

# White Paper: Using and Automating the Virtual Disk Shrink Feature

Document Version 1.1

**RTFM Education**

Beyond the Manual... with Mike Laverick

By Mike Laverick  
© RTFM Education

Contact:  
[mikelaverick@rtfm-ed.co.uk](mailto:mikelaverick@rtfm-ed.co.uk)

**ESX Version:**

2.x

**Author:**

Mike Laverick 

**Audience:**

Novice to Enthusiast

**Style:**

Bit techy with some humour, not a po-faced article. I use unhappy faces ☹ to flag up events and experiences that are less than pleasant. I use a happy face 😊 to flag up something which is very advantageous.

**Objective:**

This article discusses the purposes and various usages of VMware's REDO file in conjunction with Virtual Disks. It begins with a description of the disk mode options and then quickly moves on to an explanation of how the REDO can be leveraged for backup purposes. It includes a restatement of some of the recommendations for the use of the REDO file from a performance and stability perspective.

**Disclaimers & Acknowledgements:**

This issue was brought to my attention by [Ron Oglesby](#) (of the Advanced Design Guide fame). The solution comes from forum user called [HaveBlue](#). I have used some of the comments from the original forum posts to assemble this information – and put it into a slightly easier format to read – and added some clarification and some tests here and there. If you want to read the original forum post it is [here](#)

I think it's important to state – that before Ron raised this issue and HaveBlue offered a fix – I was entirely in ignorance of the significance of this issue. All I have done is written a round-up of the issue.

# Using and Automating the Virtual Disk Shrink Feature

## The Issue

You might know that VMware Tools has a "shrink disks" feature. This is used to optimise the virtual disk file within the Guest OS - prior to exporting the disk either for:

- Transferring to another VMware product
- For use in ESX as a template
- For use in ESX as a backup – especially for those of you use vmbk.pl which uses the export feature together with REDO files to backup a virtual disk

Significantly, the shrink process can significantly reduce the space occupied by the virtual disks in an exported format – this can significantly reduce backup time and space taken up with this format of backup. The shrink takes place over two distinct phases "Shrink Preparation" (happens within Guest OS) and "Shrink Phase" (happens outside the OS). Shrinking does require SP4 on Windows 2000 to work without errors, there is a work around for those running SP3 or older. However, this can only be done from Remote Console – and unfortunately as of yet, VMware does not offer any tools to automate this process.

What happens during the shrink process that causes this saving of disk space?



Well, when Windows deletes a file, it doesn't actually delete the data in the disk; it just deletes the references/pointers in the file allocation table. So when ESX is exporting a VMDK and is looking at the raw disk, it's seeing values that aren't empty (non-zero), and exports them as such. The result is more disk space is used and takes longer to export the disk.



## The Evidence

If you want to see this phenomena create a new VMDK on a VM try this:

- Create a new virtual machine with a new virtual disk
- Use the VMware Tools to Shrink the disk
- Export the VMDK right after you have completed the Shrink process
- Fill the VMDK up with files and export again; it will be very big (most likely full size of VMDK)
- Now, delete all the files in the VMDK and export again what you will find is that is still big – even though you have deleted data from it.

I tried this test out a 10GB drive that had just the base Windows 2000 Install to it – and this what I found:

Test Type	Test Results
<b>Test1:</b> Size of Export Disk after the shrink	 <b>shrink.vmdk</b> rw - : 1526MB : Fri Nov 25 12:46:31 2005 4mins to export
<b>Test2:</b> Size of Export Disk after filling up disk and deleting the data – and emptying the recycle bin – but not using the shrink feature!	 <b>withoutshrink.vmdk</b> rw - : 9631MB : Fri Nov 25 14:25:36 2005 6mins to Export

