


Why You Should Be Compacting Your Hyper-V Virtual Disks

 [Eric Siron](#)

 February 20, 2018

9 mins read



Table of contents

1. "Compact" or "Shrink" Hyper-V Virtual Hard Disks?

2. What Does "Compact" Mean for Hyper-V Virtual Hard Disks

3. Why Should I Compact Hyper-V Virtual Hard Disks?

4. NOT

5. How Do I Use PowerShell to Compact a VHDX?

6. What Do the Different VHDX Compact Modes Mean?

7. How Do I Use Hyper-V Manager to Compact a VHDX?

8. Can I Compact a VHDX that has Differencing Children?

9. What About VHDXs that Contain Non-Microsoft File Systems?

10. Why Does the Compact Operation Save Nothing or Very Little?

11. What Block Sizes are Valid for a Dynamically-Expanding VHDX?

12. How Can I Find a Dynamically-Expanding VHDX's Block Size?

13. What Block Size Should I Use?

Hyper-V’s dynamically expanding virtual hard disks (VHD/VHDX) provide numerous benefits. I suspect that most of us use them to achieve efficient use of physical storage. However, their convenience does come with a maintenance cost. They grow as needed, but they will never release any space unless you manually intervene (it could be scripted if you so desire). In this article, we’ll cover the reasons, prerequisites, and steps necessary to reclaim unused space from your dynamically-expanding VHDXs.

“Compact” or “Shrink” Hyper-V Virtual Hard Disks?

Sometimes definitions for “compact” and “shrink” get crossed when talking about VHDXs. They do not mean the same thing and you use completely different operations to achieve them.

Every [virtual hard disk](#) type (fixed, dynamically-expanding, and differencing) has a fixed upper size limit. “Shrinking” a VHDX reduces that limit. Any of the types can be shrunk. “Compacting” does not change the upper limit at all. Instead, it reduces the physical disk space consumed by a dynamically-expanding or differencing VHDX by removing empty blocks. I’ll explain what a “block” means to these VHDXs in the next section. You can run a compact operation on any of the types, but it will never have any effect on a fixed VHDX (the graphical interface screens will not even allow you to try). For more information on the shrink process, [refer to our earlier article on the issue](#).

