks were added and the disks were converted to GPT disks by right clicking on them and selecting Convert to GPT.

Then the disks were configured as a Spanned RAID as seen in the next figure.

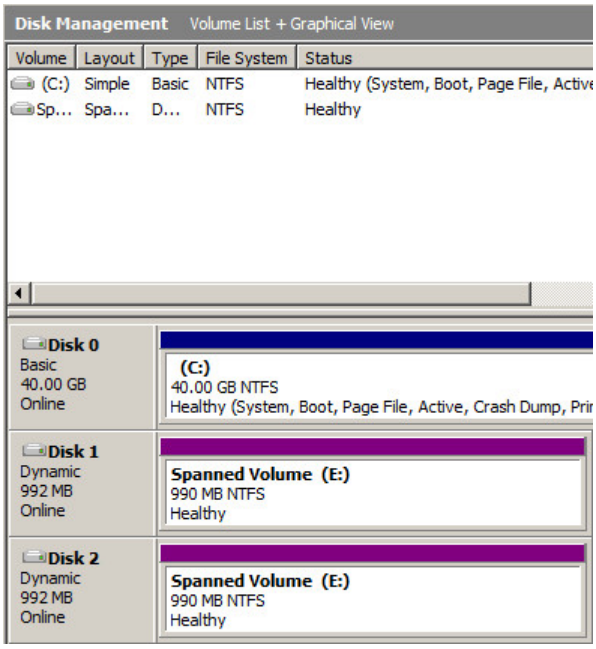
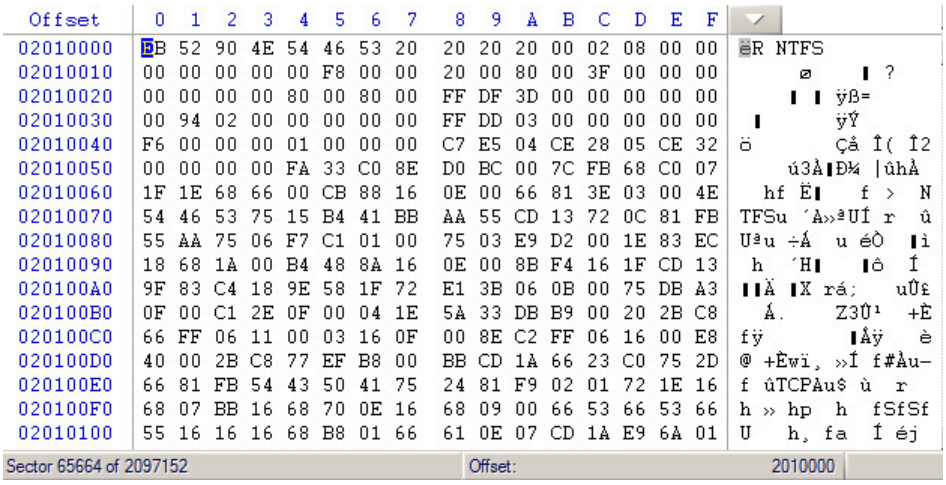



Figure 125. GPT Spanned Disk Setup

Once the GPT partitions were configured it was necessary to find the manual parameters as the SoftwareRAIDMount.exe is currently incapable of parsing the data structures. In order to do this WinHex was used to view the raw bytes as well as to apply templates quickly parsing the data. The first piece of information that was needed was the start sector, this was easy to find by searching the disk for NTFS. As you can see in the below figure, the start sector of the NTFS volumes on these GPT disks was 65664, which can be seen in the bottom left hand corner.



| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 02010000 | EB | 52 | 90 | 4E | 54 | 46 | 53 | 20 | 20 | 20 | 20 | 00 | 02 | 08 | 00 | 00 | ëR NTFS |
| 02010010 | 00 | 00 | 00 | 00 | 00 | F8 | 00 | 00 | 20 | 00 | 80 | 00 | 3F | 00 | 00 | 00 | ø ▌ ? |
| 02010020 | 00 | 00 | 00 | 00 | 80 | 00 | 80 | 00 | FF | DF | 3D | 00 | 00 | 00 | 00 | 00 | ▌▌ ÿß= |
| 02010030 | 00 | 94 | 02 | 00 | 00 | 00 | 00 | 00 | FF | DD | 03 | 00 | 00 | 00 | 00 | 00 | ▌ ÿÝ |
| 02010040 | F6 | 00 | 00 | 00 | 01 | 00 | 00 | 00 | C7 | E5 | 04 | CE | 28 | 05 | CE | 32 | ö Çå Î( Î2 |
| 02010050 | 00 | 00 | 00 | 00 | FA | 33 | C0 | 8E | D0 | BC | 00 | 7C | FB | 68 | C0 | 07 | ú3À▌Ð¼ \|ûhÀ |
| 02010060 | 1F | 1E | 68 | 66 | 00 | CB | 88 | 16 | 0E | 00 | 66 | 81 | 3E | 03 | 00 | 4E | hf Ë▌ f > N |
| 02010070 | 54 | 46 | 53 | 75 | 15 | B4 | 41 | BB | AA | 55 | CD | 13 | 72 | 0C | 81 | FB | TFSu ´A»ªUÍ r û |
| 02010080 | 55 | AA | 75 | 06 | F7 | C1 | 01 | 00 | 75 | 03 | E9 | D2 | 00 | 1E | 83 | EC | Uªu ÷Á u éÒ ▌ì |
| 02010090 | 18 | 68 | 1A | 00 | B4 | 48 | 8A | 16 | 0E | 00 | 8B | F4 | 16 | 1F | CD | 13 | h ´H▌ ▌ô Í |
| 020100A0 | 9F | 83 | C4 | 18 | 9E | 58 | 1F | 72 | E1 | 3B | 06 | 0B | 00 | 75 | DB | A3 | ▌▌Ä ▌X rá; uÛ£ |
| 020100B0 | 0F | 00 | C1 | 2E | 0F | 00 | 04 | 1E | 5A | 33 | DB | B9 | 00 | 20 | 2B | C8 | Á. Z3Û¹ +È |
| 020100C0 | 66 | FF | 06 | 11 | 00 | 03 | 16 | 0F | 00 | 8E | C2 | FF | 06 | 16 | 00 | E8 | fÿ ▌Åÿ è |
| 020100D0 | 40 | 00 | 2B | C8 | 77 | EF | B8 | 00 | BB | CD | 1A | 66 | 23 | C0 | 75 | 2D | @ +Èwï¸ »Í f#Àu- |
| 020100E0 | 66 | 81 | FB | 54 | 43 | 50 | 41 | 75 | 24 | 81 | F9 | 02 | 01 | 72 | 1E | 16 | f ûTCPAu$ ù r |
| 020100F0 | 68 | 07 | BB | 16 | 68 | 70 | 0E | 16 | 68 | 09 | 00 | 66 | 53 | 66 | 53 | 66 | h » hp h fSfSf |
| 02010100 | 55 | 16 | 16 | 16 | 68 | B8 | 01 | 66 | 61 | 0E | 07 | CD | 1A | E9 | 6A | 01 | U h¸ fa Í éj |

Sector 65664 of 2097152 Offset: 2010000

Figure 126. GPT Spanned Disk Start Sector

Once we have found the boot sector of the NTFS volume as well as the start sector, then we can apply the Boot Sector NTFS Template from WinHex to quickly parse all of the NTFS boot sector data. One of the fields, seen in the next figure, is labelled Total Sectors which is the field needed to calculate the sizes. Because the field is 0 indexed the value of 4055039 needs to be increased to 4055040, then it is divided by two to give the 2027520 that is the number of sectors used on each disk.

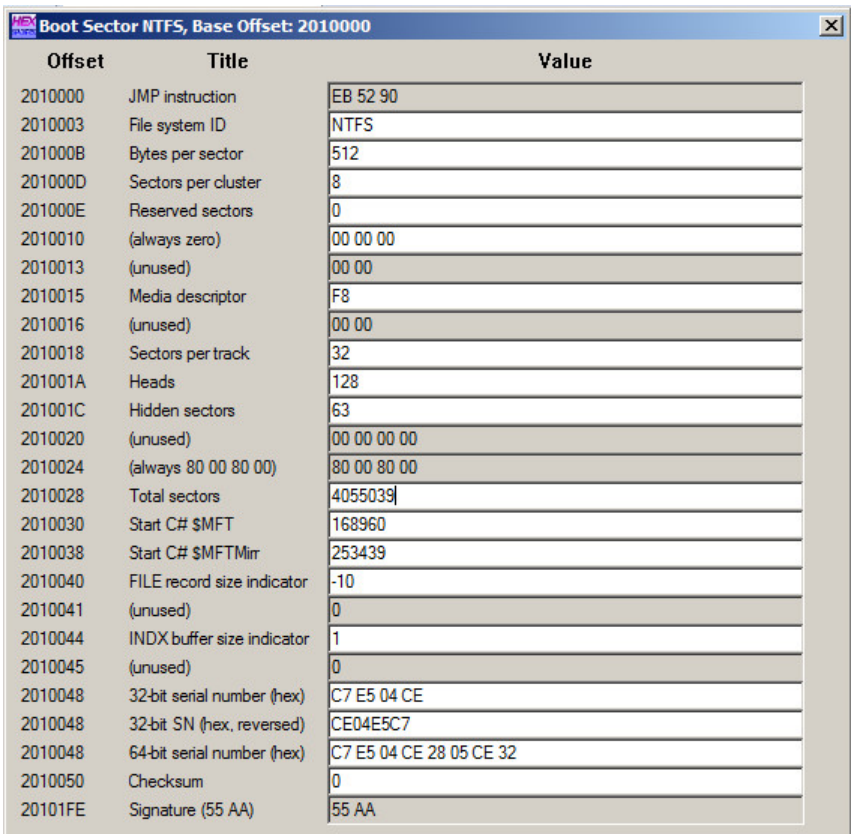



Figure 127. GPT Spanned Disk Size

Once the information was gathered, the SoftwareRAIDMount.exe was run and all of the information was entered.

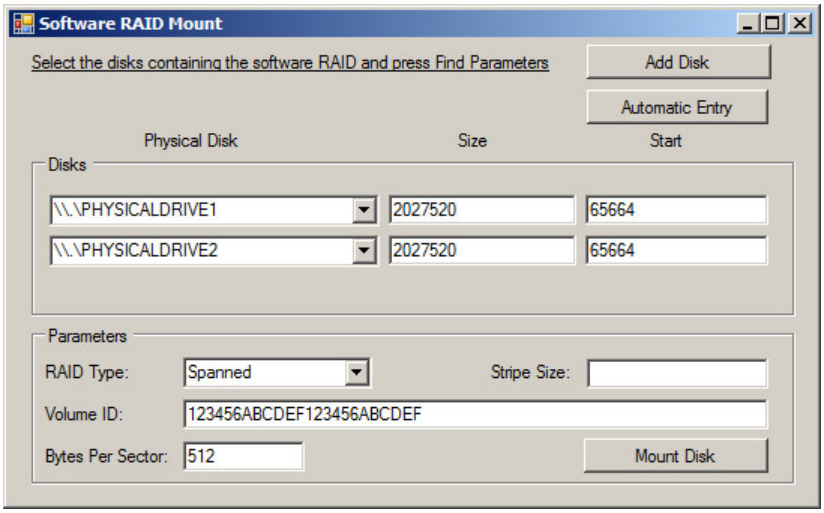



Figure 128. GPT Spanned Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

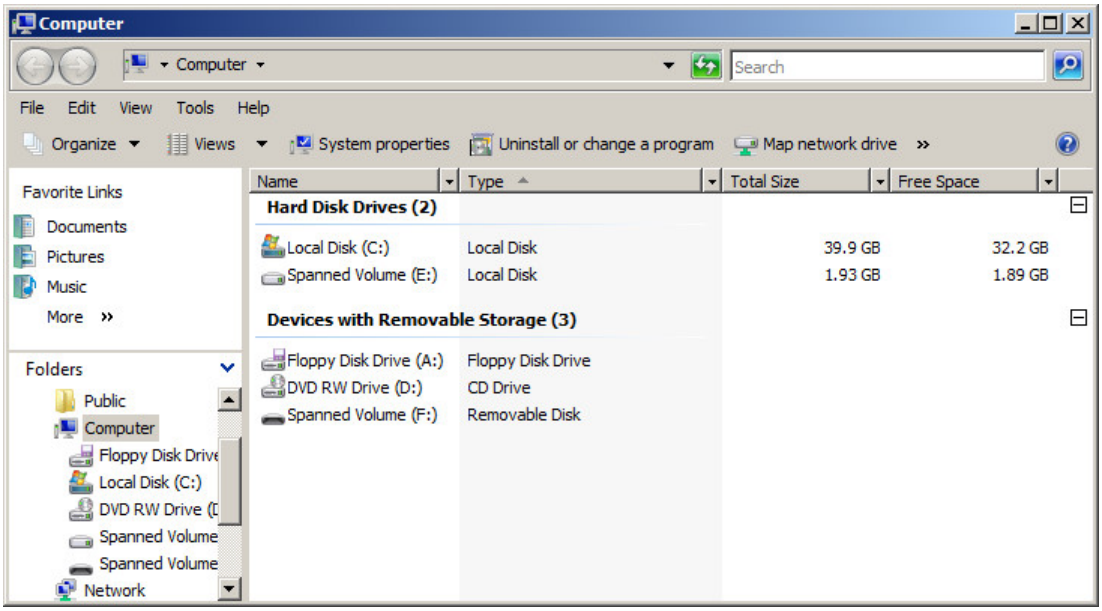



Figure 129. GPT Spanned Disk Mounted in Computer

### 4.4.4 Striped RAID

The next RAID to test was the Striped RAID which again has been mounted numerous times in other tests. To set the RAID up, 2 disks were added and configured as a Striped RAID as seen in the next figure.

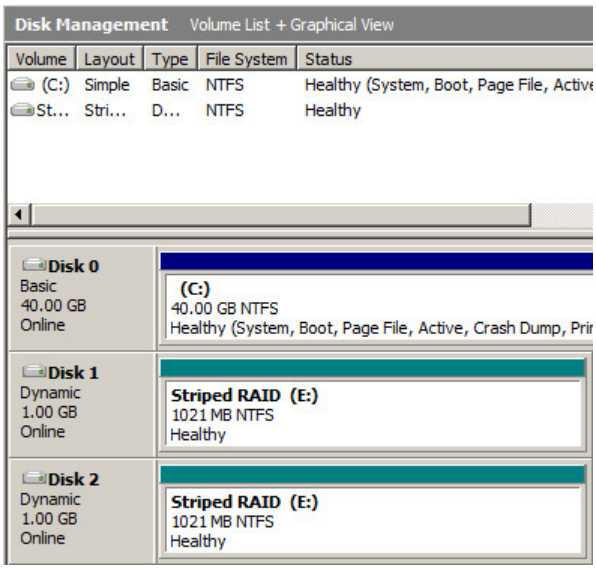



Figure 130. Striped Disk Setup

Once it was setup, the SoftwareRAIDMount.exe was run and found all of the information.

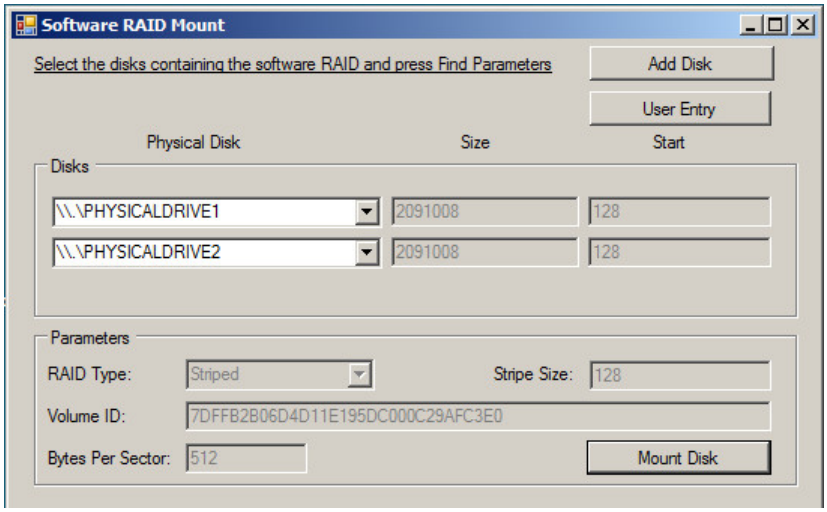



Figure 131. Striped Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

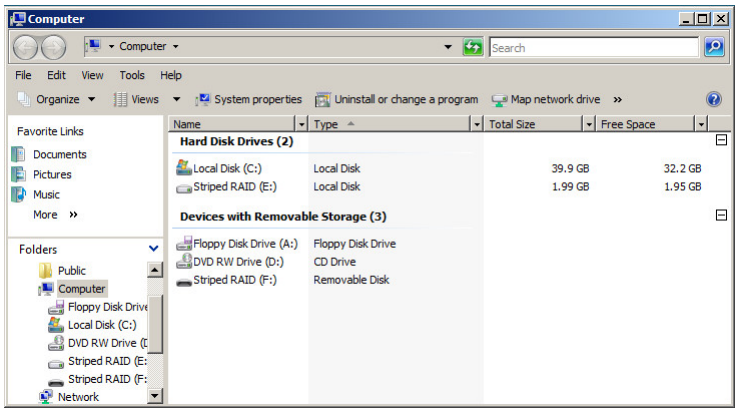



Figure 132. Striped Disk Mounted in Computer

### 4.4.5 Simple RAID

Next tested was the Simple RAID which is just a volume on a single dynamic disk. To set the RAID up, 1 disk was added and configured as a Simple RAID as seen in the next figure.

