

**FREE
PREVIEW
CHAPTER**

COMPREHENSIVE AND AFFORDABLE Guide Series!

Bonus Material at www.rationalpress.com

The Rational Guide To

Scripting Microsoft[®] Virtual Server 2005

Anil Desai
Microsoft MVP
Windows Servers

In this guide you'll learn how to...

- ✓ *Manage Virtual Server 2005 using VBScript, VB.NET, and C#*
- ✓ *Understand the architecture of Virtual Server 2005*
- ✓ *Automate complex virtual environments*



*Rational Guides for a
Fast-Paced World™*



Mike Sterling
Program Manager
Windows Virtualization
Microsoft Corporation

**To order this book, visit
www.RationalPress.com**

PUBLISHED BY

Rational Press - An imprint of the Mann Publishing Group

710 Main Street, 6th Floor

PO Box 580

Rollinsford, NH 03869, USA

www.rationalpress.com

www.mannpublishing.com

+1 (603) 601-0325

Copyright © 2006 by Mann Publishing Group.

All rights reserved. No part of the contents of this book may be reproduced in any form or by any means without the written permission of the publisher. For questions about rights and permissions, send e-mail to permissions@mannpublishing.com.

ISBN: 1-932577-29-7

Library of Congress Control Number (LCCN): 2006923187

Printed and bound in the United States of America.

10 9 8 7 6 5 4 3 2 1

Trademarks

Mann Publishing, Mann Publishing Group, Agility Press, Rational Press, Inc.Press, NetImpress, Farmhouse Press, BookMann Press, The Rational Guide To, Rational Guides, ExecuGuide, AdminExpert, From the Source, the Mann Publishing Group logo, the Agility Press logo, the Rational Press logo, the Inc.Press logo, Timely Business Books, Rational Guides for a Fast-Paced World, and Custom Corporate Publications are all trademarks or registered trademarks of Mann Publishing Incorporated.

All brand names, product names, and technologies presented in this book are trademarks or registered trademarks of their respective holders.

Disclaimer of Warranty

While the publisher and author(s) have taken care to ensure accuracy of the contents of this book, they make no representation or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties or merchantability or fitness for a specific purpose. The advice, strategies, or steps contained herein may not be suitable for your situation. You should consult with a professional where appropriate before utilizing the advice, strategies, or steps contained herein. Neither the publisher nor author(s) shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Credits

Author:	Anil Desai
Technical Editor:	Mike Sterling
Editorial Director:	Jeff Edman
Book Layout:	Molly Barnaby
Index:	Christine Frank
Series Concept:	Anthony T. Mann
Cover Concept:	Marcelo Paiva

All Mann Publishing Group books may be purchased at bulk discounts.

Book Contents



Part I - SCRIPTING AND AUTOMATION BASICS

Chapter 1 - The Benefits of Virtualization

Chapter 2 - Virtual Server Architecture & Requirements

Chapter 3 - Automating Virtual Server with VBScript

Chapter 4 - Automating Virtual Server with .NET

Chapter 5 - Monitoring Virtualization Using WMI

Part II - AUTOMATING VIRTUAL MACHINE MANAGEMENT

Chapter 6 - Managing Virtual Machines

Chapter 7 - Managing Virtual Hard Disks

Chapter 8 - Managing Virtual Networks

Part III - ADVANCED AUTOMATION

Chapter 9 - Using Scripts, Events, and Tasks

Chapter 10 - Automating VMRC

Chapter 7

Managing Virtual Hard Disks

The content of hard disks defines what a computer—physical or virtual—does. Hard disks contain the operating system, applications, and data. That includes pretty much everything that persists between uses of the virtual machine. On physical computers, it's difficult to add and remove hard disks. Usually, the local storage is tied to the physical computer, and some manual labor is required to switch out these components. Through the use of virtualization, on the other hand, a few mouse clicks or a couple of lines of code are all that's required to attach and detach virtual hard disk files.