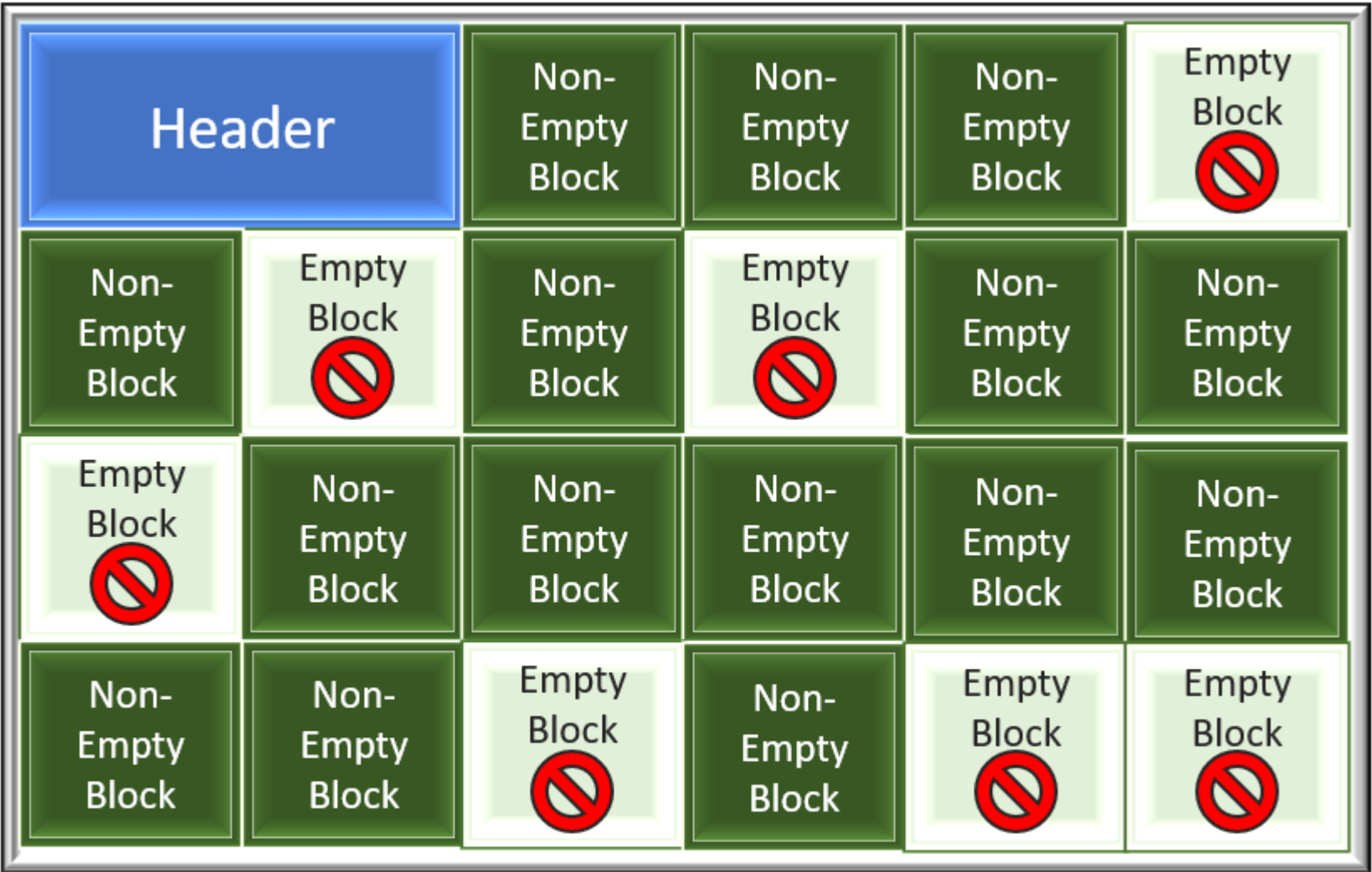
What Does “Compact” Mean for Hyper-V Virtual Hard Disks

Dynamically-expanding VHDXs function is a straightforward process. When created, each has a specific, permanently-set “block size”. When a guest operating system or some external action attempts to write information into the VHDX, the VHDX driver first checks to see if the file contains sufficient space in unclaimed blocks for that data. If it does, then the write proceeds. If it does not, the VHDX driver expands the physical file by the minimum number of blocks needed to contain the write.

However, when a process deletes a file (or uses some other technique to clear data), the VHDX driver does not do anything with the block. Most importantly, it does not automatically remove empty blocks from the file. I’ve read a fair number of complaints on that, but they lack merit. To maintain such a thing would be inherently dangerous and prohibitively expensive in terms of IOPS.



Instead, we manually initiate the *compact* process. The system scans the data region of the VHDX, looking for completely empty blocks. When it finds one, it removes it from the file. I’m not aware of the precise mechanism that it uses, but I suspect that it simply finds the next used block and shifts it backward, eventually lopping off the end of the file.



GuestOS.VHDX

Why Should I Compact Hyper-V Virtual Hard Disks?

Physical systems rarely use up all of their disk space. The same can be said for virtual systems. Therein lies the primary reason to use dynamically-expanding virtual hard disks. I especially like them for VHDXs that contain operating systems when you’ll be placing data on different VHDXs. I’ve got some VHDXs nearing five years of age that have a 60GB maximum but still use only around 25GB on disk. With many OS VHDXs, that quickly adds up to a great deal of saved storage. That’s only one example, though; I generally use dynamically-expanding VHDX in any case where I know it won’t cause problems (high-churn database systems would be a poor usage for them).

However, sometimes operations occur that consume an inordinate amount of disk space. Sometimes space is just used and there’s nothing you can do about it. Other times, the usage is only temporary. For example, you might copy several ISO files to the C: drive of a guest in order to perform a local software installation. If you’re doing in-place operating system upgrades, they’ll make a duplicate of the existing installation so that you can perform a rollback. A system may begin life with an oversized swap file that you later move to a different VHDX, leaving behind gigabytes of unused space. I always compact VHDXs with sysprepped operating systems that I will use as templates.