<b>Test3:</b> Size of the Exported disk after re-running the Shrink Process	 <b>withshrink.vmdk</b> r w - : 1672MB : Fri Nov 25 4mins to export
<b>Test4:</b> Size of the Exported disk after running sdelete to emulate the shrink process	 <b>usingsdelete.vmdk</b> r w - : 1585MB : Fri Nov 25 17:14:39 2005 4mins to export

**Note:**

I know that there is a 146MG difference between shrink.vmdk and withshrink.vmdk. But compared to the difference between these files and withoutshrink.vmdk. I think the difference is insignificant. Obviously, there has been other changes taking place between Test3 and Test1. You might also notice that the "usingsdelete.vmdk" file is closer in size to "shrink.vmdk" than is the "withshrink.vmdk". These test compares VMware's shrink process to sdeletes and might suggest that sdelete is fractionally better. I don't think we can read too much in these slight differences. The wait times are not enormously different – but this was taken on a development server – with NO other activity on the ESX Host during the export time – this wouldn't be the case. They offered simply for your pleasure and amusement.

The key thing to get from the table above – is that if you have a lot read/write and read/delete events – then shrinking the disk invaluable in reducing the overall file size of the exported disk.

So, when you're doing an export, if you have the time, you'll get much smaller exported VMDK (and quicker export) if you shrink the VMDK first. What the 'shrink' does is to zero-fill the VMDK, this zero-filled content then gets ignored during export process. The way that the VMware Tools achieves this is to fill the virtual disk with zero filled files. While this does work, it's quite clunky, because if you stop the shrink process, it can leave behind all the zero-filled files!

It's good practise to separate disks where you would have amounts of R/W from disks where you get low amounts. This will allow you to "shrink" on the disks for which there is a maximum benefit. It's also best backup practise to separate System Disks (OS+Apps) from Data Disks anyway.

**The Solution**

The down side of VMware Tools "shrink" is that it cannot be scripted and an operator has to be there to perform the shrink prior to export. If you using the export feature as you backup method – you may need to find another way of doing the shrink that can be scripted. One way to accomplish this is using [Mark Russinovich's "SDelete"](#). This was originally designed to securely delete files in Windows. It can be re-compiled to do the same thing that the shrink options do in VMware Tools. I've used Mark's psexec in the past with my P2V Scripts and found his tools really cool.

**Link:** <http://www.sysinternals.com/Utilities/SDelete.html>

SDelete is a great utility. It basically random-fills a hard drive to delete data securely. However, until a new version is released, you'll have to modify and re-compile the version available on the SysInternals website to accomplish the 'shrink' functionality. The trick is to modify the SecureOverwrite function. Change the code so that the cleanBuffer value is not filled with any values (so by default, is zero-filled). Next, you want to modify the code so that only one pass is done (as we're not trying to securely over-write, so it just generates unnecessary I/O). Re-compile the source and voila! You now have a CLI utility that will quickly and cleanly zero out the hard drive space, and therefore be 'shrunk'...

This sounds more complicated than really is – it's in face a very simple process. The only catch with this utility is that right now, it's not "re-distributable" in this

modified state. You will need something like Microsoft Visual C++ 6.0 to re-compile the source code. Perhaps if Mark discovers that we VMware people are using his tool in this re-compiled way – he might modify the source to allow us to use this tool directly without any of the work below...

So to make your own “custom” version of sdelete – use the following procedure

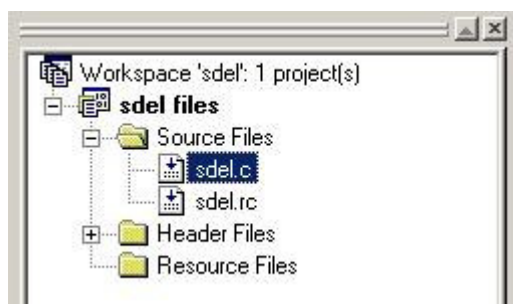
1. **Acquire a copy of Microsoft Visual C++ 6.0 and install it.** (I have no idea if it's possible to use an alternate compiler)
2. Go to <http://www.sysinternals.com/Utilities/SDelete.html> and download the program and source package
3. **Unzip the package,** and you get these files:



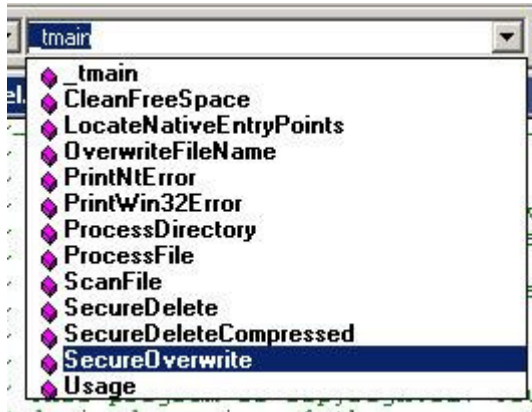
4. **Double-click on sdel.dsw** to load the project up in Microsoft Visual C++ 6.0
5. **On the left side of the screen is the workspace panel,** with 3 tabs at the bottom - click on the **FileView** tab



6. **Up at the top of the workspace panel,** expand the “**sdel files**” tree, then the “**Source Files**” tree, and **double-click on sdel.c**



7. **We could scroll through the source manually**, scanning for the SecureOverwrite() function but we can jump right to it with the **WizardBar C++ Members dropdown**:



8. **Scroll down a little way until you're looking at this section of code:**

```
for( i = 0; i < 3; i++ ) {
    cleanBuffer[i] = VirtualAlloc( NULL, CLEANBUFSIZE, MEM_COMMIT, PAGE_READWRITE );
    if( !cleanBuffer[i] ) {
        for( j = 0; j < i; j++ ) {
            VirtualFree( cleanBuffer[j], 0, MEM_RELEASE );
        }
        return FALSE;
    }
    //
    // Fill each buffer with a different signature
    //
    switch( i ) {
    case 0:
        // do nothing - buffer is zero-filled by Windows
        break;
    case 1:
        // fill with complement of 0 - 0xFF
        memset( cleanBuffer[i], 0xFF, CLEANBUFSIZE );
        break;
    case 2:
        // fill with a random value
        for( j = 0; j < CLEANBUFSIZE; j++ ) cleanBuffer[i][j] = (BYTE) rand();
        break;
    }
}
buffersAllocated = TRUE;
```

9. **What we need to do is comment out the switch statement so that the buffer always gets zero filled.** You might note that the preceding double slashes on a line designate a comment, but we can block out entire areas as a comment by using /\* to start a comment and \*/ to end it. So put a /\* just before the switch(i){ line and a \*/ just after the closing bracket for the switch so that the code looks like this:

```

    }
    return FALSE;
}
//
// Fill each buffer with a different signature
//
/*
switch( i ) {
case 0:
    // do nothing - buffer is zero-filled by Windows
    break;
case 1:
    // fill with complement of 0 - 0xFF
    memset( cleanBuffer[i], 0xFF, CLEANBUFSIZE );
    break;
case 2:
    // fill with a random value
    for( j = 0; j < CLEANBUFSIZE; j++ ) cleanBuffer[i][j] = (BYTE) rand();
    break;
}
*/
}
buffersAllocated = TRUE;

```

**Note:**

That the whole switch(i) block now appears in green, indicating that it is commented out.

10. **This step is optional, but recommended - jump down to the `_tmain()` function at the bottom of the file, and add a few lines to the banner to indicate that this is a modified version of the program and is no longer suitable for secure overwrite:**

```

int _tmain( int argc, TCHAR *argv[] )
{
    TCHAR        searchPattern[MAX_PATH];
    TCHAR        searchPath[MAX_PATH];
    PTCHAR       filePart;
    BOOL         foundFileArg = FALSE;
    int          i;

    //
    // Print banner and perform parameter check
    //
    _tprintf(_T("\nSDelete - Secure Delete v1.4\n"));
    _tprintf(_T("Copyright (C) 1999-2005 Mark Russinovich\n"));
    _tprintf(_T("Sysinternals - www.sysinternals.com\n\n"));
    _tprintf(_T("Modified by Mike Laverick - 2005\n\n"));
    _tprintf(_T("Used to Shrink VMware Virtual Disks\n\n"));
    _tprintf(_T("Secure Delete has been disabled\n\n"));
}

```

**Note:**

We have now finished modifying the source - it's time to compile!