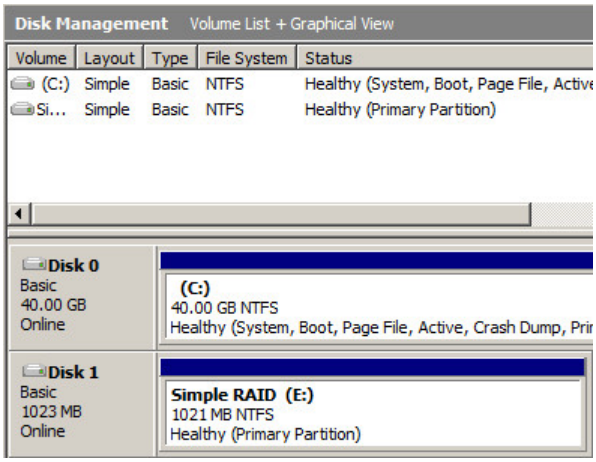



Figure 133. Simple Disk Setup

Once it was setup, the SoftwareRAIDMount.exe was run and found all of the information although it is currently incorrectly labelling it as a spanned RAID. This is not a problem as you will only ever pass in that disk.

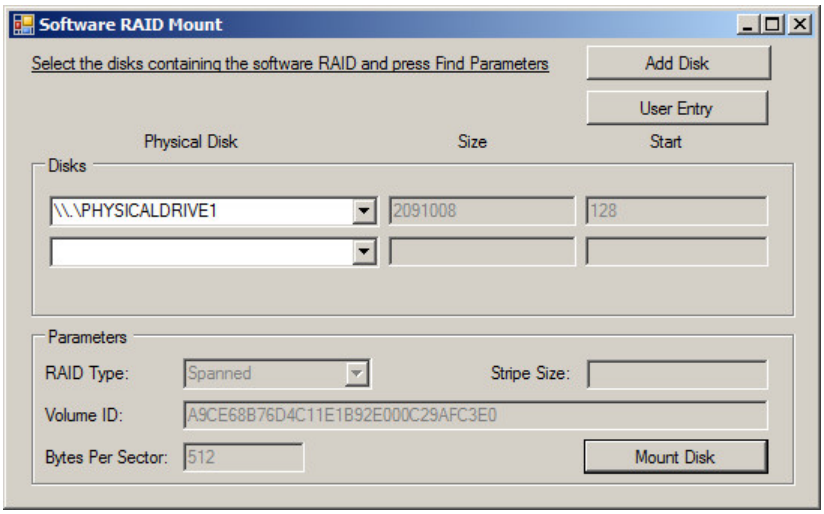



Figure 134. Simple Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

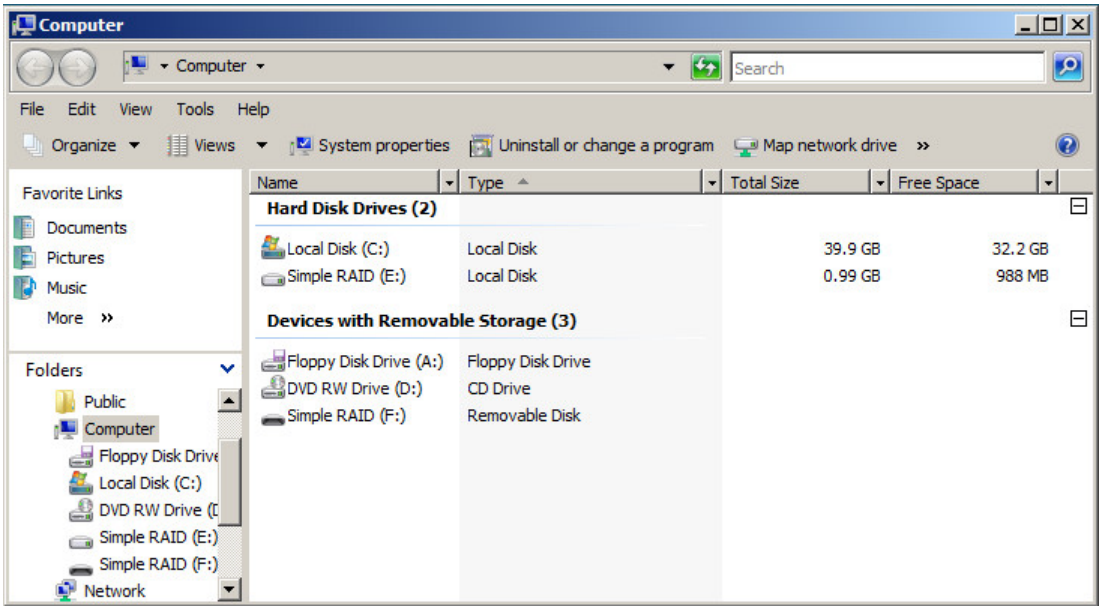



Figure 135. Simple Disk Mounted in Computer

### 4.4.6 Mirrored RAID

Next tested was the Mirrored RAID. To set the RAID up, 2 disks were added and configured as a Mirrored RAID as seen in the next figure.

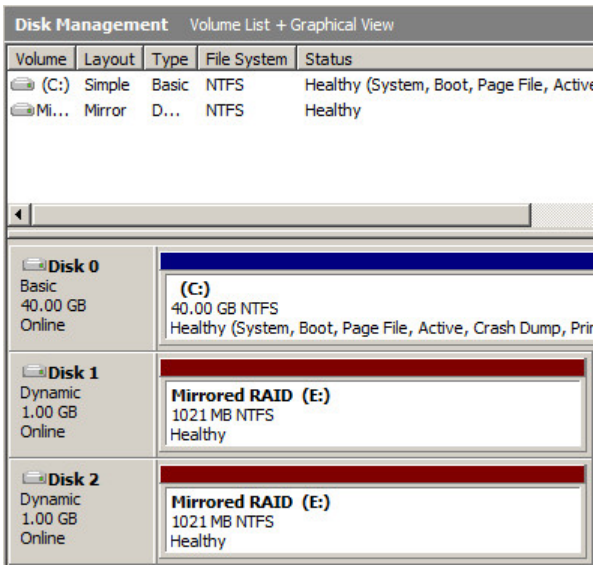



Figure 136. Mirrored Disk Setup

Once it was setup, the SoftwareRAIDMount.exe was run and found all of the information although it is currently incorrectly labelling it as a spanned RAID. This is not a problem as long as you only mount either 1 of the disks but will be an issue if you incorrectly pass in both disks.

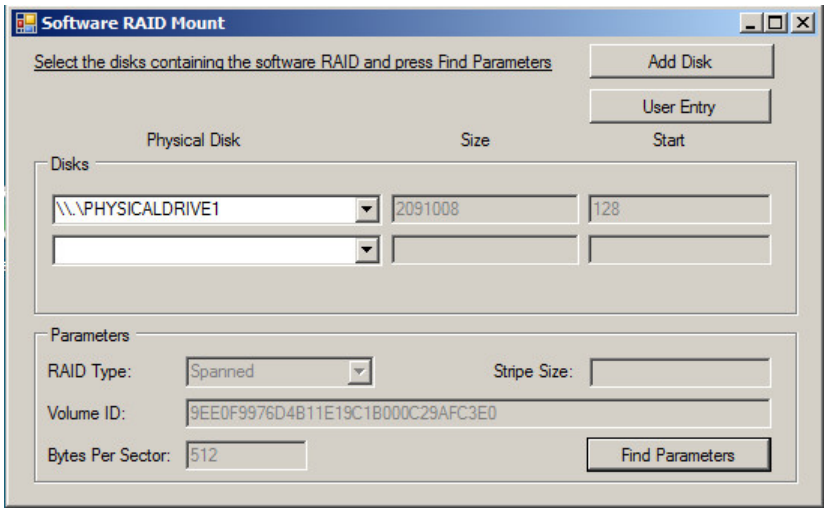



Figure 137. Mirrored Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

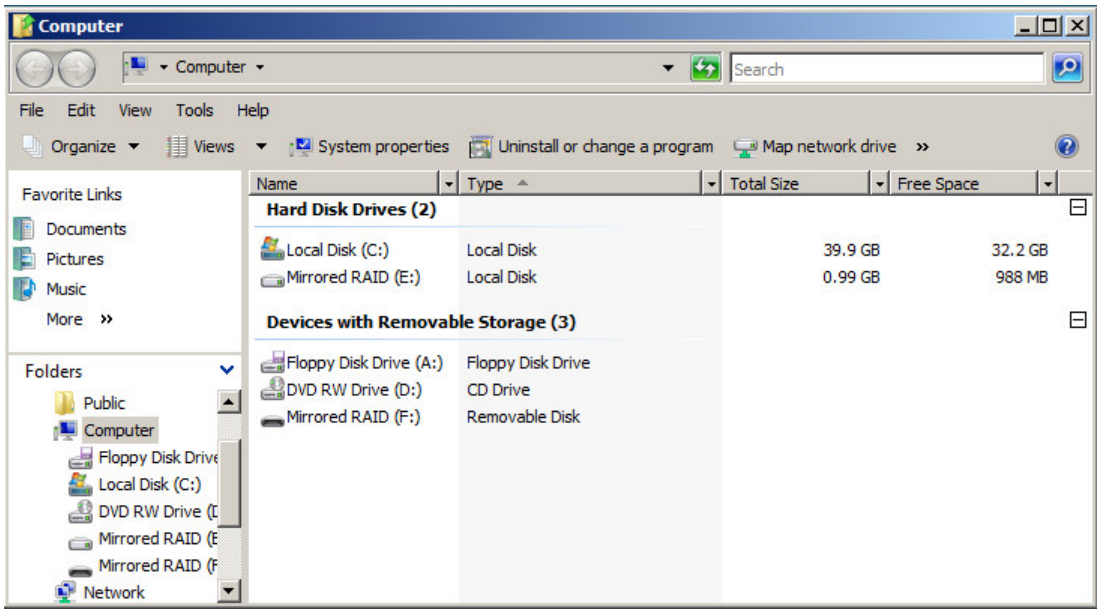



Figure 138. Mirrored Disk Mounted in Computer

### 4.4.7 Multiple Disk Striped RAID

The next RAID to test was the Striped RAID on more than 2 disks. To set the RAID up, 3 disks were added and configured as a Striped RAID as seen in the next figure.

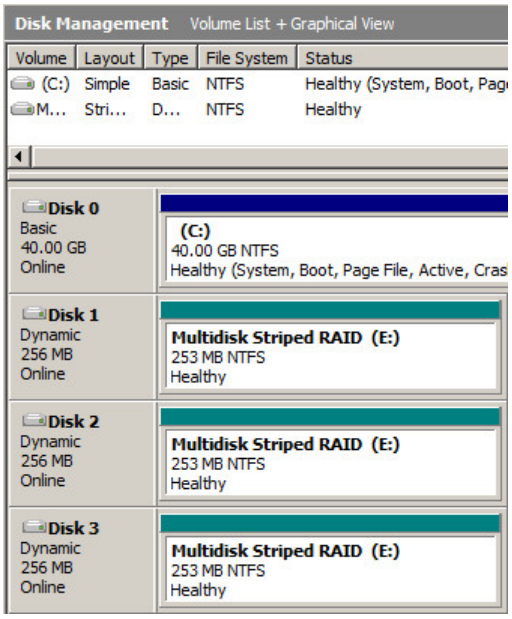



Figure 139. Multidisk Striped Disk Setup

Once it was setup, the SoftwareRAIDMount.exe was run and found all of the information.

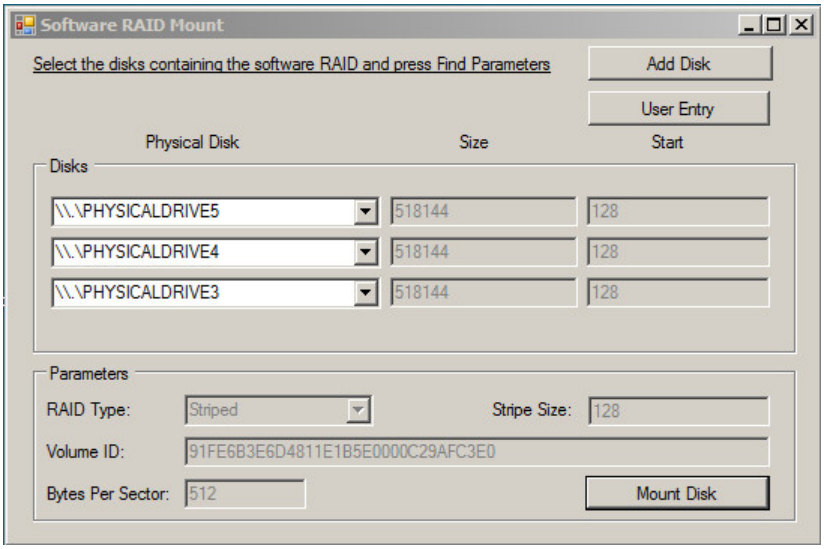



Figure 140. Multidisk Striped Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

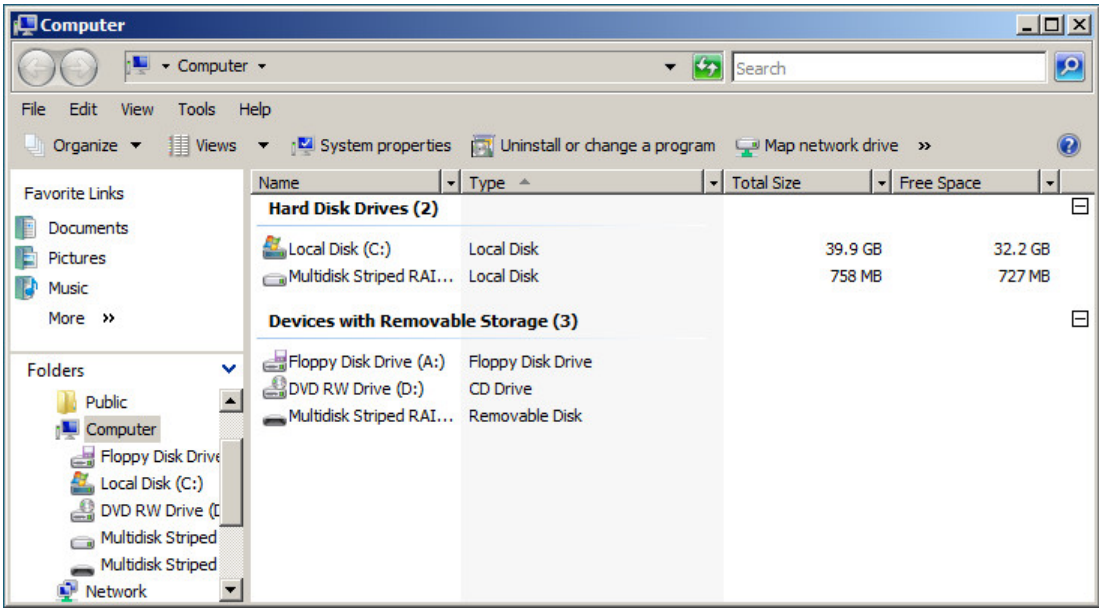



Figure 141. Multidisk Striped Disk Mounted in Computer

### 4.4.8 Linux RAID

When we first booted into Windows Server Manager was checked to ensure that the two partitions that were created in the Ubuntu install were present as seen in the next figure.