NOT

[“Premature optimization is the root of all evil.” — Donald Knuth](#)

I often complain about administrators that arbitrarily guess that their disk performance needs will be so great that anything other than fixed (or, egads, pass-through) is unacceptable. At the other end of the spectrum, some administrators became agitated when their VHDXs consume even a few gigabytes above their minimum. Most dynamically-expanding VHDXs will eventually find some sort of “normal state” in which they can re-use space from deleted data (example: Windows Updates).

I can’t give you a fine description of when a compact operation becomes viable, but do not use it to try to squeeze every last drop of space out of your system. You’ll eventually lose that battle one way or another. Do not compact over minor gains. Do not compact VHDXs that will never find any reasonable steady state (such as file servers). Do not compact as part of some routine maintenance cycle. If you have set up an automated script because your systems constantly approach the hard limits of your storage, spend the money to expand your storage. Compacting requires downtime and equipment wear, so it’s not free.

How Do I Use PowerShell to Compact a VHDX?

You can compact using GUI tools, but PowerShell provides the complete method. Remember two things:

- You cannot compact a VHDX connected to a running virtual machine.
- The computer system that runs the Optimize-VHD cmdlet **must** have the complete Hyper-V role installed. The PowerShell module alone does not contain the necessary system services. If you must run the compact operation on a system that does not have the complete Hyper-V, [you can use diskpart instead](#).



The process centers around the [Optimize-VHD cmdlet](#). In general, you’ll want to mount the VHDX into the [management operating system](#) to get the best results. For example:

```
Mount-VHD .dyndisk.vhdx -ReadOnly
Optimize-VHD .dyndisk.vhdx -Mode Full
Dismount-VHD .dyndisk.vhdx
```

Be aware that even for very small files, that can take some time to complete. The amount of time depends mostly on your available CPU cycle and the speed of your disk subsystem.

A PowerShell Tool for Compacting VHDX Files

You can easily convert the above three lines into a re-usable utility:

```
function Compact-VHD
{
    <#
    .SYNOPSIS
    Fully optimizes a Hyper-V VHD/X file.

    .DESCRIPTION
    Mounts the target VHD/X file as read-only, performs full optimization, and then dismounts it.

    .NOTES
    v1.0 January 28th, 2018
    (c) 2018 Eric Siron
    #>
    #requires -Module Hyper-V
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, ValueFromPipeline = $true)][String]$Path,
        [Parameter()][Microsoft.Vhd.PowerShell.VhdCompactMode]$Mode =
[Microsoft.Vhd.PowerShell.VhdCompactMode]::Full
    )
    try
    {
        $Path = (Resolve-Path -Path $Path -ErrorAction Stop).Path
        if ($Path -notmatch '.a?vhdx?$') { throw }
    }
    catch
    {
        throw('{0} is not a valid VHDX file.' -f $Path)
    }
    Mount-VHD -Path $Path -ReadOnly -ErrorAction Stop
    Optimize-VHD -Path $Path -Mode $Mode -ErrorAction Continue
    Dismount-VHD -Path $Path
}
```

You can [dot-source](#) that or add it to your [PowerShell profile](#) to use it anytime.

What Do the Different VHDX Compact Modes Mean?

If you read the help or tab through the options for the -Full parameter, you’ll see several choices. I feel like the built-in help describes it well enough, but more than a few people have gotten lost in terms with similar meanings. A bit of clarity:

- An **empty** block means a continuous length of space within the VHDX that its header has designated as a block and contains all zeros. A single 1 anywhere in the entire block means that the VHDX driver will treat it as used.
- An **unused** block may have some 1s, but the guest operating system’s file system has marked the space as not being used. This is common after file deletion; few operating systems will actually remove the bits from the disk. They simply mark it as unused in their file table(s). Two points on that:
 - At this time, the VHDX driver works most reliably with the NTFS file system. I have not yet tested for ReFS. It does not recognize any Linux file systems. In order for the system to best detect unused blocks, the VHDX must be mounted in read-only mode.
 - The VHDX driver also recognizes the [Trim and Unmap commands](#). That allows it to work with your hardware to determine what blocks are unused.