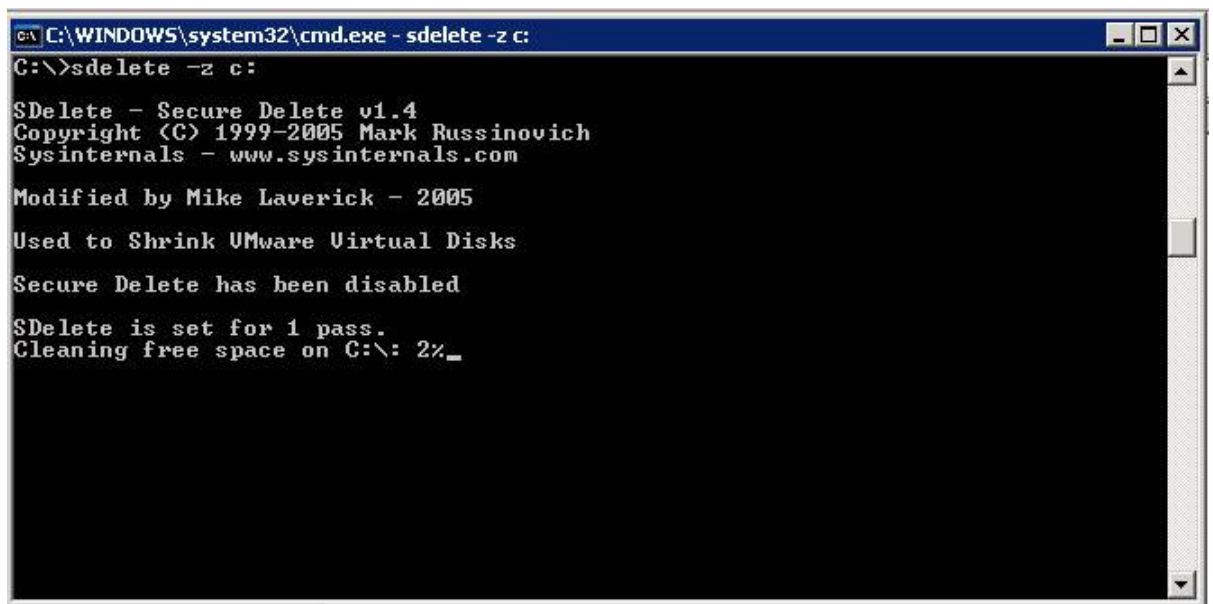
11. **Up at the top of the screen**, you'll see the **Build Mini-Bar** - click on the second icon on that bar (Build):



...and you'll see lines scrolling in the bottom Output window:



12. **Return back to the location that you had uncompressed the files to, and you'll see that there is now a Debug directory.** Have a look inside, and **you'll see your very own personalized copy of sdelete.exe**
13. **Usage is very simple** - fire up a command prompt, switch to the directory containing the executable, and invoke the sacred words **sdelete.exe -z c:** Naturally, replace c: with whatever drive you're clearing:



14. **To run the shrink** - type **sdelete -z c:**

**Warning:**

It's not a bad idea to first test this out on a non-production machine to ensure that there aren't any unwanted effects.

**Note:**

Once all is confirmed well, copy sdelete.exe to your target machine(s) and use the scheduler of your choice to run the program before automated snapshots are made. Naturally, the time it takes to clean a drive depends primarily on the free space available on the drive, so keep that in mind when scheduling.