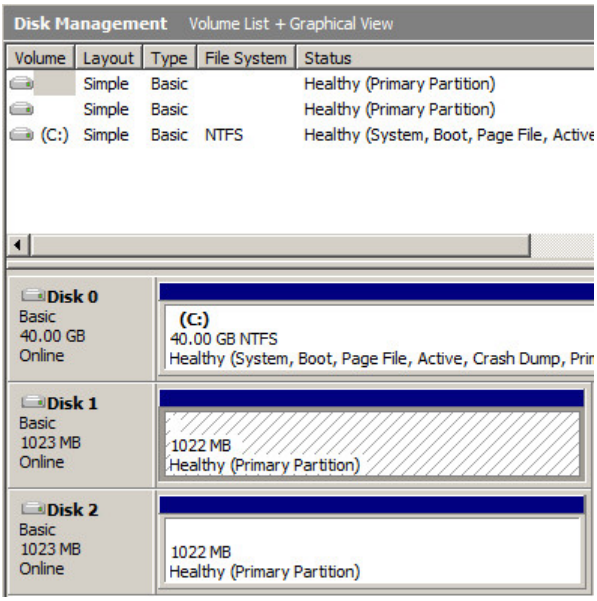
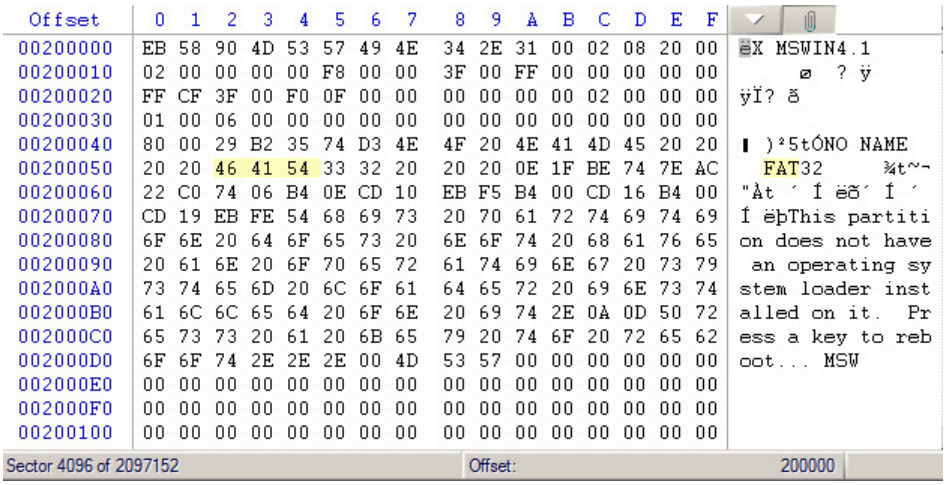



Figure 142. Linux Disk Setup

Because these are Linux partitions it was necessary to find the manual parameters as the SoftwareRAIDMount.exe is currently incapable of parsing the data structures. In order to do this WinHex was used to view the raw bytes as well as to apply templates quickly parsing the data. The first piece of information that was needed was the start sector, this was easy to find by searching the disk for FAT which was found twice. As you can see in the below figure, the start sector of the FAT32 volumes on these disks was 4096, which can be seen in the bottom left hand corner.

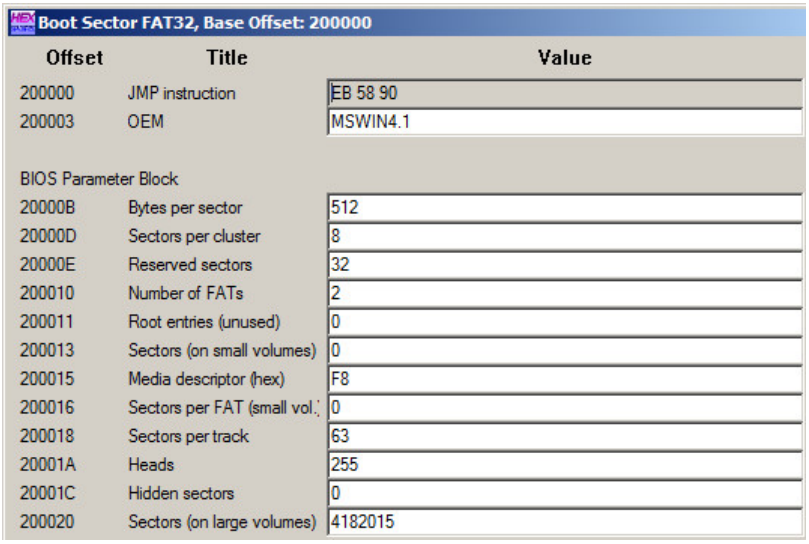


| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00200000 | EB | 58 | 90 | 4D | 53 | 57 | 49 | 4E | 34 | 2E | 31 | 00 | 02 | 08 | 20 | 00 | ëX MSWIN4.1 |
| 00200010 | 02 | 00 | 00 | 00 | 00 | F8 | 00 | 00 | 3F | 00 | FF | 00 | 00 | 00 | 00 | 00 | ø ? ÿ |
| 00200020 | FF | CF | 3F | 00 | F0 | 0F | 00 | 00 | 00 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | ÿÏ? ð |
| 00200030 | 01 | 00 | 06 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00200040 | 80 | 00 | 29 | B2 | 35 | 74 | D3 | 4E | 4F | 20 | 4E | 41 | 4D | 45 | 20 | 20 | ▌ )²5tÓNO NAME |
| 00200050 | 20 | 20 | 46 | 41 | 54 | 33 | 32 | 20 | 20 | 20 | 0E | 1F | BE | 74 | 7E | AC | FAT32 ¾t~¬ |
| 00200060 | 22 | C0 | 74 | 06 | B4 | 0E | CD | 10 | EB | F5 | B4 | 00 | CD | 16 | B4 | 00 | "Àt ´ Í ëõ´ Í ´ |
| 00200070 | CD | 19 | EB | FE | 54 | 68 | 69 | 73 | 20 | 70 | 61 | 72 | 74 | 69 | 74 | 69 | Í ëþThis partiti |
| 00200080 | 6F | 6E | 20 | 64 | 6F | 65 | 73 | 20 | 6E | 6F | 74 | 20 | 68 | 61 | 76 | 65 | on does not have |
| 00200090 | 20 | 61 | 6E | 20 | 6F | 70 | 65 | 72 | 61 | 74 | 69 | 6E | 67 | 20 | 73 | 79 | an operating sy |
| 002000A0 | 73 | 74 | 65 | 6D | 20 | 6C | 6F | 61 | 64 | 65 | 72 | 20 | 69 | 6E | 73 | 74 | stem loader inst |
| 002000B0 | 61 | 6C | 6C | 65 | 64 | 20 | 6F | 6E | 20 | 69 | 74 | 2E | 0A | 0D | 50 | 72 | alled on it. Pr |
| 002000C0 | 65 | 73 | 73 | 20 | 61 | 20 | 6B | 65 | 79 | 20 | 74 | 6F | 20 | 72 | 65 | 62 | ess a key to reb |
| 002000D0 | 6F | 6F | 74 | 2E | 2E | 2E | 00 | 4D | 53 | 57 | 00 | 00 | 00 | 00 | 00 | 00 | oot... MSW |
| 002000E0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 002000F0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00200100 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |

Sector 4096 of 2097152 Offset: 200000

Figure 143. Linux Disk Start Sector

Once we have found the boot sector of the FAT32 volume as well as the start sector, then we can apply the Boot Sector FAT32 Template from WinHex to quickly parse all of the FAT32 boot sector data. One of the fields, seen in the next figure, is labelled Sectors (on large volumes) which is the field needed to calculate the sizes. Because the field is 0 indexed the value of 4182015 needs to be increased to 4182016, then it is divided by two to give the 2091008 that is the number of sectors used on each disk.



Boot Sector FAT32, Base Offset: 200000

| Offset | Title | Value |
|---|---|---|
| 200000 | JMP instruction | EB 58 90 |
| 200003 | OEM | MSWIN4.1 |
| BIOS Parameter Block | | |
| 20000B | Bytes per sector | 512 |
| 20000D | Sectors per cluster | 8 |
| 20000E | Reserved sectors | 32 |
| 200010 | Number of FATs | 2 |
| 200011 | Root entries (unused) | 0 |
| 200013 | Sectors (on small volumes) | 0 |
| 200015 | Media descriptor (hex) | F8 |
| 200016 | Sectors per FAT (small vol.) | 0 |
| 200018 | Sectors per track | 63 |
| 20001A | Heads | 255 |
| 20001C | Hidden sectors | 0 |
| 200020 | Sectors (on large volumes) | 4182015 |

Figure 144. Linux Disk Size

Once the information was gathered, the SoftwareRAIDMount.exe was run with the default stripe size of 128 and all of the other information entered. This stripe size can only be powers of 32 (32, 64, 128, 256, 512, etc.) and as such can be found through trial and error. A quick google search found that as of 2011 the default stripe size on Ubuntu RAID0 was 128 so that should be what we try first. [1]

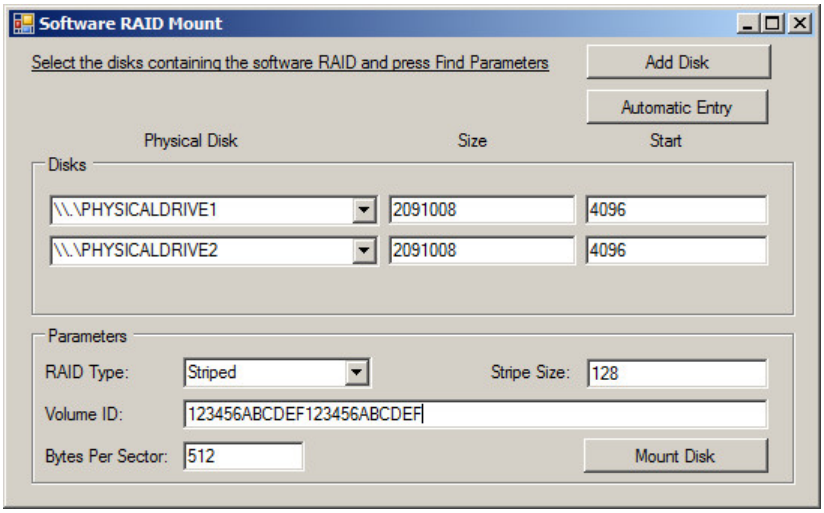



Figure 145. Linux Disk Mount Information

After running the SoftwareRAIDMount.exe, Computer was opened in order to ensure that the drive had mounted.

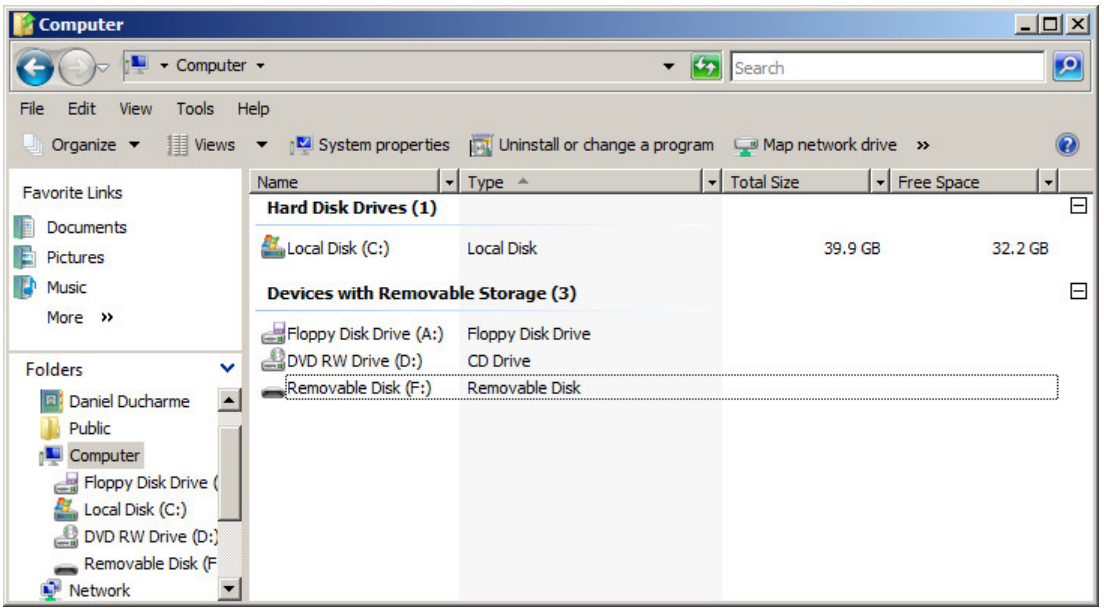



Figure 146. Linux Disk Mounted in Computer

Because there is no volume naming to show that the drive came over correctly the empty drive was also opened to ensure it was properly formatted.

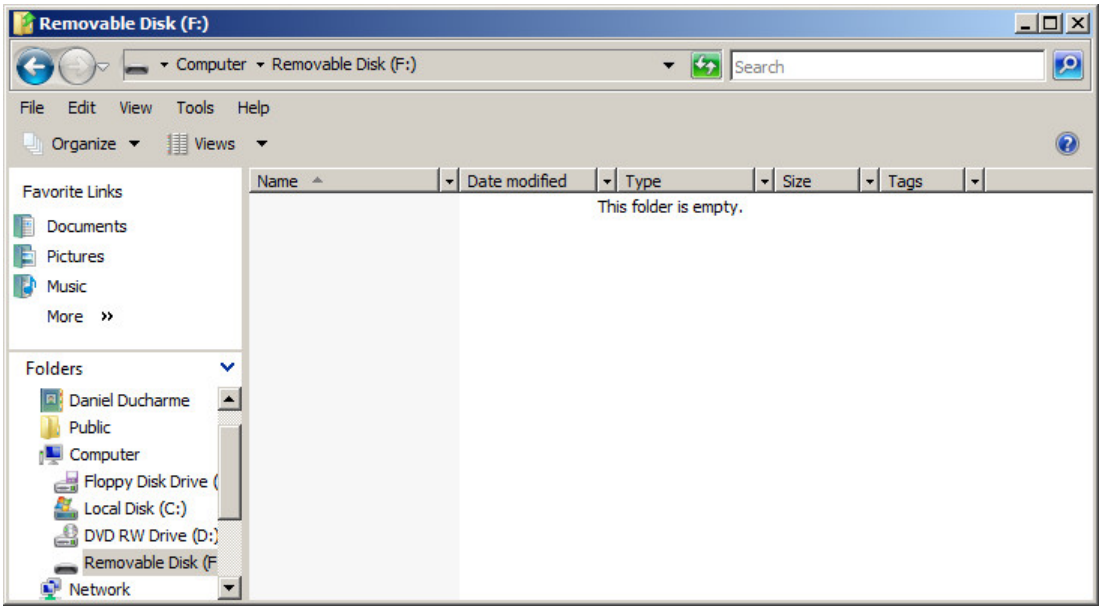



Figure 147. Linux Removable Drive Explorer

## 4.5 Content Testing

After setting up the striped RAID and copying over all of the files, the Windows RAID was taken offline. The RAID was mounted with SoftwareRAID-Mount.exe, then WinHex was started and an initial MD5 hash was taken of the volume.

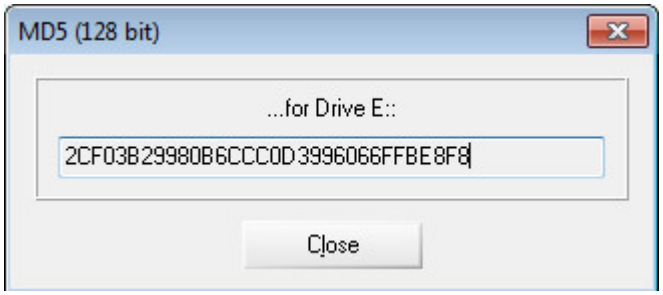



Figure 148. Volume Starting Hash

After the hash was finished, the files that were stored on the driver were copied back to the drive again. This failed as expected with the error shown in the figure below.

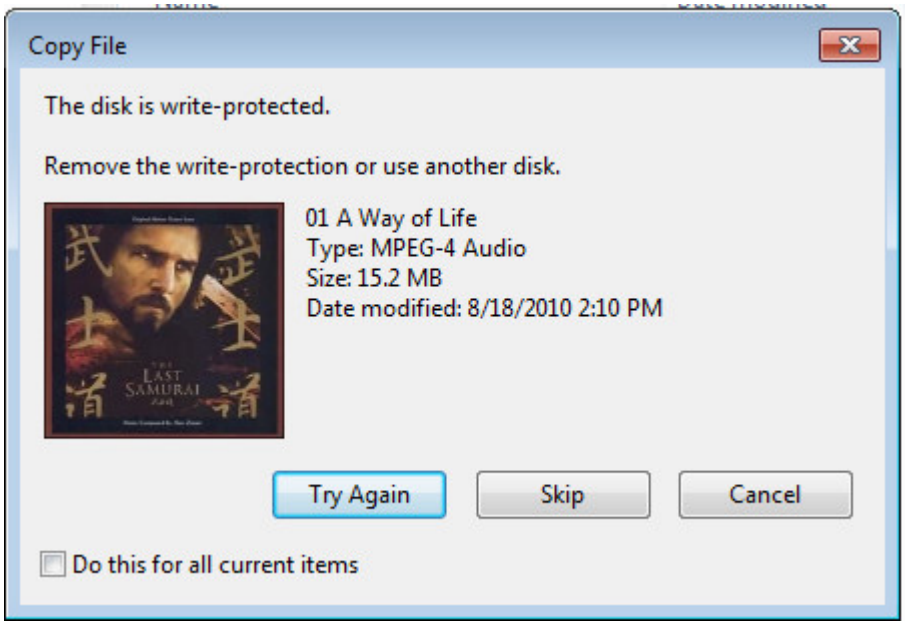



Figure 149. Copy Error

A hash was again computed to make sure that nothing was changed even though the copy failed, and the hash was the same as the initial hash.

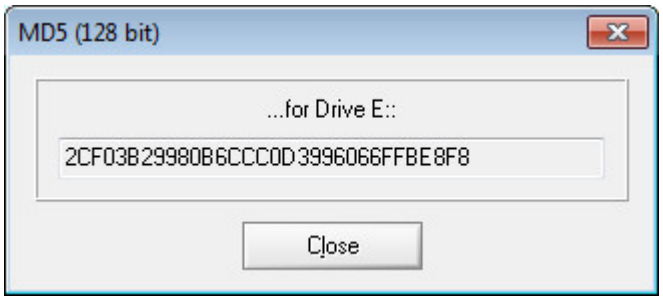



Figure 150. Volume Hash After Copying

Next each of the files was opened in order to ensure that they were being correctly read from the RAID. The first file that was opened was the mpeg which was listened to completely and sounded exactly as expected.