Hopefully, that helps smooth out explanations of the modes:

- **Full:** Full optimization requires the most time and resources. It will take out both *empty* and *unused* blocks. In order to detect unused blocks, the VHDX must be mounted in read-only mode. If the VHDX isn’t mounted, then it won’t be able to find *empty* blocks as easily (assume that it won’t find them at all).



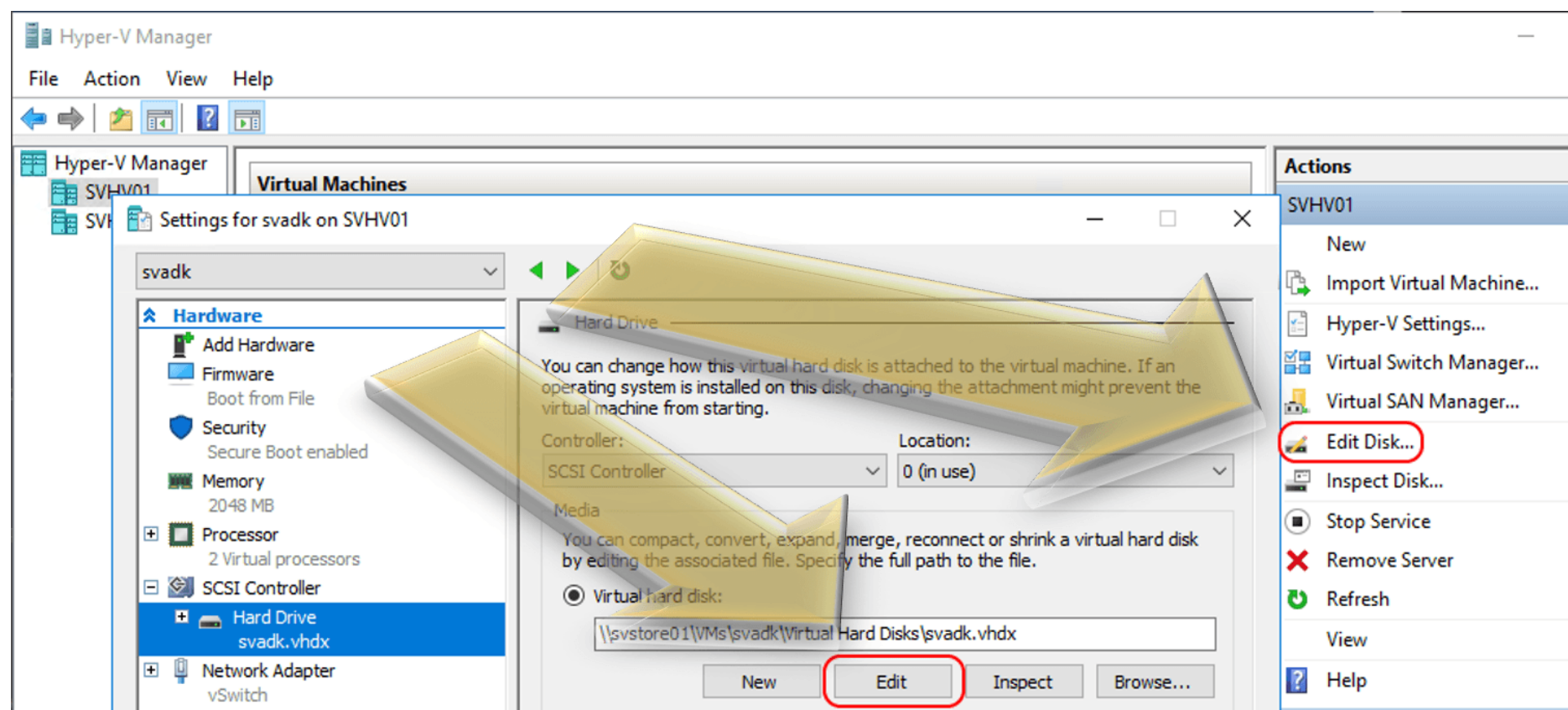
- **Quick:** The system only looks for *unused* blocks using the contained file system's metadata. If you don't mount the VHDX first, nothing will change.
- **Pretrimmed:** Utilizes information from the trim/unmap commands to detect *unused* blocks. Does not look for empty blocks and does not query the contained file system for unused blocks.
- **Prezeroed:** If the VHDX driver intercepted zero writes for existing blocks (such as from tools like [sdelete](#), then it has already recorded in its own tables that they're empty. Use this mode to remove only those blocks.
- **Retrim:** Retrim reads from the VHDX file's metadata for blocks marked as empty and reports them as trim or unmap commands to the underlying hardware.