One of the most flexible and useful aspects of working with virtual machines is the power and flexibility of virtual disk architecture provided by Virtual Server. Figure 7.1 provides an introduction to the various types of objects that make up this architecture. Starting from the left, a virtual machine can contain one or more disk controllers. Each of the disk controllers can support the attachment of a virtual hard disk. Each of the virtual hard disks can refer to a physical file on the host's physical file system.

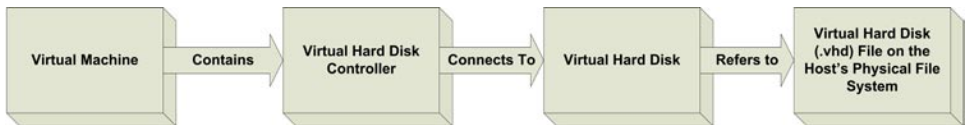


Figure 7.1: Overview of the Components of Virtual Server Hard Disk Architecture.

Figure 7.2 provides an overview of the major disk-related objects in the Virtual Server COM API.

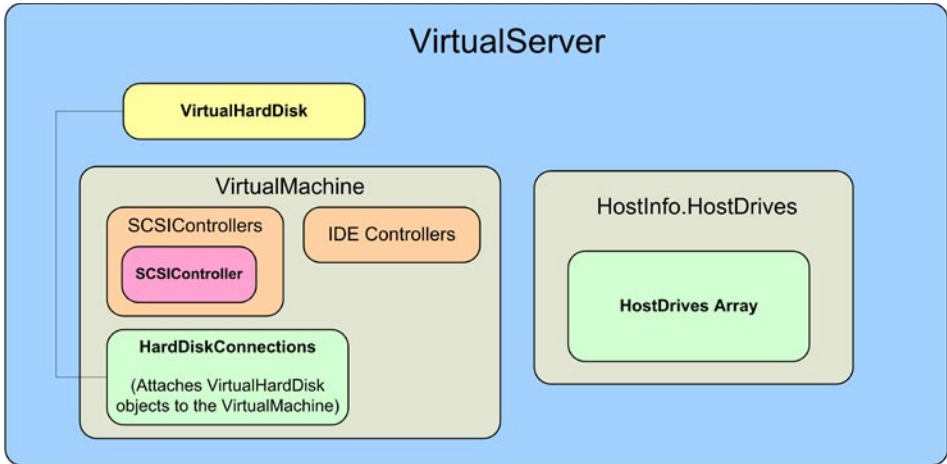


Figure 7.2: Overview of the Disk-Related Objects in the Virtual Server's COM API.

In this chapter, we'll look at the details related to working with virtual hard disks. We'll start by examining how virtual hard disks work in Virtual Server. Then we'll move on to details related to automating the creation, attachment, and maintenance of VHD files.

Disk Architecture Overview

On a Virtual Server host computer, a Virtual Hard Disk (VHD) is simply a file that resides on the host's file system. These files can then be attached to a virtual machine (through an IDE or SCSI-based disk controller). From within the virtual machine, all disk I/O operations seem to be occurring as if they were on a physical disk. This isn't limited to just standard file operations. It's perfectly possible to format, partition, and defragment a virtual hard disk from within a VM. Since they are just files, virtual hard disks can be easily moved, copied, and backed up.

From a conceptual standpoint, working with virtual hard disks is fairly simple. Whenever the virtual machine makes a read or write request to a virtual hard disk, Virtual Server intercepts that call. It then remaps the request to the contents of the VHD file on the host's file system, performs the operation, and returns the results to the virtual machine. Figure 7.3 shows this process in action. Virtual hard disks can be created within either the Virtual Server Administration Web Site or programmatically (as we'll see later in this chapter).