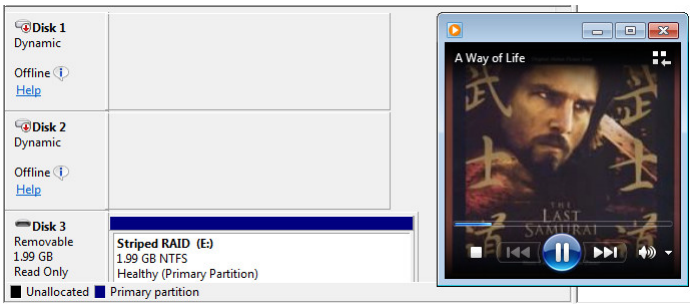



Figure 151. Music Successfully Playing

After the music finished playing, the mp4 movie was opened, and 5 minutes were viewed to ensure that there were no problems. This worked flawlessly again, and both the video and audio played without a hitch.

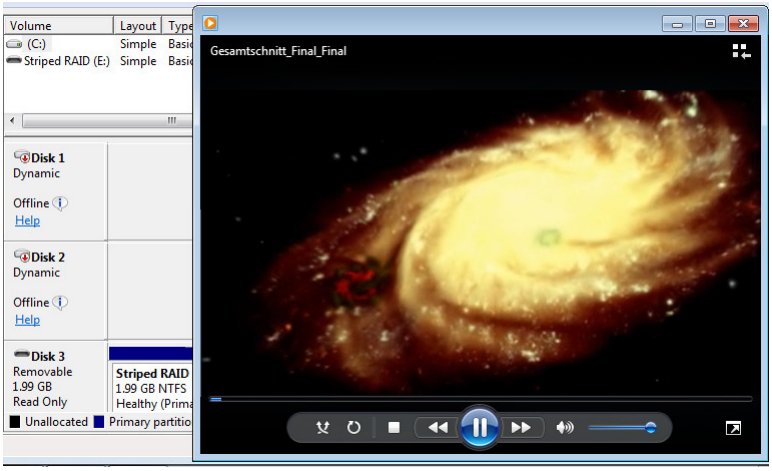



Figure 152. Video Successfully Playing

The next file to be tested was a pdf which again opened with no problem. Scrolling through the file, there were no pages that I viewed which contained any corruption.

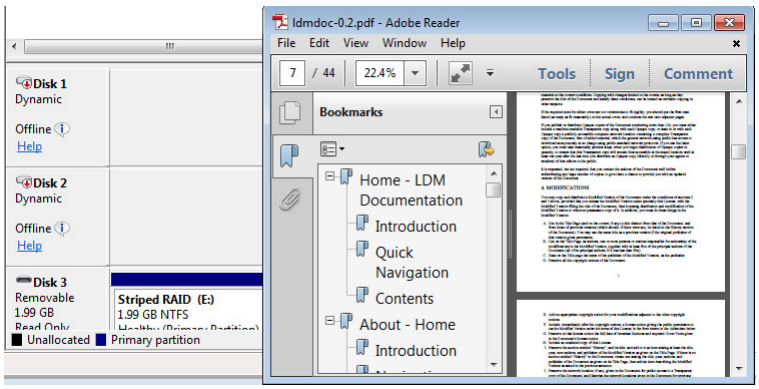



Figure 153. PDF Successfully Opened

Next the jpeg was opened to ensure that there was no problem with images. The image was complete, and there were no problems as the figure below shows.

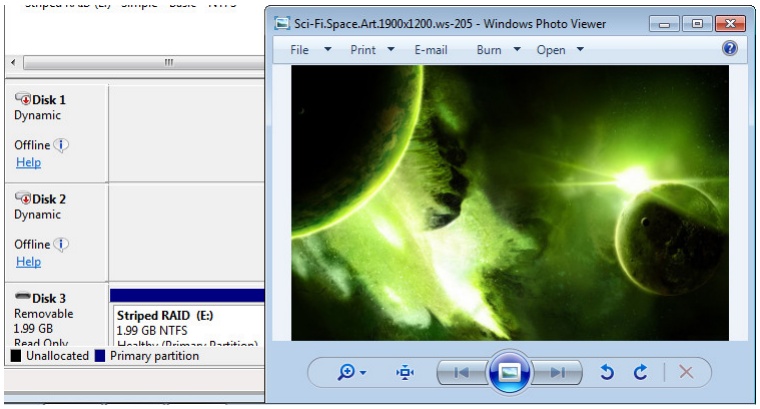



Figure 154. Image Successfully Opened

Finally the text file was opened which worked as expected.

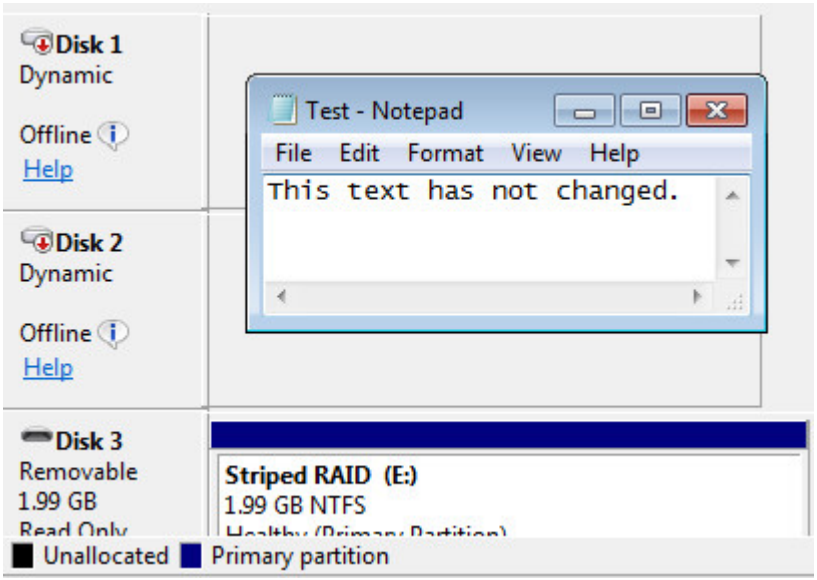



Figure 155. Text Successfully Opened

In order to ensure that no time-stamps had been changed in the opening of the files, the hash was again taken and compared to the initial hash. As expected there is no difference showing that none of the time-stamps were modified.

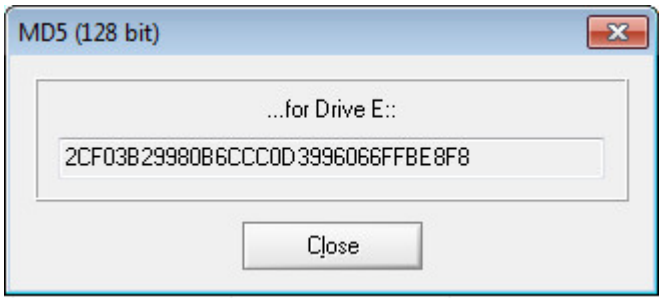



Figure 156. Volume Hash After Files Opened

Finally the text file was modified and then attempted to be saved. Again there was an error message as seen in the next figure.

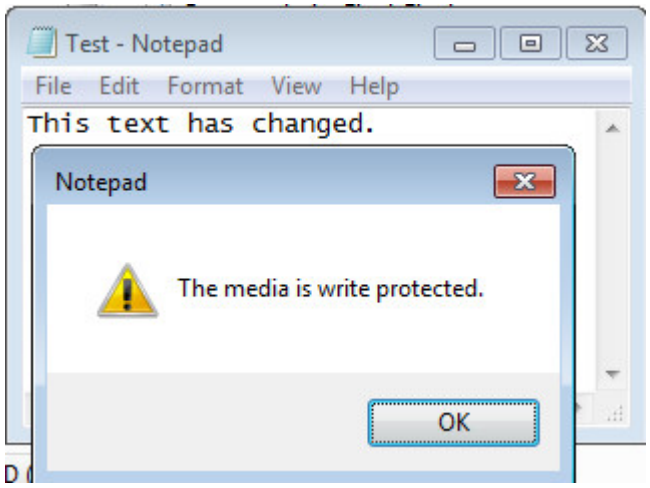



Figure 157. Text Save Fails

Then a final hash of the volume was taken and compared to the initial hash, and as expected there is still no change proving that the driver will not modify the underlying volume even when no write blocker is attached.

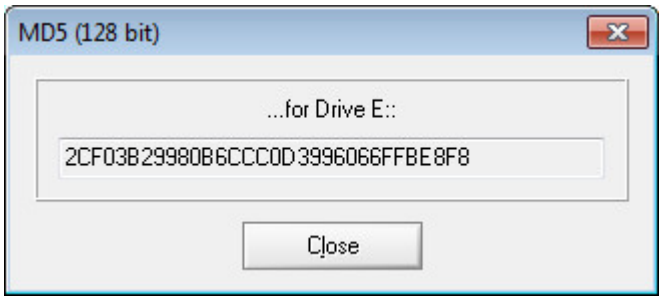



Figure 158. Volume Ending Hash

## List of References

[1] Linux Tutorial. "Configure and install ubuntu on raid 0." [Online; accessed 2-May-2012]. July 2011. [Online]. Available: www.numango.com/5078_install-ubuntu-on-raid.html

# CHAPTER 5

# Conclusion

## 5.1 Speed Testing

The speed testing is one of the most subjective of the tests that were performed on the Software RAID Virtual Disk. Overall the driver functioned as expected. In every situation the drivers worst case speeds exceeded that of Windows worst case speed. Windows is obviously doing something interesting because its maximum speed is faster then the maximum speed of the disk which leads me to believe Windows may use an algorithm to prefetch data in order to increase speed. On average the Software RAID Virtual Disk dropped 1 MB/s with the spanned disks and 3 MB/s for the striped disks. This represents between a 4% decrease in speed to 15% decrease in the worst case. It should be noted, however, that the 15% decrease was only while attached to a write blocker, otherwise the worst case was only a 10% decrease. Overall I feel that these results are acceptable although future work should definitely be done in order to optimize the reading algorithms.

## 5.2 Hash Testing

There is not much to say about these tests, as long as the hashes were done back to back, as explained in the Methodology section, then the two hashes matched every time. Any attempt to bring Windows offline or a reboot of the computer would change the hashes, however, but as this test was more designed to show that the Software RAID Virtual Disk was providing a bit-identical copy of the logical volume, this test was a success.

## 5.3 Operating System Compatibility Testing

There were a few surprises in the results although for the most part it did work as expected. It was not too surprising when it did not work on Windows

Server 2003 as the Storport miniport used by the driver was still new and all of its functionality had not yet been implemented. Thus while it would have been nice, it was not unexpected. What was surprising is that even in test mode it still failed to work on both Windows Vista x64 and Server 2008 x64, while it was able to work on Windows 7 x64. I postulate that in the newer operating system they may have lowered some of the security on drivers due to complaints, or the 64-bit version of Vista and Server 2008 does not have all of the functionality on its implementation of the Storport miniport. In either case it worked fine on the other tested operating systems as expected so the driver supports: Windows Vista x86, Windows Server 2008 x86, and Windows 7 x86 and x64.

## 5.4 Configuration Compatibility Testing

The results were as expected. The driver was able to handle any correct input from the front-end program including the ability to mount corrupted, GPT, and Linux RAIDs as long as the investigator is able to figure out the correct information and the underlying file system has not been corrupted.

## 5.5 Content Testing

There were no surprises with the content testing seeing as how there is no write function implemented. It is great that all of the files played, but seeing as how the hash was identical with the Windows mounted volume, it had already been shown that the information was the same.

## 5.6 Final Conclusion and Future Work

Overall the tests went exactly as expected. While it would have been nice to work on more systems, the driver does rely on several new innovations added to the latest operating systems, and as such, was not expected to run on everything. The speed achieved was acceptable, although not as good as Windows, but that is

again acceptable due to the amount of time using this driver over the competitors solutions will save. Finally the hashes confirmed that the driver was supplying a bit-perfect copy of the volume, and when the volumes contents were tested, the files correctly opened and the hash was preserved.

There are some features that could still be added, however, to make this product far more useful. The first would be to have it correctly parse the LDM on a GPT database instead of relying on hand calculations. Next would be to handle a variety of Linux formats for the same reason. At this point it can handle all of the RAIDs except for RAID-5 which includes parity information. There is a place holder in the driver for that to be mounted, but there was no time to implement it. Finally the addition of the ability for the front end to choose from a variety of RAIDs when there are multiple RAIDs on one disk is needed as it would currently only grabs the last RAID.

# APPENDIX A

## Important Front-End Code

### A.1 Bytes Per Sector

```
public ErrorCodes GetDiskInfo()
{
  try
  {
    WqlObjectQuery wqlQuery = new WqlObjectQuery("SELECT * FROM
        Win32_DiskDrive");
    ManagementObjectSearcher searcher = new ManagementObjectSearcher(
        wqlQuery);
    foreach (ManagementObject disk in searcher.Get())
    {
      if (disk.GetPropertyValue("Name").ToString() == diskPath)
      {
        bytesPerSector = Int32.Parse(disk.GetPropertyValue("
            BytesPerSector").ToString());
        break;
      }
    }
  }
  catch{return ErrorCodes.SRVD_COM_FAILURE;}

  return ErrorCodes.SRVD_OK;
}
```

Code/BytesPerSector.cs

### A.2 Master Boot Record

```
public ErrorCodes GetMBR()
{
```

```
MBR = new byte[bytesPerSector];
byte[] tempPartition = new byte[16];
ErrorCodes result;

diskPartitions = new List<Partition>();
try
{
  bool success = setHandle();
  if (!success)
  {
    return ErrorCodes.SRVD_BAD_DISK;
  }

  diskHandle.Position = 0;
  diskHandle.Read(MBR, 0, bytesPerSector);
  for (int j = 0; j < 4; j++)
  {
    for (int i = 0; i < 16; i++)
    {
      tempPartition[i] = MBR[i + 446 + (j*16)];
    }
    GCHandle pinnedPart = GCHandle.Alloc(tempPartition,
        GCHandleType.Pinned);
    PARTITION newPartition = (PARTITION)Marshal.PtrToStructure(
      pinnedPart.AddrOfPinnedObject(),
      typeof(PARTITION));
    diskPartitions.Add(new Partition(newPartition));
    pinnedPart.Free();
  }

  BootRecordSignature = new byte[2];
  for (int i = 0; i < 2; i++)
```

```
        BootRecordSignature[i] = MBR[i + 510];
      if (BootRecordSignature[0] == 0x55 && BootRecordSignature[1] == 0
          xaa)
      {
        result = ErrorCodes.SRVD_OK;
      }
      else
        result = ErrorCodes.SRVD_NO_DISKS;

    }
    catch { return ErrorCodes.SRVD_COM_FAILURE; }
    finally
    {
      diskHandle.SafeFileHandle.Close();
      diskHandle.Close();
    }

    for (int i = 0; i < 4; i++)
    {
      if (diskPartitions[i].GetDynamic())
      {
        DynamicPresent = true;
      }
    }
    return result;
}
```