Only the Full mode directly scans the file for empty blocks. The others work only with the VHDX and/or guest file system metadata.

How Do I Use Hyper-V Manager to Compact a VHDX?

Hyper-V Manager can compact a disconnected VHD/X or one attached to a powered-off virtual machine. It does not perform a very thorough job, though. I do not know which method it uses, but you'll discover very quickly that it is not the equivalent of PowerShell's "Full". Whatever technique it uses, it will not automatically mount the VHDX.

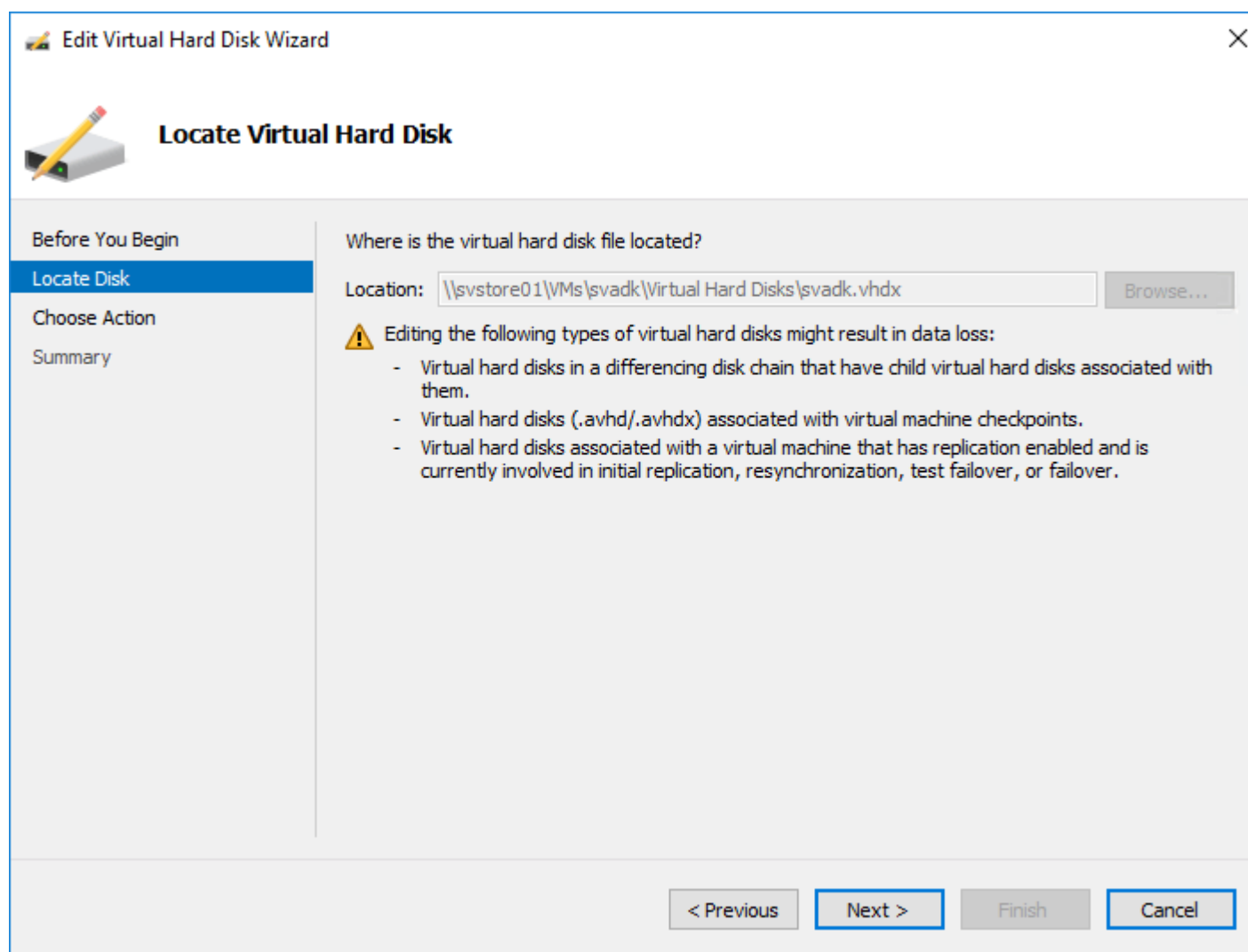
You can access the **Edit** function directly on a VM's disk property page or by going to the **Edit Disk** link in the main interface. Use the main interface for VHDXs that have no owning virtual machine.