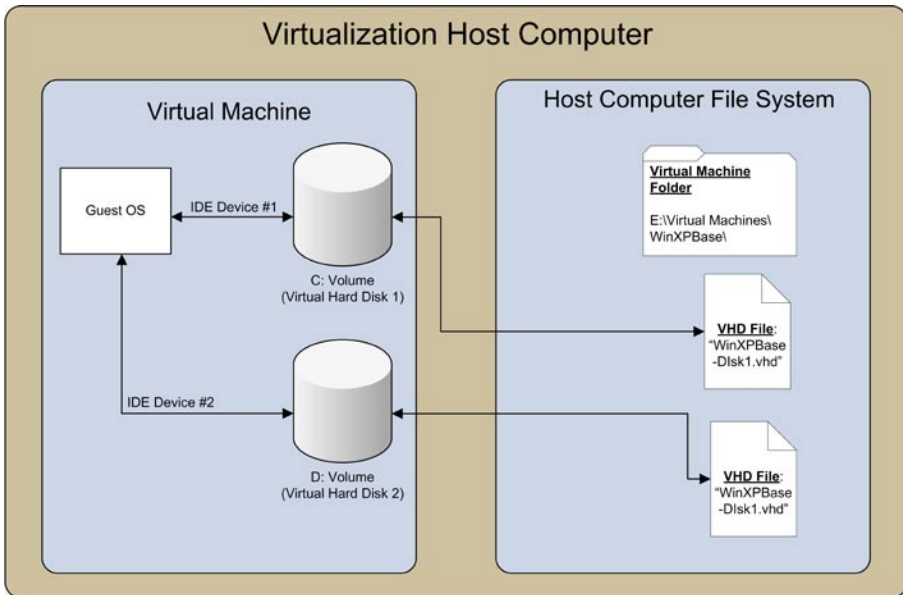


Figure 7.3: How Virtual Hard Disks and VHD Files Work.



Note:

VHD files can also be accessed over a network connection or over a SAN. For more information about storage options based on SAN, NAS, iSCSI and other network storage methods, see *The Rational Guide To Managing Microsoft® Virtual Server 2005*, available from www.rationalpress.com.

Virtual Hard Disk Types

There are several main considerations to keep in mind about virtual hard disks. On one hand, Virtual Server administrators want to minimize the amount of disk space that's taken up by their virtual machines. Although hard disk storage options have become inexpensive, supporting multiple virtual machines can require a lot of storage capacity. The other major issue is performance. Many server applications and technologies rely heavily upon disk operations. File servers, database servers, and line-of-business applications can generate a tremendous amount of I/O. From a virtualization standpoint—where multiple, independent virtual machines are competing for disk resources—optimizing overall performance is very important.

To address the space- and performance-management trade-offs, Virtual Server supports the creation of several different types of virtual hard disks. These types are:

- ▶ **Dynamically-Expanding VHDs** — Dynamically-expanding VHDs use physical disk space on the host only when it is required by the VM. When you create this type of VHD, you specify the maximum size to which the file can grow (the default in the Virtual Server Administration Web Site is 16 GB). From within the VM, the hard disk appears to be a full 16-GB physical disk (and it can be formatted and partitioned as needed). Dynamically-expanding VHDs start with a very small physical file size on the host computer's file system. They then grow, as the guest operating system starts requesting more disk space (see Figure 7.4). Because of the simplicity of creating dynamically-expanding VHDs, they are often recommended as a good first option for virtual machine installations. Note, however, that if the host computer runs out of disk space, the virtual machines that are using these disks could lose data and/or crash.

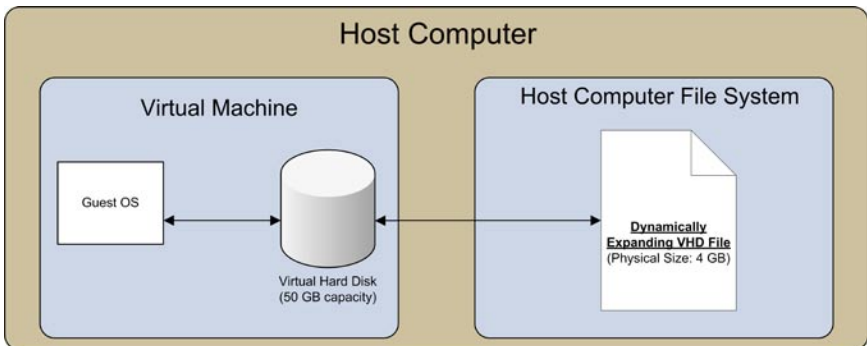


Figure 7.4: How Dynamically-Expanding VHDs Work.