Code/MBR.cs

## A.3 Logical Disk Manager

```
public ErrorCodes GetLDM()
{
  int LDMSize = 1048576;
```

```
Int64 Size = 0;
Int64 TotalSize = 0;
uint output = 0;
byte[] privheader = new byte[512];

try
{
  bool success = setHandle();
  if (!success)
  {
    return ErrorCodes.SRVD_BAD_DISK;
  }

  bool result = DeviceIoControl(diskHandle.SafeFileHandle,
          FSConstants.IOCTL_DISK_GET_LENGTH_INFO,
          IntPtr.Zero, 0,
          out TotalSize, (uint)(Marshal.SizeOf(TotalSize)),
          out output, IntPtr.Zero);

  if (!result)
    return ErrorCodes.SRVD_COM_FAILURE;

  Size = TotalSize;
  Size -= 512;
  diskHandle.Seek(Size, SeekOrigin.Begin);

  Int64 phead = 512;

  Size += 512;
  Size -= LDMSize;

  diskHandle.Seek(Size, SeekOrigin.Begin);
```

```
byte[] LDM = new byte[LDMSize];
diskHandle.Read(LDM, 0, LDMSize);
Array.ConstrainedCopy(LDM, LDMSize-512, privheader, 0, (int)phead
    );
privateHeader = new PRIVHEAD(privheader);
//Find start of TOCBLOCK
int DatabaseStart = 0;
char test = 'T';
while(LDM[DatabaseStart] != test)
{
  DatabaseStart++;
  if (DatabaseStart >= LDMSize)
  {
    return ErrorCodes.SRVD_LDM_FAIL;
  }
}
byte[] tocBlock = new byte[512];
for(int i = 0; i < 512; i++)
  tocBlock[i] = LDM[DatabaseStart+i];

TOCblock = new TOCBLOCK(tocBlock);
ulong ConfigStart = TOCblock.GetConfigStart() * 512;

byte[] Vmdb = new byte[512];
for(int i = 0; i < 512; i++)
  Vmdb[i] = LDM[ConfigStart + (ulong)i];

MyVMDB = new VMDB(Vmdb);
result = MyVMDB.GetFail();
if (result)
  return ErrorCodes.SRVD_VMDB_FAILURE;
```

```
    String PrivHeadGUID = (privateHeader.PrivateHeader.DiskGroupGUID.
        Replace("-","" )).ToLower();
    String VMDBGUID = (MyVMDB.myVMDB.DiskGroupGUID.Replace("-", "")).
        ToLower();

    result = String.Equals(PrivHeadGUID, VMDBGUID, StringComparison.
        OrdinalIgnoreCase);
    if (!result)
      return ErrorCodes.SRVD_VMDB_FAILURE;

   MyVBLK = new VBLK[MyVMDB.VBLKCount() + 1];

    ulong k = 4;
    for(int i = 4; k < MyVMDB.VBLKCount()+4; i++)
    {
      byte[] vblk = new byte[128];
      for(int j = 0; j < 128; j++)
        vblk[j] = LDM[ConfigStart + (ulong)(i*128) + (ulong)j];
      MyVBLK[k - 4] = new VBLK();
      if (!MyVBLK[k-4].Parse(vblk))
        return ErrorCodes.SRVD_VBLK_FAILURE;

      if(MyVBLK[k-4].GetEmpty())
      {
        MyVBLK[k-4].Initialize();
        continue;
      }
      k++;
    }
  }
  catch { return ErrorCodes.SRVD_BAD_DISK; }
  finally
```

```
  {
    if (!diskHandle.SafeFileHandle.IsClosed)
      diskHandle.Close();
  }

  return ErrorCodes.SRVD_OK;
}
```

Code/LDM.cs

### A.3.1 Private Header

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct PrivHead
{
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
  public String signature;
  [MarshalAs(UnmanagedType.I4)]
  public int seq;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
  public String majversion;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
  public String minversion;
  [MarshalAs(UnmanagedType.U8)]
  public ulong timestamp;
  [MarshalAs(UnmanagedType.U8)]
  public ulong number;
  [MarshalAs(UnmanagedType.U8)]
  public ulong size1;
  [MarshalAs(UnmanagedType.U8)]
  public ulong size2;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
  public String DiskGUID;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
```

```
public String HostGUID;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
public String DiskGroupGUID;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 32)]
public String DiskGroupName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 11)]
public String trash;
[MarshalAs(UnmanagedType.U8)]
public ulong LogicalDiskStart;
[MarshalAs(UnmanagedType.U8)]
public ulong LogicalDiskSize;
[MarshalAs(UnmanagedType.U8)]
public ulong ConfigurationStart;
[MarshalAs(UnmanagedType.U8)]
public ulong ConfigurationSize;
[MarshalAs(UnmanagedType.U8)]
public ulong NumberofTOCs;
[MarshalAs(UnmanagedType.U8)]
public ulong TOCSize;
[MarshalAs(UnmanagedType.U4)]
public uint NumberOfConfigs;
[MarshalAs(UnmanagedType.U4)]
public uint NumberOfLogs;
[MarshalAs(UnmanagedType.U8)]
public ulong SizeOfConfig;
[MarshalAs(UnmanagedType.U8)]
public ulong SizeOfLog;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4)]
public String DiskSignature;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 16)]
public String DiskSetGUID;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 16)]
```

```
public String DiskSetGUID2;

//public PrivHead(int num = 0)
//{
//  signature = new char[8];
//  seq = 0;
//  majversion = new char[2];
//  minversion = new char[2];
//  timestamp = 0L;
//  number = 0L;
//  size1 = 0L;
//  size2 = 0L;
//  DiskGUID = new char[64];
//  HostGUID = new char[64];
//  DiskGroupGUID = new char[64];
//  DiskGroupName = new char[32];
//  trash = new char[11];
//  LogicalDiskStart = 0L;
//  LogicalDiskSize = 0L;
//  ConfigurationStart = 0L;
//  ConfigurationSize = 0L;
//  NumberofTOCs = 0L;
//  TOCSize = 0L;
//  NumberOfConfigs = 0;
//  NumberOfLogs = 0;
//  SizeOfConfig = 0L;
//  SizeOfLog = 0L;
//  DiskSignature = new char[4];
//  DiskSetGUID = new char[16];
//  DiskSetGUID2 = new char[16];
//}
```

```
};
```

Code/PrivHead.cs

### A.3.2 Table Of Contents Block

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct TOCBlock
{
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
  public String Signature;
  [MarshalAs(UnmanagedType.I4)]
  public int Sequence;
  [MarshalAs(UnmanagedType.I4)]
  public int Zero1;
  [MarshalAs(UnmanagedType.I4)]
  public int Sequence1;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 16)]
  public String Zero2;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
  public String BitmapName0;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
  public String BitmapFlags00;
  [MarshalAs(UnmanagedType.U8)]
  public ulong BitmapStart0;
  [MarshalAs(UnmanagedType.U8)]
  public ulong BitmapSize0;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
  public String BitmapFlags01;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
  public String BitmapName1;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
  public String BitmapFlags10;
  [MarshalAs(UnmanagedType.U8)]
```

```
    public ulong BitmapStart1;
    [MarshalAs(UnmanagedType.U8)]
    public ulong BitmapSize1;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
    public String BitmapFlags11;


    //public TOCBlock(int num = 0)
    //{
    //  Signature = new char[8];
    //  Sequence = 0;
    //  Zero1 = 0;
    //  Sequence1 = 0;
    //  Zero2 = new char[16];
    //  BitmapName0 = new char[8];
    //  BitmapFlags00 = new char[2];
    //  BitmapStart0 = 0L;
    //  BitmapSize0 = 0L;
    //  BitmapFlags01 = new char[8];
    //  BitmapName1 = new char[8];
    //  BitmapFlags10 = new char[2];
    //  BitmapStart1 = 0L;
    //  BitmapSize1 = 0L;
    //  BitmapFlags11 = new char[8];
    //}
};
```

Code/TOCBlock.cs

### A.3.3 Volume Master DataBase

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct vmdb
{
```

```
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 4)]
public String Signature;
[MarshalAs(UnmanagedType.I4)]
public int Sequence;
[MarshalAs(UnmanagedType.I4)]
public int Size;
[MarshalAs(UnmanagedType.I4)]
public int Offset;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
public String UpdateStatus;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
public String MajorVersion;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2)]
public String MinorVersion;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 31)]
public String DiskGroupName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
public String DiskGroupGUID;
[MarshalAs(UnmanagedType.U8)]
public ulong CommittedSequence;
[MarshalAs(UnmanagedType.U8)]
public ulong PendingSequence;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofCommittedVolumes;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofCommittedComponents;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofCommittedPartitions;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofCommittedDisks;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 12)]
public String unused;
```

```
[MarshalAs(UnmanagedType.U4)]
public uint NumberofPendingVolumes;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofPendingComponents;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofPendingPartitions;
[MarshalAs(UnmanagedType.U4)]
public uint NumberofPendingDisks;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 12)]
public String unused1;
[MarshalAs(UnmanagedType.I4)]
public int lastAccessedTime;

//public vmdb(int none = 0)
//{
//  Signature = new char[4];
//  Sequence = 0;
//  Size = 0;
//  Offset = 0;
//  UpdateStatus = new char[2];
//  MajorVersion = new char[2];
//  MinorVersion = new char[2];
//  DiskGroupName = new char[31];
//  DiskGroupGUID = new char[64];
//  CommittedSequence = 0L;
//  PendingSequence = 0L;
//  NumberofCommittedVolumes = 0;
//  NumberofCommittedComponents = 0;
//  NumberofCommittedPartitions = 0;
//  NumberofCommittedDisks = 0;
//  unused = new char[12];
//  NumberofPendingVolumes = 0;
```