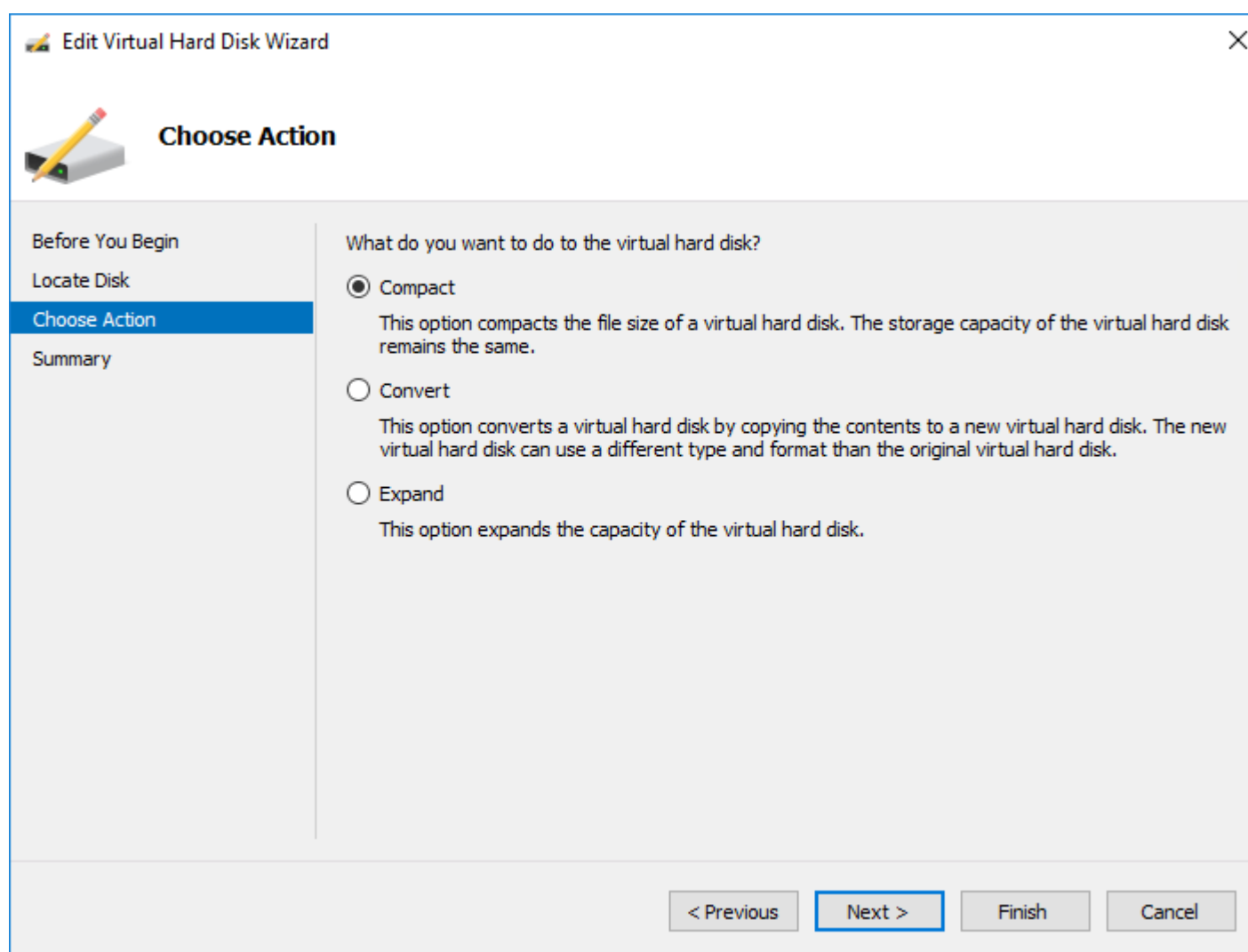
Both ways invoke the same wizard.