- ▶ **Fixed-Size VHDs** — As their name suggests, fixed-size VHDs take up the maximum amount of space specified at the time of their creation. If you create a new “blank” VHD of size 16 GB, you will see a 16-GB file on the host's operating system (see Figure 7.5). There are a couple of benefits of fixed-size hard disks. First, you don't have to worry about running out of disk space—since it's already allocated, there's no way that another VM

or host process can fill up the disk. The other benefit is that fixed-size disks perform better since they don't have to be expanded when more disk space is needed. This makes them an excellent choice for disk-intensive virtual machines (such as virtualized file servers).

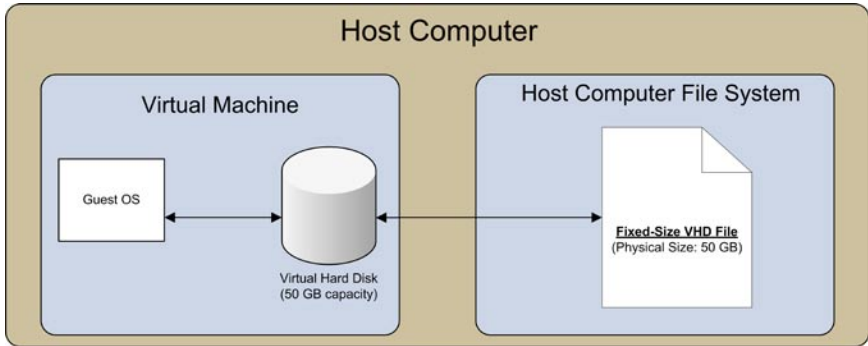


Figure 7.5: How Fixed-Size VHDs Work.

- ▶ **Linked Hard Disks** — Linked hard disks function as pointers to an entire physical volume on the host. Although a very small file is created when you created a new virtual hard disk, all read and write operations are actually performed on the host's file system itself (see Figure 7.6). Note that an entire physical hard disk on the host computer (which might include multiple logical volumes) is the target of the link. Linked hard disks were created for one primary purpose: to help convert physical hard disks to virtual ones. This supports the migration of a physical server to a virtual server. When you convert a linked hard disk into a fixed-size or dynamically-expanding VHD, all of the contents of the physical volume will be copied as well. Once you create a linked hard disk, you should be careful to avoid using it from within the host operating system, as these changes could cause problems within the VM. To provide a layer of protection, a linked hard disk can be made read-only.

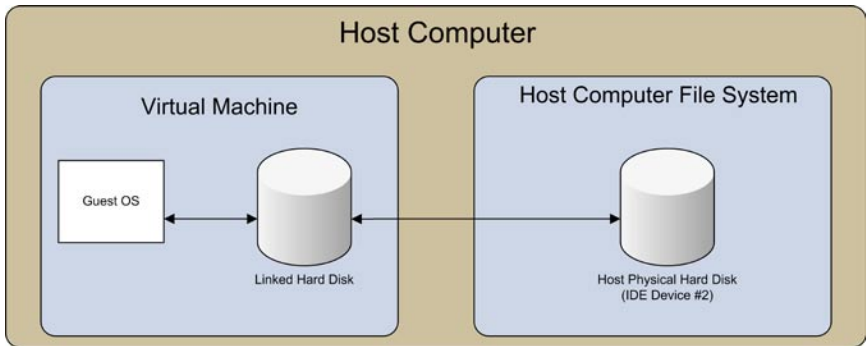


Figure 7.6: How Linked Hard Disks Work.



Caution:

Newcomers to the world of virtualization tend to worry that disk-related operations (such as deleting files or formatting a hard disk) might affect the host operating system. In almost all cases, this is not a concern—all changes with a VHD will only affect the VHD files on the host computer. There's one important exception, however: linked hard disks actually point to an entire physical disk on the host computer. In this case, all file operations that are performed on those volumes will be performed on the actual host file system. Use linked hard disks with care!