```
    //  NumberofPendingComponents = 0;
    //  NumberofPendingPartitions = 0;
    //  NumberofPendingDisks = 0;
    //  unused1 = new char[12];
    //  lastAccessedTime = 0;
    //}
};
```

Code/VMDB.cs

### A.3.4 Volume BLocK

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKHeader
{
    [MarshalAs(UnmanagedType.U4)]
    public uint Signature;
    [MarshalAs(UnmanagedType.U4)]
    public uint Sequence;
    [MarshalAs(UnmanagedType.U4)]
    public uint GroupNumber;
    [MarshalAs(UnmanagedType.U2)]
    public ushort RecordNumber;
    [MarshalAs(UnmanagedType.U2)]
    public ushort NumberofRecords;
    [MarshalAs(UnmanagedType.U2)]
    public ushort UpdateStatus;
    [MarshalAs(UnmanagedType.U2)]
    public ushort RecordType;
    [MarshalAs(UnmanagedType.I4)]
    public int DataLength;

    //public VBLKHeader(int none = 0)
    //{
```

```
  //  Signature = new char[4];
  //  Sequence = 0;
  //  GroupNumber = 0;
  //  RecordNumber = 0;
  //  NumberofRecords = 0;
  //  UpdateStatus = new char[2];
  //  RecordType = new char[2];
  //  DataLength = 0;
  //}
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKComponent{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public String VolumeState;
  public ushort VolumeState_length;
  public byte ComponentType;
  public ulong NumberofChildren;
  public ulong LogCommitId;
  public ulong ParentID;
  public ulong StripeSize;
  public ulong NumberofColumns;
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKPartition{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public ulong LogCommitId;
```

```
  public ulong Start;
  public ulong VolumeOffset;
  public ulong Size;
  public ulong ParentsObjectID;
  public ulong DisksObjectID;
  public ulong ComponentPartIndex;
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKDisk1{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public String DiskID;
  public ushort DiskID_length;
  public String AlternateName;
  public ushort AlternateName_length;
  public ulong LogCommitId;
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKDisk2{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public String DiskID1;
  public String DiskID2;
  public ushort ID;
  public ulong LogCommitId;

  public VBLKDisk2(int none = 0)
  {
```

```
    ObjectID = 0L;
    Name = "";
    Name_length = 0;
    DiskID1 = "";
    DiskID2 = "";
    ID = 0;
    LogCommitId = 0L;
  }
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKDiskGroup1{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public String DiskGroupID;
  public ushort DiskGroupID_length;
  public ulong LogCommitId;
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKDiskGroup2{
  public ulong ObjectID;
  public String Name;
  public ushort Name_length;
  public byte[] DiskGroupID;
  public byte[] DiskSetID;
  public ulong LogCommitId;

  public VBLKDiskGroup2(int none = 0)
  {
    ObjectID = 0L;
```

```
        Name = "";
        Name_length = 0;
        DiskGroupID = new byte[16];
        DiskSetID = new byte[16];
        LogCommitId = 0L;
    }
};

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct VBLKVolume
{
    public ulong ObjectID;
    public String Name;
    public ushort Name_length;
    public String VolumeType;
    public ushort VolumeType_length;
    public byte[] VolumeState;
    public char VolumeType1;
    public char VolumeNumber;
    public char Flag;
    public ulong NumberofChildren;
    public ulong LogCommitId;
    public byte[] Id;
    public ulong Size;
    public char PartitionType;
    public byte[] VolumeID;
    public String Id1;
    public ushort Id1_length;
    public String Id2;
    public ushort Id2_length;
    public ulong OptSize;
    public String DriveHint;
```

```
  public ushort DriveHint_length;

  public VBLKVolume(int none = 0)
  {
    ObjectID = 0L;
    Name = "";
    Name_length = 0;
    VolumeType = "";
    VolumeType_length = 0;
    VolumeState = new byte[14];
    VolumeType1 =' ';
    VolumeNumber = ' ';
    Flag = ' ';
    NumberofChildren = 0L;
    LogCommitId = 0L;
    Id = new byte[8];
    Size = 0L;
    PartitionType = ' ';
    VolumeID = new byte[16];
    Id1 = "";
    Id1_length = 0;
    Id2 = "";
    Id2_length = 0;
    OptSize = 0L;
    DriveHint = "";
    DriveHint_length = 0;
  }
};

public class VBLK
{
  public VBLKHeader myVBLKHeader;
```

```
public VBLKComponent myVBLKComponent;
public VBLKPartition myVBLKPartition;
public VBLKDisk1 myVBLKDisk1;
public VBLKDisk2 myVBLKDisk2;
public VBLKDiskGroup1 myVBLKDiskGroup1;
public VBLKDiskGroup2 myVBLKDiskGroup2;
public VBLKVolume myVBLKVolume;
public bool fail;
public bool empty;
public bool Volume;
public bool Component;
public bool Partition;
public bool Disk1;
public bool Disk2;
public bool DiskGroup1;
public bool DiskGroup2;
public int Children;
public int Children_length;
public int Parent;

public VBLK()
{
  Initialize();
}

public void Initialize()
{
  fail = false;
  empty = false;
  Volume = false;
  Component = false;
  Partition = false;
```

```
    Disk1 = false;
    Disk2 = false;
    DiskGroup1 = false;
    DiskGroup2 = false;
    Children_length = 0;
}

public bool Parse(byte[] rawarray)
{
    int i = 0;
    short length = 0;
    bool even = false;
    byte Length;
    Initialize();

    GCHandle pinnedHeader = GCHandle.Alloc(rawarray, GCHandleType.
        Pinned);
    VBLKHeader tempHeader = (VBLKHeader)Marshal.PtrToStructure(
                pinnedHeader.AddrOfPinnedObject(),
                typeof(VBLKHeader));
    myVBLKHeader = tempHeader;
    pinnedHeader.Free();
    int count = Marshal.SizeOf(myVBLKHeader);

    if(BitConverter.IsLittleEndian)
    {
        myVBLKHeader.Sequence = (uint)IPAddress.HostToNetworkOrder((int
            )myVBLKHeader.Sequence);
        myVBLKHeader.GroupNumber = (uint)IPAddress.HostToNetworkOrder((
            int)myVBLKHeader.GroupNumber);
        myVBLKHeader.RecordNumber = (ushort)IPAddress.
            HostToNetworkOrder((short)myVBLKHeader.RecordNumber);
```

```
    myVBLKHeader.NumberofRecords = (ushort)IPAddress.
        HostToNetworkOrder((short)myVBLKHeader.NumberofRecords);
    myVBLKHeader.DataLength = IPAddress.HostToNetworkOrder(
        myVBLKHeader.DataLength);
}


if(myVBLKHeader.UpdateStatus%2 != 0)
    fail = true;

//RecordType
{
    byte[] recordType = BitConverter.GetBytes(myVBLKHeader.
        RecordType);
    if (recordType.Length < 2)
      return false;

    if (recordType[1] == 0x32)
    {
      Component = true;
      myVBLKComponent = new VBLKComponent();
    }
    else if (recordType[1] == 0x33)
    {
      Partition = true;
      myVBLKPartition = new VBLKPartition();
    }
    else if (recordType[1] == 0x34)
    {
      Disk1 = true;
      myVBLKDisk1 = new VBLKDisk1();
    }
```

```
    else if (recordType[1] == 0x35)
    {
      DiskGroup1 = true;
      myVBLKDiskGroup1 = new VBLKDiskGroup1();
    }
    else if (recordType[1] == 0x44)
    {
      Disk2 = true;
      myVBLKDisk2 = new VBLKDisk2(0);
    }
    else if (recordType[1] == 0x45)
    {
      DiskGroup2 = true;
      myVBLKDiskGroup2 = new VBLKDiskGroup2(0);
    }
    else if (recordType[1] == 0x51)
    {
      Volume = true;
      myVBLKVolume = new VBLKVolume(0);
    }
  }

  if(Component)
  {
    //ObjectID
    {

      Length = rawarray[count];
      count++;
      length = (short)Length;
      even = true;
      if(length%2 != 0) even = false;
```

```
  byte[] ObjectID;
  if(even)
  {
    ObjectID = new byte[length];
    for(i = 0; i < length; i++)
      ObjectID[i] = rawarray[count + i];
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKComponent.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempName = new char[length];
  myVBLKComponent.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempName[i] = (char)rawarray[count + i];
  myVBLKComponent.Name = new String(tempName);
  count += length;
}

//VolumeState
```

```
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempVolState = new char[length];
  myVBLKComponent.VolumeState_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempVolState[i] = (char)rawarray[count + i];
  myVBLKComponent.VolumeState = new String(tempVolState);
  count += length;
}

myVBLKComponent.ComponentType = rawarray[count];
count += 1;

//Zero
count += 4;

//NumberofChildren
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] NumberofChildren;
  if(even)
  {
    NumberofChildren = new byte[length];
    for(i = 0; i < length; i++)
      NumberofChildren[i] = rawarray[count + i];
  }
```

```
    else
    {
      NumberofChildren = new byte[length + 1];
      NumberofChildren[0] = 0x00;
      for(i = 0; i < length; i++)
        NumberofChildren[i+1] = rawarray[count + i];
    }
    myVBLKComponent.NumberofChildren = convertValue(
        NumberofChildren, length);
    count += length;
  }

  myVBLKComponent.LogCommitId = (ulong)BitConverter.ToInt64(
      rawarray, count);
  count += 8;

  //Zero
  count += 8;

  //ParentID
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] ParentID;
    if(even)
    {
      ParentID = new byte[length];
      for(i = 0; i < length; i++)
        ParentID[i] = rawarray[count + i];
```

```
  }
  else
  {
    ParentID = new byte[length + 1];
    ParentID[0] = 0x00;
    for(i = 0; i < length; i++)
      ParentID[i+1] = rawarray[count + i];
  }
  myVBLKComponent.ParentID = convertValue(ParentID, length);
  count += length;
}

//Zero
count += 1;

byte[] recordType = BitConverter.GetBytes(myVBLKHeader.
    RecordType);
if (recordType.Length < 2)
  return false;

if (recordType[0] == 0x10)
{
  //StripeSize
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] StripeSize;
    if(even)
    {
```

```
      StripeSize = new byte[length];
      for(i = 0; i < length; i++)
        StripeSize[i] = rawarray[count + i];
    }
    else
    {
      StripeSize = new byte[length + 1];
      StripeSize[0] = 0x00;
      for(i = 0; i < length; i++)
        StripeSize[i+1] = rawarray[count + i];
    }
    myVBLKComponent.StripeSize = convertValue(StripeSize,
        length);
    count += length;
  }

  //NumberofColumns
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] NumberofColumns;
    if(even)
    {
      NumberofColumns = new byte[length];
      for(i = 0; i < length; i++)
        NumberofColumns[i] = rawarray[count + i];
    }
    else
    {
```

```
            NumberofColumns = new byte[length + 1];
            NumberofColumns[0] = 0x00;
            for(i = 0; i < length; i++)
              NumberofColumns[i+1] = rawarray[count + i];
          }
          myVBLKComponent.NumberofColumns = convertValue(
              NumberofColumns, length);
          count += length;
        }
      }

      if (BitConverter.IsLittleEndian)
      {
        myVBLKComponent.LogCommitId = (ulong)IPAddress.
            HostToNetworkOrder((long)myVBLKComponent.LogCommitId);
      }
    }
    else if(Partition)
    {
      //ObjectID
      {
        Length = rawarray[count];
        count++;
        length = (short)Length;
        even = true;
        if(length%2 != 0) even = false;
        byte[] ObjectID;
        if(even)
        {
          ObjectID = new byte[length];
          for(i = 0; i < length; i++)
            ObjectID[i] = rawarray[count + i];
```

```
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKPartition.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKPartition.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKPartition.Name = new String(tempStr);
  count += length;
}

//Zero
count += 4;

byte[] temp = new byte[8];
Array.ConstrainedCopy(rawarray, count, temp, 0, 8);
myVBLKPartition.LogCommitId = (ulong)BitConverter.ToInt64(temp,
    0);
```

```
count += 8;
Array.ConstrainedCopy(rawarray, count, temp, 0, 8);
myVBLKPartition.Start = (ulong)BitConverter.ToInt64(temp, 0);
count += 8;
Array.ConstrainedCopy(rawarray, count, temp, 0, 8);
myVBLKPartition.VolumeOffset = (ulong)BitConverter.ToInt64(temp
    , 0);
count += 8;

//Size
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] size;
  if(even)
  {
    size = new byte[length];
    for(i = 0; i < length; i++)
      size[i] = rawarray[count + i];
  }
  else
  {
    size = new byte[length + 1];
    size[0] = 0x00;
    for(i = 0; i < length; i++)
      size[i+1] = rawarray[count + i];
  }
  myVBLKPartition.Size = convertValue(size, length);
  count += length;
```

```
}

//ParentsObjectID
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] ParentsObjectID;
  if(even)
  {
    ParentsObjectID = new byte[length];
    for(i = 0; i < length; i++)
      ParentsObjectID[i] = rawarray[count + i];
  }
  else
  {
    ParentsObjectID = new byte[length + 1];
    ParentsObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ParentsObjectID[i+1] = rawarray[count + i];
  }
  myVBLKPartition.ParentsObjectID = convertValue(
     ParentsObjectID, length);
  count += length;
}

//DisksObjectID
{
  Length = rawarray[count];
  count++;
```

```
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] DisksObjectID;
  if(even)
  {
    DisksObjectID = new byte[length];
    for(i = 0; i < length; i++)
      DisksObjectID[i] = rawarray[count + i];
  }
  else
  {
    DisksObjectID = new byte[length + 1];
    DisksObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      DisksObjectID[i+1] = rawarray[count + i];
  }
  myVBLKPartition.DisksObjectID = convertValue(DisksObjectID,
     length);
  count += length;
}

//ComponentPartIndex
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] ComponentPartIndex;
  if(even)
  {
```

```
        ComponentPartIndex = new byte[length];
        for(i = 0; i < length; i++)
          ComponentPartIndex[i] = rawarray[count + i];
      }
      else
      {
        ComponentPartIndex = new byte[length + 1];
        ComponentPartIndex[0] = 0x00;
        for(i = 0; i < length; i++)
          ComponentPartIndex[i+1] = rawarray[count + i];
      }
      myVBLKPartition.ComponentPartIndex = convertValue(
          ComponentPartIndex, length);
      count += length;
    }

    if (BitConverter.IsLittleEndian)
    {
      myVBLKPartition.LogCommitId = (ulong)IPAddress.
          HostToNetworkOrder((long)myVBLKPartition.LogCommitId);
      myVBLKPartition.Start = (ulong)IPAddress.HostToNetworkOrder((
          long)myVBLKPartition.Start);
      myVBLKPartition.VolumeOffset = (ulong)IPAddress.
          HostToNetworkOrder((long)myVBLKPartition.VolumeOffset);
    }
  }
  else if(Disk1)
  {
    //ObjectID
    {
      Length = rawarray[count];
      count++;
```

```
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] ObjectID;
  if(even)
  {
    ObjectID = new byte[length];
    for(i = 0; i < length; i++)
      ObjectID[i] = rawarray[count + i];
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKDisk1.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDisk1.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDisk1.Name = new String(tempStr);
  count += length;
```

```
}

//DiskID
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDisk1.DiskID_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDisk1.DiskID = new String(tempStr);
  count += length;
}

//AlternateName
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDisk1.AlternateName_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDisk1.AlternateName = new String(tempStr);
  count += length;
}

//Zero
count += 4;

myVBLKDisk1.LogCommitId = (ulong)(rawarray[count]);
```

```
    count += 8;

    if (BitConverter.IsLittleEndian)
    {
      myVBLKDisk1.LogCommitId = (ulong)IPAddress.HostToNetworkOrder
          ((long)myVBLKDisk1.LogCommitId);
    }
  }
  else if(Disk2)
  {
    //ObjectID
    {
      Length = rawarray[count];
      count++;
      length = (short)Length;
      even = true;
      if(length%2 != 0) even = false;
      byte[] ObjectID;
      if(even)
      {
        ObjectID = new byte[length];
        for(i = 0; i < length; i++)
          ObjectID[i] = rawarray[count + i];
      }
      else
      {
        ObjectID = new byte[length + 1];
        ObjectID[0] = 0x00;
        for(i = 0; i < length; i++)
          ObjectID[i+1] = rawarray[count + i];
      }
      myVBLKDisk2.ObjectID = convertValue(ObjectID, length);
```

```
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDisk2.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDisk2.Name = new String(tempStr);
  count += length;
}

//DiskID1
{
  length = 16;
  char[] tempStr = new char[length];
  for (i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDisk2.DiskID1 = new String(tempStr);
}
count += 16;

//DiskID2
{
  length = 16;
  char[] tempStr = new char[length];
  for (i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
```

```
      myVBLKDisk2.DiskID2 = new String(tempStr);
    }
    count += 16;

    //Zero
    count += 3;

    myVBLKDisk2.ID = (ushort)(rawarray[count]);
    count += 2;

    myVBLKDisk2.LogCommitId = (ulong)(rawarray[count]);
    count += 8;

    if(BitConverter.IsLittleEndian)
    {
      myVBLKDisk2.ID = (ushort)IPAddress.HostToNetworkOrder(
          myVBLKDisk2.ID);
      myVBLKDisk2.LogCommitId = (ulong)((long)myVBLKDisk2.
          LogCommitId);
    }
  }
  else if(DiskGroup1)
  {
    //ObjectID
    {
      Length = rawarray[count];
      count++;
      length = (short)Length;
      even = true;
      if(length%2 != 0) even = false;
      byte[] ObjectID;
      if(even)
```

```
  {
    ObjectID = new byte[length];
    for(i = 0; i < length; i++)
      ObjectID[i] = rawarray[count + i];
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKDiskGroup1.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDiskGroup1.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDiskGroup1.Name = new String(tempStr);
  count += length;
}

//DiskGroupID
{
  Length = rawarray[count];
```

```
      count++;
      length = (short)Length;
      char[] tempStr = new char[length];
      myVBLKDiskGroup1.DiskGroupID_length = (ushort)length;
      for(i = 0; i < length; i++)
        tempStr[i] = (char)rawarray[count + i];
      myVBLKDiskGroup1.DiskGroupID = new String(tempStr);
      count += length;
    }

    //Zero
    count += 4;

    myVBLKDiskGroup1.LogCommitId = (ulong)(rawarray[count]);
    count += 8;

    if (BitConverter.IsLittleEndian)
    {
      myVBLKDiskGroup1.LogCommitId = (ulong)IPAddress.
          HostToNetworkOrder((long)myVBLKDiskGroup1.LogCommitId);
    }
  }
  else if(DiskGroup2)
  {
    //ObjectID
    {
      Length = rawarray[count];
      count++;
      length = (short)Length;
      even = true;
      if(length%2 != 0) even = false;
      byte[] ObjectID;
```

```
  if(even)
  {
    ObjectID = new byte[length];
    for(i = 0; i < length; i++)
      ObjectID[i] = rawarray[count + i];
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKDiskGroup2.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKDiskGroup2.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKDiskGroup2.Name = new String(tempStr);
  count += length;
}

Array.ConstrainedCopy(rawarray, count, myVBLKDiskGroup2.
    DiskGroupID, 0, 16);
```

```
    count += 16;
    Array.ConstrainedCopy(rawarray, count, myVBLKDiskGroup2.
        DiskGroupID, 0, 16);
    count += 16;

    //Zero
    count += 4;

    myVBLKDiskGroup2.LogCommitId = (ulong)(rawarray[count]);


    if(BitConverter.IsLittleEndian)
    {
      myVBLKDiskGroup2.LogCommitId = (ulong)IPAddress.
          HostToNetworkOrder((long)myVBLKDiskGroup2.LogCommitId);
    }
  }
  else if(Volume)
  {
    //ObjectID
    {
      Length = rawarray[count];
      count++;
      length = (short)Length;
      even = true;
      if(length%2 != 0) even = false;
      byte[] ObjectID;
      if(even)
      {
        ObjectID = new byte[length];
        for(i = 0; i < length; i++)
          ObjectID[i] = rawarray[count + i];
```

```
  }
  else
  {
    ObjectID = new byte[length + 1];
    ObjectID[0] = 0x00;
    for(i = 0; i < length; i++)
      ObjectID[i+1] = rawarray[count + i];
  }
  myVBLKVolume.ObjectID = convertValue(ObjectID, length);
  count += length;
}

//Name
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  myVBLKVolume.Name_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKVolume.Name = new String(tempStr);
  count += length;
}

//VolumeType
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr = new char[length];
  tempStr = new char[length];
```

```
    myVBLKVolume.VolumeType_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.VolumeType = new String(tempStr);
    count += length;
}

//Zero
count += 1;

Array.ConstrainedCopy(rawarray, count, myVBLKVolume.VolumeState
    , 0, 14);
count += 14;

myVBLKVolume.VolumeType1 = (char)(rawarray[count]);
count += 1;

//One
count += 1;

myVBLKVolume.VolumeNumber = (char)(rawarray[count]);
count += 1;

//Zero
count += 3;

myVBLKVolume.Flag = (char)(rawarray[count]);
count += 1;

//NumberofChildren
{
    Length = rawarray[count];
```

```
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] NumberofChildren;
  if(even)
  {
    NumberofChildren = new byte[length];
    for(i = 0; i < length; i++)
      NumberofChildren[i] = rawarray[count + i];
  }
  else
  {
    NumberofChildren = new byte[length + 1];
    NumberofChildren[0] = 0x00;
    for(i = 0; i < length; i++)
      NumberofChildren[i+1] = rawarray[count + i];
  }
  myVBLKVolume.NumberofChildren = convertValue(NumberofChildren
      , length);
  count += length;
}

myVBLKVolume.LogCommitId = (ulong)(rawarray[count]);
count += 8;

Array.ConstrainedCopy(rawarray, count, myVBLKVolume.Id, 0, 8);
count += 8;

//Size
{
  Length = rawarray[count];
```

```
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] size;
    if(even)
    {
      size = new byte[length];
      for(i = 0; i < length; i++)
        size[i] = rawarray[count + i];
    }
    else
    {
      size = new byte[length + 1];
      size[0] = 0x00;
      for(i = 0; i < length; i++)
        size[i+1] = rawarray[count + i];
    }
    myVBLKVolume.Size = convertValue(size, length);
    count += length;
  }

  //Zero
  count += 4;

  myVBLKVolume.PartitionType = (char)(rawarray[count]);
  count += 1;

  Array.ConstrainedCopy(rawarray, count, myVBLKVolume.VolumeID,
      0, 16);
  count += 16;
```

```
byte[] recordType = BitConverter.GetBytes(myVBLKHeader.
    RecordType);
if (recordType.Length < 2)
  return false;

//Optional Information Based on RecordType flag
if (recordType[0] == 0x08)
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr;
  tempStr = new char[length];
  myVBLKVolume.Id1_length = (ushort)length;
  for(i = 0; i < length; i++)
     tempStr[i] = (char)rawarray[count + i];
  myVBLKVolume.Id1 = new String(tempStr);
  count += length;
}
else if (recordType[0] == 0x20)
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr;
  tempStr = new char[length];
  myVBLKVolume.Id2_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKVolume.Id2 = new String(tempStr);
  count += length;
}
```

```
else if (recordType[0] == 0x80)
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] OptSize;
  if(even)
  {
    OptSize = new byte[length];
    for(i = 0; i < length; i++)
      OptSize[i] = rawarray[count + i];
  }
  else
  {
    OptSize = new byte[length + 1];
    OptSize[0] = 0x00;
    for(i = 0; i < length; i++)
      OptSize[i+1] = rawarray[count + i];
  }
  myVBLKVolume.OptSize = convertValue(OptSize, length);
  count += length;
}
else if (recordType[0] == 0x02)
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr;
  tempStr = new char[length];
  myVBLKVolume.DriveHint_length = (ushort)length;
```

```
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.DriveHint = new String(tempStr);
    count += length;
  }
  else if (recordType[0] == 0x88)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    char[] tempStr;
    tempStr = new char[length];
    myVBLKVolume.Id1_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.Id1 = new String(tempStr);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] OptSize;
    if(even)
    {
      OptSize = new byte[length];
      for(i = 0; i < length; i++)
        OptSize[i] = rawarray[count + i];
    }
    else
    {
```

```
      OptSize = new byte[length + 1];
      OptSize[0] = 0x00;
      for(i = 0; i < length; i++)
        OptSize[i+1] = rawarray[count + i];
    }
    myVBLKVolume.OptSize = convertValue(OptSize, length);
    count += length;
  }
  else if (recordType[0] == 0x0A)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    char[] tempStr;
    tempStr = new char[length];
    myVBLKVolume.Id1_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.Id1 = new String(tempStr);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    tempStr = new char[length];
    myVBLKVolume.DriveHint_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.DriveHint = new String(tempStr);
    count += length;
  }
  else if (recordType[0] == 0x8A)
```

```
{
  Length = rawarray[count];
  count++;
  length = (short)Length;
  char[] tempStr;
  tempStr = new char[length];
  myVBLKVolume.Id1_length = (ushort)length;
  for(i = 0; i < length; i++)
    tempStr[i] = (char)rawarray[count + i];
  myVBLKVolume.Id1 = new String(tempStr);
  count += length;

  Length = rawarray[count];
  count++;
  length = (short)Length;
  even = true;
  if(length%2 != 0) even = false;
  byte[] OptSize;
  if(even)
  {
    OptSize = new byte[length];
    for(i = 0; i < length; i++)
      OptSize[i] = rawarray[count + i];
  }
  else
  {
    OptSize = new byte[length + 1];
    OptSize[0] = 0x00;
    for(i = 0; i < length; i++)
      OptSize[i+1] = rawarray[count + i];
  }
  myVBLKVolume.OptSize = convertValue(OptSize, length);
```

```
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    tempStr = new char[length];
    myVBLKVolume.DriveHint_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.DriveHint = new String(tempStr);
    count += length;
  }
  else if (recordType[0] == 0xA0)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    char[] tempStr;
    tempStr = new char[length];
    myVBLKVolume.Id2_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.Id2 = new String(tempStr);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] OptSize;
    if(even)
```

```
    {
      OptSize = new byte[length];
      for(i = 0; i < length; i++)
        OptSize[i] = rawarray[count + i];
    }
    else
    {
      OptSize = new byte[length + 1];
      OptSize[0] = 0x00;
      for(i = 0; i < length; i++)
        OptSize[i+1] = rawarray[count + i];
    }
    myVBLKVolume.OptSize = convertValue(OptSize, length);
    count += length;
  }
  else if (recordType[0] == 0x22)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    char[] tempStr;
    tempStr = new char[length];
    myVBLKVolume.Id2_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.Id2 = new String(tempStr);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    tempStr = new char[length];
```

```
    myVBLKVolume.DriveHint_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.DriveHint = new String(tempStr);
    count += length;
  }
  else if (recordType[0] == 0xA2)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    char[] tempStr;
    tempStr = new char[length];
    myVBLKVolume.Id2_length = (ushort)length;
    for(i = 0; i < length; i++)
      tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.Id2 = new String(tempStr);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] OptSize;
    if(even)
    {
      OptSize = new byte[length];
      for(i = 0; i < length; i++)
        OptSize[i] = rawarray[count + i];
    }
    else
```

```
    {
      OptSize = new byte[length + 1];
      OptSize[0] = 0x00;
      for(i = 0; i < length; i++)
        OptSize[i+1] = rawarray[count + i];
    }
    myVBLKVolume.OptSize = convertValue(OptSize, length);
    count += length;

    Length = rawarray[count];
    count++;
    length = (short)Length;
    tempStr = new char[length];
    myVBLKVolume.DriveHint_length = (ushort)length;
    for(i = 0; i < length; i++)
       tempStr[i] = (char)rawarray[count + i];
    myVBLKVolume.DriveHint = new String(tempStr);
    count += length;
  }
  else if (recordType[0] == 0x82)
  {
    Length = rawarray[count];
    count++;
    length = (short)Length;
    even = true;
    if(length%2 != 0) even = false;
    byte[] OptSize;
    if(even)
    {
      OptSize = new byte[length];
      for(i = 0; i < length; i++)
        OptSize[i] = rawarray[count + i];
```

```
      }
      else
      {
        OptSize = new byte[length + 1];
        OptSize[0] = 0x00;
        for(i = 0; i < length; i++)
          OptSize[i+1] = rawarray[count + i];
      }
      myVBLKVolume.OptSize = convertValue(OptSize, length);
      count += length;

      Length = rawarray[count];
      count++;
      length = (short)Length;
      char[] tempStr;
      tempStr = new char[length];
      myVBLKVolume.DriveHint_length = (ushort)length;
      for(i = 0; i < length; i++)
        tempStr[i] = (char)rawarray[count + i];
      myVBLKVolume.DriveHint = new String(tempStr);
      count += length;
    }

    if(BitConverter.IsLittleEndian)
    {
      myVBLKVolume.LogCommitId = (ulong)IPAddress.
          HostToNetworkOrder((long)myVBLKVolume.LogCommitId);
    }
  }

  if(myVBLKHeader.NumberofRecords == 0)
    empty = true;
```

```
  return !fail;
}

private ulong convertValue(byte[] rawarray, short length)
{
  ulong returnVal = 0L;
  if (length == 0)
    return 0;
  if (length <= 2)
  {
    short intermediate = 0;
    intermediate = BitConverter.ToInt16(rawarray, 0);
    if (BitConverter.IsLittleEndian)
      returnVal = (ulong)IPAddress.HostToNetworkOrder(intermediate)
          ;
  }
  else if (length <= 4)
  {
    int intermediate = 0;
    intermediate = BitConverter.ToInt32(rawarray, 0);
    if (BitConverter.IsLittleEndian)
      returnVal = (ulong)IPAddress.HostToNetworkOrder(intermediate)
          ;
  }
  else if (length <= 8)
  {
    returnVal = (ulong)BitConverter.ToInt64(rawarray, 0);
    if (BitConverter.IsLittleEndian)
      returnVal = (ulong)IPAddress.HostToNetworkOrder((long)
          returnVal);
  }
  return returnVal;
```

```
    }
}
```