1. On the first screen, you select the target VHDX file. If you started from a VM's property page, then the **Location** line is filled in and grayed out.



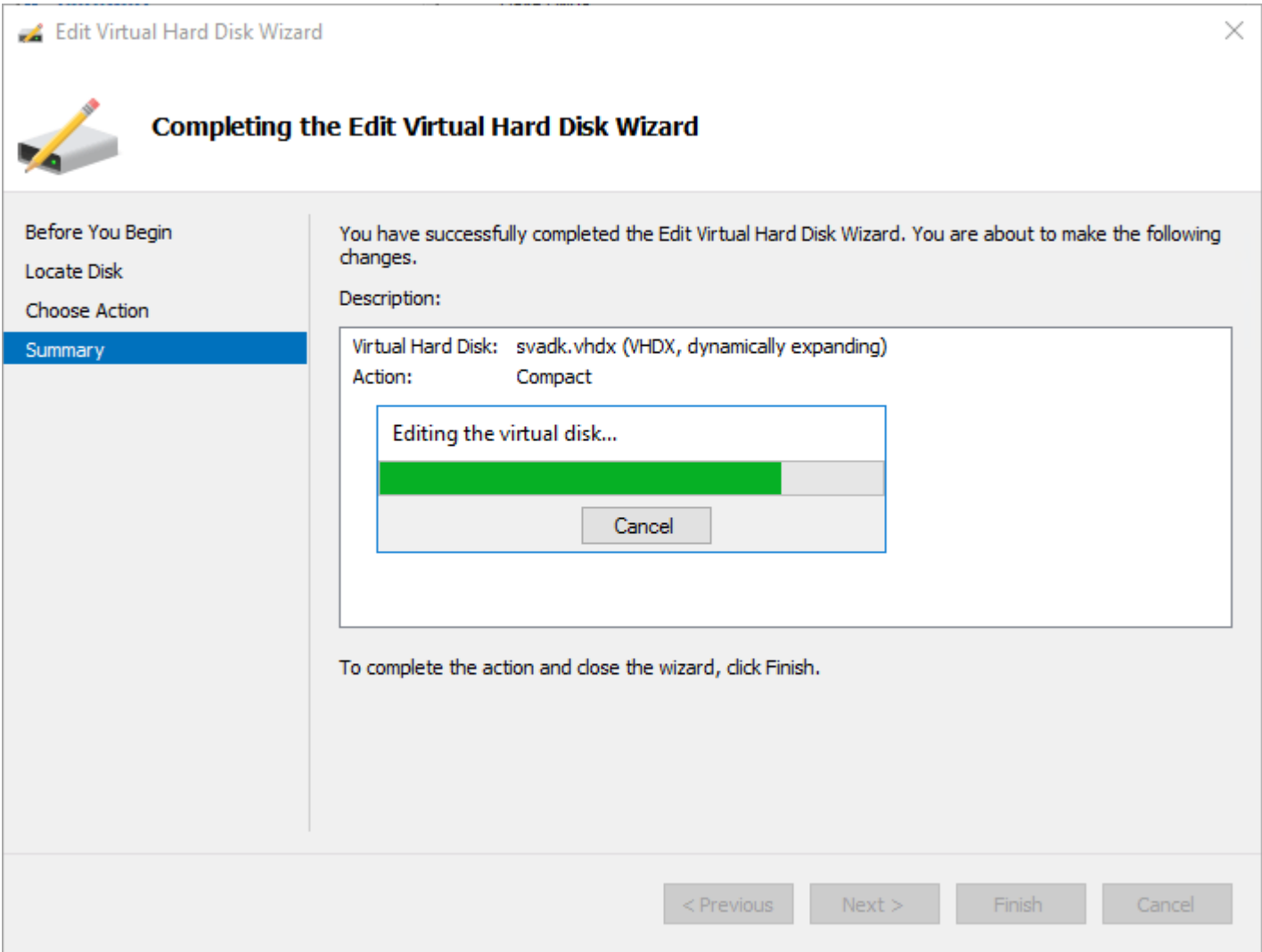


2. Next, the wizard will ask what you want to do with the disk. Choose **Compact**. If the Compact option does not appear, the virtual machine is turned on or you selected a fixed disk.