Code/VBLK.cs

# APPENDIX B

## Important Driver Code

### B.1 RAID Configuration Storing

```
//
   ////////////////////////////////////////////////////////////////////////////

//
//  CreateConnection
//
//    Creates a connection to the specified volume, if it does not
   already
//    exists.
//
//  INPUTS:
//
//    PGInfo - Pointer to the Global Information BLock.
//
//    PConnectInfo - Pointer to the connection information to create
//
//  OUTPUTS:
//
//    None.
//
//  RETURNS:
//
//    STATUS_SUCCESS if okay, an error otherwise.
//
//  IRQL:
//
//  This routine is called at any IRQL PASSIVE_LEVEL.
```

```
//
//  NOTES:
//
//
    ////////////////////////////////////////////////////////////////////////////

NTSTATUS CreateConnection(PUSER_GLOBAL_INFORMATION PGInfo,
    PCONNECT_IN PConnectInfo)
{
  NTSTATUS        status = STATUS_UNSUCCESSFUL;
  IO_STATUS_BLOCK  ioStatus;
  BOOLEAN         bInserted = FALSE;
  OBJECT_ATTRIBUTES    objectAttributes;
  UNICODE_STRING      uString;
  KIRQL           oldIrql;
  GUID           tmpGuid;
  ULONG           bytesReturned;

  OsrTracePrint(TRACE_LEVEL_VERBOSE,OSRVMINIPT_DEBUG_FUNCTRACE,(
      __FUNCTION__": Enter\n"));

  //
  // See if we already have a connection that matches this.
  //
  if(FindConnectionMatch(PGInfo,PConnectInfo,NULL)) {
    return STATUS_OBJECT_NAME_COLLISION;
  }

  RtlZeroMemory(&tmpGuid,sizeof(GUID));

  //
  // Add the connection to the list.
```

```
//
PCONNECTION_LIST_ENTRY pEntry = (PCONNECTION_LIST_ENTRY)
  ExAllocatePoolWithTag(NonPagedPool, sizeof(CONNECTION_LIST_ENTRY),
      'pCLE');

if(!pEntry) {
  return STATUS_INSUFFICIENT_RESOURCES;
}

RtlZeroMemory(pEntry, sizeof(CONNECTION_LIST_ENTRY));

OsrAcquireSpinLock(&PGInfo->ConnectionListLock,&oldIrql);

InsertTailList(&PGInfo->ConnectionList,&pEntry->ListEntry);

OsrReleaseSpinLock(&PGInfo->ConnectionListLock, oldIrql);
bInserted = TRUE;

//I get a warning if I don't use wcscpy_s which should be defined
    in <wchar.h> but even including that
// I get a linker error so I have just commented it out for now
StringCchCopyW(pEntry->VolumeID, sizeof(pEntry->VolumeID)/sizeof(
    WCHAR), PConnectInfo->VolumeID);

//wcscpy(pEntry->VolumeID, PConnectInfo->VolumeID);
KdPrint(( "VolumeID: %s", pEntry->VolumeID ));
pEntry->TotalDiskSize = 0;
pEntry->NumOfDisks = PConnectInfo->NumOfDisks;
KdPrint(( "NumOfDisks: %d", pEntry->NumOfDisks ));
pEntry->Raid = PConnectInfo->Raid;
KdPrint(( "Raid: %d", pEntry->Raid ));
pEntry->StripeSize = PConnectInfo->StripeSize;
```

```
KdPrint(( "StripeSize: %d", pEntry->StripeSize ));
pEntry->SectorSize = PConnectInfo->SectorSize;
KdPrint(( "SectorSize: %d", pEntry->SectorSize ));

for(unsigned int i = 0; i < pEntry->NumOfDisks; i++)
{
  pEntry->TotalDiskSize += PConnectInfo->DiskSize[i];
}

KdPrint(( "TotalDiskSize: %d", pEntry->TotalDiskSize ));

for(unsigned int i = 0; i < pEntry->NumOfDisks; i++)
{
  pEntry->DiskStart[i] = PConnectInfo->DiskStart[i];
  pEntry->DiskSize[i] = PConnectInfo->DiskSize[i];
  KdPrint(( "DiskStart: %d", pEntry->DiskStart[i] ));
  KdPrint(( "DiskSize: %d", pEntry->DiskSize[i] ));
}

for(unsigned int i = 0; i < pEntry->NumOfDisks; i++)
{
  StringCchCopyW(pEntry->DiskPath[i], sizeof(pEntry->DiskPath[i])/
      sizeof(WCHAR), PConnectInfo->DiskPath[i]);
  KdPrint(( "DiskPath: %s", pEntry->DiskPath[i] ));
}

//
// For our Virtual Disks, it comes from the Disk Header.
//
status = ExUuidCreate(&tmpGuid);

if(!NT_SUCCESS(status)) {
```

```
    goto cleanupAfterError;
  }

  //
  // We now have the information about the file that this disk, which
      we are about to
  // create, represents.  We need to build some SCSI inquiry
      information about the
  // disk, so that the Disk Class Driver knows about us.
  //
#pragma prefast(suppress:28197,"This memory is not leaked")
  PINQUIRYDATA pInquiryData = (PINQUIRYDATA) ExAllocatePoolWithTag(
      NonPagedPool,
                              sizeof(INQUIRYDATA),
                              'diSO');

  if(pInquiryData) {

    // typedef struct _INQUIRYDATA {
    //   UCHAR DeviceType : 5;
    //   UCHAR DeviceTypeQualifier : 3;
    //   UCHAR DeviceTypeModifier : 7;
    //   UCHAR RemovableMedia : 1;
    //   UCHAR Versions;
    //   UCHAR ResponseDataFormat : 4;
    //   UCHAR HiSupport : 1;
    //   UCHAR NormACA : 1;
    //   UCHAR ReservedBit : 1;
    //   UCHAR AERC : 1;
    //   UCHAR AdditionalLength;
    //   UCHAR Reserved[2];
    //   UCHAR SoftReset : 1;
```

```
//    UCHAR CommandQueue : 1;
//    UCHAR Reserved2 : 1;
//    UCHAR LinkedCommands : 1;
//    UCHAR Synchronous : 1;
//    UCHAR Wide16Bit : 1;
//    UCHAR Wide32Bit : 1;
//    UCHAR RelativeAddressing : 1;
//    UCHAR VendorId[8];
//    UCHAR ProductId[16];
//    UCHAR ProductRevisionLevel[4];
//    UCHAR VendorSpecific[20];
//    UCHAR Reserved3[40];
// } INQUIRYDATA, *PINQUIRYDATA;

RtlZeroMemory(pInquiryData, sizeof(INQUIRYDATA));

//
// The media is now either an OSR Disk or a regular disk, either
    way
// we return the same information.
//
pInquiryData->DeviceType = DIRECT_ACCESS_DEVICE;
pInquiryData->DeviceTypeQualifier = DEVICE_CONNECTED;
pInquiryData->DeviceTypeModifier = 0;
pInquiryData->RemovableMedia = TRUE;
pInquiryData->Versions = 2;        // SCSI-2 support
pInquiryData->ResponseDataFormat = 2;    // Same as Version??
    according to SCSI book
pInquiryData->Wide32Bit = TRUE;     // 32 bit wide transfers
pInquiryData->Synchronous = TRUE;     // Synchronous commands
pInquiryData->CommandQueue = FALSE;  // Does not support tagged
    commands
```

```
    pInquiryData->AdditionalLength = INQUIRYDATABUFFERSIZE-5;   //
        Amount of data we are returning
    pInquiryData->LinkedCommands = FALSE;     // No Linked Commands
    RtlCopyMemory((PUCHAR) &pInquiryData->VendorId[0],
        OSR_INQUIRY_VENDOR_ID,
                    strlen(OSR_INQUIRY_VENDOR_ID));
    RtlCopyMemory((PUCHAR) &pInquiryData->ProductId[0],
        OSR_INQUIRY_PRODUCT_ID,
                    strlen(OSR_INQUIRY_PRODUCT_ID));
    RtlCopyMemory((PUCHAR) &pInquiryData->ProductRevisionLevel[0],
        OSR_INQUIRY_PRODUCT_REVISION,
                strlen(OSR_INQUIRY_PRODUCT_REVISION));
    RtlCopyMemory((PUCHAR) &pInquiryData->VendorSpecific[0],
        OSR_INQUIRY_VENDOR_SPECIFIC,
                strlen(OSR_INQUIRY_VENDOR_SPECIFIC));


   ULONG bitNumber = RtlFindClearBitsAndSet(&ScsiBitMapHeader,1,0);

    if(bitNumber == 0xFFFFFFFF) {
      status = STATUS_INSUFFICIENT_RESOURCES;
      DoClose(PGInfo,pEntry);
      goto cleanupAfterError;
    }

   ULONG targetId = bitNumber % SCSI_MAXIMUM_TARGETS_PER_BUS;
   ULONG BusId = bitNumber / SCSI_MAXIMUM_BUSES;

#pragma prefast(suppress:28197,"This memory is not leaked")
    PUSER_INSTANCE_INFORMATION pLocalInfo = (
        PUSER_INSTANCE_INFORMATION)
                        ExAllocatePoolWithTag(NonPagedPool,
```

```
                                sizeof(USER_INSTANCE_INFORMATION),
                                'DLUp');

if(!pLocalInfo) {
  status = STATUS_INSUFFICIENT_RESOURCES;
  DoClose(PGInfo,pEntry);
  goto cleanupAfterError;
}

RtlZeroMemory(pLocalInfo,sizeof(USER_INSTANCE_INFORMATION));
pLocalInfo->MagicNumber = USER_INSTANCE_INFORMATION_MAGIC_NUMBER;
pLocalInfo->PInquiryData = pInquiryData;

//
// Create a PDO for this new disk.
//

pLocalInfo->OsrSPLocalHandle = OsrSPCreateScsiDevice(PGInfo->
    OsrSPHandle,
                        BusId /*IN ULONG BusIndex*/,
                        targetId /*IN ULONG TargetIndex*/,
                        LunId /*IN ULONG LunIndex*/,
                        pLocalInfo, /* Our local Data for Device */
                        FALSE,
                        pInquiryData,
                        1);

//
// Okay, we've got a PDO, we can now invalidate relations and see
     what happens.
//
```

```
if(pLocalInfo) {

  static ULONG indexNumber = 0x08051958;

  pLocalInfo->PGInfo = PGInfo;

  //
  // Get the infor for the unique ID.
  //
  GUID* pUniqueId = &tmpGuid;

  RtlCopyMemory(&pLocalInfo->UniqueID.UniqueID,pUniqueId,sizeof(
      GUID));
  pLocalInfo->UniqueID.FileId = (ULONGLONG) InterlockedIncrement
      ((volatile LONG*) &indexNumber);

  //
  // Store away some other useful information.
  //
  pLocalInfo->ConnectionInformation = pEntry;
  pLocalInfo->TargetIndex = targetId;
  pLocalInfo->BusIndex = BusId;
  pLocalInfo->LunIndex = LunId;
  if(STATUS_SUCCESS != RtlStringCbPrintfA(&pLocalInfo->
      AsciiSignature[0],
    sizeof(pLocalInfo->AsciiSignature),
    "%08x%04x%04x%2x%2x%02x%02x%02x%02x%02x%02x%0I64x",
    pUniqueId->Data1,pUniqueId->Data2,pUniqueId->Data3,
    pUniqueId->Data4[0],pUniqueId->Data4[1],pUniqueId->Data4[2],
        pUniqueId->Data4[3],
    pUniqueId->Data4[5],pUniqueId->Data4[5],pUniqueId->Data4[6],
        pUniqueId->Data4[7],
```

```
        pLocalInfo->UniqueID.FileId)) {
        status = STATUS_INSUFFICIENT_RESOURCES;
        DoClose(PGInfo, pEntry);
        goto cleanupAfterError;
      }

      pEntry->PIInfo = pLocalInfo;
      pEntry->BusIndex = BusId;
      pEntry->TargetIndex = targetId;
      pEntry->LunIndex = LunId;

      InterlockedIncrement(&PGInfo->ConnectionCount);

      targetId++;

      //
      // Tell the OSR SP that our bus has changed.
      //
      OsrSPAnnounceArrival(PGInfo->OsrSPHandle);

      pEntry->Connected = TRUE;

      status = STATUS_SUCCESS;

    }

    OsrTracePrint(TRACE_LEVEL_VERBOSE, OSRVMINIPT_DEBUG_FUNCTRACE, (
        __FUNCTION__": Exit\n"));

    return status;
  } else {
    status = STATUS_INSUFFICIENT_RESOURCES;
```

```
    }

cleanupAfterError:

    if(bInserted) {

        DeleteConnectionEntry(PGInfo,pEntry,PConnectInfo);

    }

    if(pEntry) {

        ExFreePool(pEntry);

    }

    OsrTracePrint(TRACE_LEVEL_ERROR,OSRVMINIPT_DEBUG_FUNCTRACE,(
        __FUNCTION__": Exit\n"));

    return status;
}
```

Code/RaidConfig.cpp

## B.2 Mirrored Read Method

```
if(pConnectionInformation->Raid == None) //SIMPLE OR MIRRORED
{
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("Simple
        : Enter"));

    if(ReadLbn + readLength > pConnectionInformation->DiskSize[0]) {
      readLength0 = (ULONG) (pConnectionInformation->DiskSize[0] -
          ReadLbn);
```

```
    Read.QuadPart += pConnectionInformation->DiskStart[0];

    status = ReadDisk(0, Read, readLength0, pBuffer,
        pConnectionInformation);
    if(status != STATUS_SUCCESS)
    {
      goto cleanupAfterError;
    }

    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
        Simple: Exit Success"));
    PSrb->SrbStatus = SRB_STATUS_SUCCESS;
    PSrb->DataTransferLength = readLength0;
    goto cleanupAfterError;
  }
  else
  {
    //
    // Offset for disk 0's start position
    //
    Read.QuadPart += pConnectionInformation->DiskStart[0];
    status = ReadDisk(0, Read, readLength, pBuffer,
        pConnectionInformation);
    if(status != STATUS_SUCCESS)
    {
      goto cleanupAfterError;
    }
  }

  OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("Simple
      : Exit Success"));
```

```
}
```

Code/MirroredRead.cpp

## B.3 Striped Read Method

```
else if(pConnectionInformation->Raid == Striped) //STRIPED
{
  OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
      Striped: Enter"));
  //First we find what disk the read should be on
  currentDisk = ((ReadLbn / pConnectionInformation->StripeSize) %
      pConnectionInformation->NumOfDisks);
  //Next we find what stripe the read is on on that disk
  stripeCount = static_cast<unsigned int>(((ReadLbn /
      pConnectionInformation->StripeSize) / pConnectionInformation->
      NumOfDisks));

  //readLength0 stores how much has been read successfully
  readLength0 = 0;
  //readLength1 is set as the amount of space left to read in this
      stripe
  readLength1 = static_cast<ULONG>(pConnectionInformation->StripeSize
      - (ReadLbn % pConnectionInformation->StripeSize));
  //readLengthHold is set as the amount left to read after reading
      the rest of this stripe
  readLengthHold = readLength - readLength1;
  //If the hold is negative then we only need to read this stripe
  if(readLengthHold < 0)
  {
    readLength1 = readLength;
    readLengthHold = 0;
  }
```

```
//We reset the read to be the proper amount into the correct stripe
Read.QuadPart = pConnectionInformation->DiskStart[currentDisk] + (
    pConnectionInformation->StripeSize * stripeCount) + (ReadLbn %
    pConnectionInformation->StripeSize);

//As long as we havn't read off of the end of all the disks we can
    continue until we are done
while(!finishedReading && static_cast<ULONGLONG>(Read.QuadPart +
    readLength1) < pConnectionInformation->TotalDiskSize)
{
  status = ReadDisk(currentDisk, Read, readLength1, &(((char*)
      pBuffer)[readLength0]), pConnectionInformation);
  if(status != STATUS_SUCCESS)
  {
    goto cleanupAfterError;
  }

  readLength0 += readLength1;
  readLength1 = readLengthHold;
  currentDisk++;

  //If it is the NumOfDisks then we need to wrap to the next stripe
  if(currentDisk == pConnectionInformation->NumOfDisks)
  {
    currentDisk = 0;
    stripeCount++;
  }

  //We reset the read pointer to the beginning of the next stripe
  Read.QuadPart = pConnectionInformation->DiskStart[currentDisk] +
      (pConnectionInformation->StripeSize * stripeCount);
```

```
  readLengthHold = 0;
  if(readLength1 > pConnectionInformation->StripeSize)
  {
    readLengthHold = static_cast<LONG>(readLength1 -
        pConnectionInformation->StripeSize);
    readLength1 = static_cast<LONG>(pConnectionInformation->
        StripeSize);
  }

  //Once it is 0 we are done reading
  if(readLength1 == 0)
  {
    finishedReading = true;
  }
}

//
// If not finished reading then we have run out of disk
//
if(!finishedReading && static_cast<ULONGLONG>(Read.QuadPart) <
    pConnectionInformation->TotalDiskSize)
{
  readLength1 = static_cast<LONG>(pConnectionInformation->
      TotalDiskSize - Read.QuadPart);

  status = ReadDisk(currentDisk, Read, readLength1, &(((char*)
      pBuffer)[readLength0]), pConnectionInformation);
  if(status != STATUS_SUCCESS)
  {
    goto cleanupAfterError;
  }
```

```
      readLength0 += readLength1;

      PSrb->SrbStatus = SRB_STATUS_SUCCESS;
      PSrb->DataTransferLength = readLength0;
      goto cleanupAfterError;
    }
    else if(!finishedReading)
    {
      PSrb->SrbStatus = SRB_STATUS_SUCCESS;
      PSrb->DataTransferLength = readLength0;
      goto cleanupAfterError;
    }
}
```

Code/StripedRead.cpp

## B.4 Spanned Read Method

```
else if(pConnectionInformation->Raid == Spanned) //SPANNED
{
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
        Spanned: Enter\n"));
    //First we find out what disk the read starts on
    currentDisk = 0;
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
        Initial ReadLbn: %llu\n", ReadLbn));
    while(ReadLbn > pConnectionInformation->DiskSize[currentDisk])
    {
      ReadLbn -= pConnectionInformation->DiskSize[currentDisk];
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          Middle ReadLbn: %llu\n", ReadLbn));
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          DiskSize: %llu\n", pConnectionInformation->DiskSize[
          currentDisk]));
```

```
    currentDisk++;
}
Read.QuadPart = ReadLbn;

//
// Does the read wrap from one disk the next
//
if((currentDisk + 1) < pConnectionInformation->NumOfDisks && (
    ReadLbn + readLength) > pConnectionInformation->DiskSize[
    currentDisk])
{
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("We
        wrapped because:\n"));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tNext Disk: %d < NumOfDisks: %d\n", currentDisk + 1,
        pConnectionInformation->NumOfDisks));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tRead: %d > DiskSize: %d\n", ReadLbn + readLength,
        pConnectionInformation->DiskSize[currentDisk]));
    //
    // Read to the end of the first disk and then read the rest from
        the beginning of the next disk
    //
    readLength0 = (ULONG) (pConnectionInformation->DiskSize[
        currentDisk] - ReadLbn);
    readLength1 = readLength - readLength0;
    //
    // Offset for the disk's start position and then read to the end
    //
    Read.QuadPart += pConnectionInformation->DiskStart[currentDisk];
    status = ReadDisk(currentDisk, Read, readLength0, pBuffer,
        pConnectionInformation);
```

```
    if(status != STATUS_SUCCESS)
    {
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          CLEANUP_AFTER_ERROR\n"));
      goto cleanupAfterError;
    }
    //
    // Offset for the next disk's start position and then read the
        rest of the length
    //
    Read.QuadPart = pConnectionInformation->DiskStart[currentDisk+1];
    status = ReadDisk(currentDisk+1, Read, readLength1, &(((char*)
        pBuffer)[readLength0]), pConnectionInformation);
    if(status != STATUS_SUCCESS)
    {
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          CLEANUP_AFTER_ERROR\n"));
      goto cleanupAfterError;
    }
  }
  else if((ReadLbn + readLength) > pConnectionInformation->DiskSize[
      currentDisk])
  {
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("We
        wrapped because:\n"));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tNext Disk: %d >= NumOfDisks: %d\n", currentDisk + 1,
        pConnectionInformation->NumOfDisks));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tRead: %d > DiskSize: %d\n", ReadLbn + readLength,
        pConnectionInformation->DiskSize[currentDisk]));
    readLength0 = (ULONG) (pConnectionInformation->DiskSize[
```

```
        currentDisk] - ReadLbn);
    Read.QuadPart += pConnectionInformation->DiskStart[currentDisk];

    status = ReadDisk(currentDisk, Read, readLength0, pBuffer,
        pConnectionInformation);
    if(status != STATUS_SUCCESS)
    {
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          CLEANUP_AFTER_ERROR\n"));
      goto cleanupAfterError;
    }

    PSrb->SrbStatus = SRB_STATUS_SUCCESS;
    PSrb->DataTransferLength = readLength0;
    goto cleanupAfterError;
  }
  else if(currentDisk < pConnectionInformation->NumOfDisks)
  {
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("We
        didn't wrap because:\n"));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tCurrent Disk: %d < NumOfDisks: %d\n", currentDisk,
        pConnectionInformation->NumOfDisks));
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("\
        tRead: %d <= DiskSize: %d\n", ReadLbn + readLength,
        pConnectionInformation->DiskSize[currentDisk]));
    //
    // Offset for the current disk's start position
    //
    Read.QuadPart += pConnectionInformation->DiskStart[currentDisk];
    status = ReadDisk(currentDisk, Read, readLength, pBuffer,
        pConnectionInformation);
```

```
    if(status != STATUS_SUCCESS)
    {
      OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
          CLEANUP_AFTER_ERROR\n"));
      goto cleanupAfterError;
    }
  }
  else
  {
    OsrTracePrint(TRACE_LEVEL_INFORMATION,OSRVMINIPT_DEBUG_ALL,("
        CLEANUP_AFTER_ERROR\n"));
    goto cleanupAfterError;
  }
}
```

Code/SpannedRead.cpp

# BIBLIOGRAPHY


adaptec. "Aha-2940uw." [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.adaptec.com/en-us/support/scsi/2940/aha-2940uw/

ASR Data. "Smart linux." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.asrdata.com/forensic-software/smart-linux/

ASUS. "N61jq." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.asus.com/Notebooks/Multimedia_Entertainment/N61Jq/

cplusplus.com. "History of c++." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.cplusplus.com/info/history

Crystal Dew World. "Crystaldiskmark." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://crystalmark.info/software/CrystalDiskMark/index-e.html

CSharp-Online.NET. "C# overview." [Online; accessed 9-March-2012]. [Online]. Available: en.csharp-online.net/CSharp_Overview#A_Brief_History_of_C.23

Dell. "Optiplex 760 desktop." [Online; accessed 9-March-2012]. 2012. [Online]. Available: www.dell.com/us/dfb/p/optiplex-760/pd

Dickerman, S. D., "Raid rebuilding," pdf, 2007.

Digital Intelligence. "Encase forensic v7." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.digitalintelligence.com/software/guidancesoftware/encase7/

EFD Software. "Hd tune." [Online; accessed 13-December-2011]. Aug. 2010. [Online]. Available: http://www.hdtune.com/

Eindhoven University of Technology. "List of partition identifiers for pcs." [Online; accessed 1-October-2011]. [Online]. Available: http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

EUSSO Technologies, Inc. "4-bay sata gigibit network terabank nas." [Online; accessed 30-September-2011]. [Online]. Available: http://www.eusso.com/Models/NAS/USS4500-RS4/USS4500-RS4.htm

eXibition Software. "Drive speed checker." [Online; accessed 13-December-2011]. 2004. [Online]. Available: http://www.exibitionsoftware.com/products/drivespeedchecker/details.asp

ForensicSoft. “Safe block.” [Online; accessed 1-October-2011]. 2010. [Online]. Available: https://www.forensicsoft.com/safeblock.php

ForensicSoft. “Software write blockers.” [Online; accessed 1-October-2011]. 2010. [Online]. Available: https://www.forensicsoft.com/sb_features.php

Futuremark. “Pcmark pc performance testing.” [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.pcmark.com/

“Encase version 6.12 modules manual,” Guidance Software.

GuidenceSoftware. “Encase forensic.” [Online; accessed 1-October-2011]. 2011. [Online]. Available: http://www.guidancesoftware.com/forensic.htm

Icon Archive. “Computer icon.” [Online; accessed 30-September-2011]. [Online]. Available: http://www.iconarchive.com/show/vista-hardware-devices-icons-by-icons-land/Computer-icon.html

intel. “Intel®core™2 duo processor e7300.”

Kato, K. “Virtual disk driver version 3.” [Online; accessed 13-December-2011]. Apr. 2005. [Online]. Available: http://chitchat.at.infoseek.co.jp/vmware/vdk.html#top

Kato, K. “Virtual floppy drive 2.1.” [Online; accessed 13-December-2011]. Feb. 2008. [Online]. Available: http://chitchat.at.infoseek.co.jp/vmware/vfd.html#top

Kovacs, J. “C#/.net history lesson.” [Online; accessed 9-March-2012]. Sept. 2007. [Online]. Available: jameskovacs.com/2007/09/07/cnet-history-lesson/

Kroll Ontrack. “Raid: History and information.” [Online; accessed 30-September-2011]. [Online]. Available: http://www.ontrackdatarecovery.co.uk/data-recovery-articles/raid-history-information/

The Linux Information Project. “Kernel definition.” [Online; accessed 13-December-2011]. May 2005. [Online]. Available: http://www.linfo.org/kernel.html

Linux-NTFS. “Linux-ntfs.” [Online; accessed 1-October-2011]. Feb. 2009. [Online]. Available: http://www.linux-ntfs.org/doku.php

Linux Tutorial. “Configure and install ubuntu on raid 0.” [Online; accessed 2-May-2012]. July 2011. [Online]. Available: www.numango.com/5078_install-ubuntu-on-raid.html

Microsoft. “Windows driver model (wdm).” [Online; accessed 9-March-2012]. Apr. 2002. [Online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463453

Microsoft. “Introduction to the windows driver foundation.” [Online; accessed 9-March-2012]. Oct. 2003. [Online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463316

Microsoft. “What are dynamic disks and volumes?” [Online; accessed 13-December-2011]. Mar. 2003. [Online]. Available: http://technet.microsoft.com/en-us/library/cc737048(v=ws.10).aspx

Microsoft. “What is virtual disk service?” [Online; accessed 9-March-2012]. Mar. 2003. [Online]. Available: http://technet.microsoft.com/en-us/library/cc778187(v=ws.10).aspx

Microsoft. “History of storport.” [Online; accessed 9-March-2012]. Feb. 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557249(v=vs.85).aspx

Microsoft. “Storport driver.” [Online; accessed 9-March-2012]. Feb. 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff567541(v=vs.85).aspx

Microsoft. “Windows enterprise.” [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.microsoft.com/en-us/windows/enterprise/products-and-technologies/windows-7/default.aspx

Novell. “Cross platform, open source .net development framework.” [Online; accessed 9-March-2012]. [Online]. Available: www.mono-project.com/Main_Page

Open Source Development Lab. “Iometer.” [Online; accessed 13-December-2011]. July 2006. [Online]. Available: www.iometer.org

Orwick, P., *Developing Drivers with the Windows Driver Foundation*. One Microsoft Way, Redmond, Washington 98052-6399: Microsoft Press, 2007.

OSR Online. “Writing a virtual storport miniport driver.” [Online; accessed 13-December-2011]. Sept. 2009. [Online]. Available: http://www.osronline.com/article.cfm?article=538

Petri IT Knowledgebase. “Difference between basic and dynamic disks in windows xp/2000/2003.” [Online; accessed 30-September-2011]. [Online]. Available: http://www.petri.co.il/difference_between_basic_and_dynamic_disks_in_windows_xp_2000_2003.htm

Runtime Software. “Raid recovery for windows v1.01.” [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.runtime.org/raid-recovery-windows.htm

Russon, R., “Home - ldm documentation,” 2002.

Seagate. "Quantum®atlas 10k ii." [Online; accessed 9-March-2012]. 2000. [Online]. Available: http://www.seagate.com/staticfiles/maxtor/en_us/documentation/data_sheets/atlas_10k_ii_datasheet.pdf

simplisoftware. "Hd tach." [Online; accessed 13-December-2011]. [Online]. Available: http://www.simplisoftware.com/Public/index.php?request=HdTach

steel bytes. "Hd_speed." Jan. [Online]. Available: http://www.steelbytes.com/?mid=20

"Prodiscover forensics," pdf, Technology Pathways, Aug. 2009.

Technology Pathways. "Prodiscover forensics." [Online; accessed 13-December-2011]. 2010. [Online]. Available: http://www.techpathways.com/prodiscoverdft.htm

Troelsen, A., *Pro C# 2010 and the .NET 4 Platform, Fifth Edition*. 233 Spring Street, New York, New York 10013: Apress, 2010.

Ubuntu. "Download ubuntu." [Online; accessed 2-May-2012]. 2012. [Online]. Available: http://www.ubuntu.com/download/server

VMware, Inc. "Vmware workstation 8." [Online; accessed 9-March-2012]. 2012. [Online]. Available: http://www.vmware.com/products/workstation/

Wikipedia, "Disk image — wikipedia, the free encyclopedia," 2011, [Online; accessed 13-December-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Disk_image&oldid=465613179

Wikipedia, "Guid partition table — wikipedia, the free encyclopedia," 2011, [Online; accessed 1-October-2011]. [Online]. Available: //en.wikipedia.org/w/index.php?title=GUID_Partition_Table&oldid=452606816

Wikipedia, "Raid — wikipedia, the free encyclopedia," 2011, [Online; accessed 13-December-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=RAID&oldid=465701597

X-Ways. "Winhex." [Online; accessed 13-December-2011]. Mar. 2010. [Online]. Available: http://www.winhex.com/winhex/

"X-ways forensics/winhex," pdf, X-Ways, 2011.

Zero Assumption Recovery. "Ldm / dynamic disks basics." [Online; accessed 13-December-2011]. 2011. [Online]. Available: http://www.z-a-recovery.com/art-dynamic-disks.htm