3. Click **Finish** on the final screen. You'll get a small dialog that shows the progress.





Can I Compact a VHDX that has Differencing Children?

The system will not prevent you from compacting a VHDX that has children unless one of the children is in use. If you manage to compact the parent VHDX, you will break the parent-child relationship.

What About VHDXs that Contain Non-Microsoft File Systems?

The Full mode will detect fully empty blocks, but cannot interact with any non-Microsoft file system metadata to locate unused blocks. However, most modern filesystems now support trim/unmap. If you can perform that operation within the guest operating system, you can then perform a Full optimize pass to shrink the VHDX. [I wrote an article on compacting VHDXs with Linux filesystems that have more information.](#)

Why Does the Compact Operation Save Nothing or Very Little?

The simple answer is that the blocks are not empty. If a block contains a single non-zero bit, then the entire block must be kept. You cannot see the VHDX's blocks from within the guest. Short of some sort of file scanner, I don't know of any way to view the blocks aside from a hex editor.

Some tips on cleaning up space:

- Delete as many files as possible
- Clear the guest's Recycle Bin(s)
- Use utilities that write zeros to free space
- Use a defragmentation tool that will consolidate free space

If you're really stuck, you can use a tool like robocopy or rsync to move the files into a completely new VHDX.

What Block Sizes are Valid for a Dynamically-Expanding VHDX?

By default, all VHDXs will have a 32 megabyte block size. You can only change that at creation time when using [New-VHD](#). Set the **BlockSizeBytes** parameter as desired. Unfortunately, they don't list possible values so it's a bit of trial and error. I tried 1MB, 8MB, 16MB, 64MB, and 256MB sizes with success, in addition to the default 32MB. 512 bytes failed, as did 512MB.

How Can I Find a Dynamically-Expanding VHDX's Block Size?

Only PowerShell's [Get-VHD](#) can show you the block size.



```
PS C:\LocalVMs\Virtual Hard Disks> Get-VHD .\blocktest.vhdx

ComputerName      : SVHV01
Path              : c:\localvms\virtual hard disks\blocktest.vhdx
VhdFormat         : VHDX
VhdType           : Dynamic
FileSize          : 4194304
Size              : 21474836480
MinimumSize       :
LogicalSectorSize : 512
PhysicalSectorSize : 4096
BlockSize         : 16777216
ParentPath        :
DiskIdentifier    : 3D5A8107-3447-4DF9-9851-4B4DB7886956
FragmentationPercentage : 0
Alignment         : 1
Attached          : False
DiskNumber        :
Number            :
```

What Block Size Should I Use?

You have a basic trade-off: small block sizes result in more writes with less unused space. Larger block sizes result in fewer writes with potentially more unused space.

With small block sizes, scattered small files will be less likely to result in a VHDX with a great many almost-but-not-quite empty blocks. For the various ext* filesystems found on Linux, a 1MB block size can greatly reduce your VHDX sizes. The greatest downside is that you might get a lot of fragmentation of the physical VHDX. Then again, you might not. It depends on how the writes occur. Infrequent small writes will result in more fragmentation. If you use robust, modern storage, then fragmentation is not usually a major concern.

Large block sizes don't have much to offer. If you were to store a rarely-accessed archival database in a dynamically-expanding VHDX with block sizes that match your guest OS's allocation unit size, you might, in theory, see something improve. Probably not, though.

I recommend that you stick to the default 32mb block size for any guest that doesn't use an ext filesystem. Use 1MB block sizes for those, if you remember.

Was this helpful? Yes No [Provide feedback about this article](#)

Share this post



Not a DOJO Member yet?

Join thousands of other IT pros and receive a weekly roundup email with the latest content & updates!

JOIN NOW, IT'S FREE!