- ▶ **Differencing Disks** — Differencing disks are based on either fixed-size or dynamically-expanding VHDs. They allow you to roll back changes to a specific virtual hard disk by recording all write operations to a separate file. Differencing disks can also be used to create a parent-child hierarchy of VHDs.

In most cases, fixed-size VHDs will perform better than dynamically-expanding VHDs, since the latter can result in a large amount of fragmentation on the host's file system. The result is that significantly more disk I/O operations might be required to satisfy the VM's read and write requests. Some of this overhead can be reduced through defragmentation operations and shrinking the VHDs. Fixed-size disks do not have these performance-related problems, but the trade-off is that they can consume a very large amount of disk

space, even when VMs are using them. If this all seems too complicated, here's a piece of welcome news: you can easily convert between fixed-size and dynamic hard disks (we'll look at the details later in this chapter).

While it's technically not a specific virtual hard disk type that you can create and manage like the other types, undo files (which we covered in Chapter 6) are also available for use with Virtual Server. On the surface, differencing disks might appear to serve the same purpose as undo disks. However, there are several differences. For further details about differencing disks, see *The Rational Guide To Managing Microsoft® Virtual Server 2005*, available from www.rationalpress.com. Overall, using differencing disks is a powerful option that can help you adapt to many different types of virtualization scenarios.

Working with Virtual Hard Disks

When you create a new virtual machine using the Administration Web Site, you're given the option to create a new virtual hard disk or add an existing one. This makes sense, since a VM really can't do much without a VHD. However, when you create a new VM programmatically (as we did in Chapter 6), you need to separately create and attach the necessary files.



Note:

There's a difference in naming between the Virtual Server COM object model and the Virtual Server Administration Web Site. The type of hard disk commonly known as "linked disks" is referred to in the object model as "HostDisks." For all practical purposes, the terms are interchangeable.

Getting Host Disk Configuration Details

Before we get into the details of creating and attaching VHDs to virtual machines, let's look at how you can view details about the host computer's disk configuration. The information in this section will allow you to determine how many hard disks are available on the host computer, as well as their sizes.

The place to start is the **HostInfo** object of the Virtual Server instance. Table 7.1 provides a summary of the useful properties and methods.



HostInfo Member	Member Type	Purpose
.HostDrives	Property	Returns an array of host drive names.
.GetHostDriveSize	Method	Returns the size of a host drive in MB. This method takes the HostDrives property as an argument.
.IsHostHardDriveMounted	Method	Returns True if the host hard disk is mounted (that is, it is partitioned, formatted, and ready for use).

Table 7.1: Disk-Related Properties of the HostInfo Object.

Creating Virtual Hard Disks

Programmatically creating new virtual hard disks can be accomplished directly using the **VirtualServer** object. The following methods can all be used to create new VHDs. The variable *imagePath* is the fully-qualified path and filename for the new VHD file.

```
objVirtualServer.CreateDynamicVirtualHardDisk(imagePath, size)
```

When creating a dynamically-expanding virtual hard disk, the *size* parameter pertains to the maximum size of the hard disk (in megabytes). The amount of physical disk space used initially will be very small.

```
objVirtualServer.CreateFixedVirtualHardDisk(imagePath, size)
```

When creating a fixed-size virtual hard disk, the *size* parameter pertains to the total size of the hard disk file to create on the host. Note that for very large file sizes, this operation can take several minutes.

```
objVirtualServer.CreateDifferencingVirtualHardDisk(imagePath, parentPath)
```

The **parentPath** is the fully-qualified path and filename to the base virtual hard disk type. The differencing disk will have a small physical size initially, and will automatically grow as needed.

```
objVirtualServer.CreateHostDriveVirtualHardDisk(imagePath,
hostDriveIdentifier, mountReadOnly)
```

This command creates a linked hard disk. The **hostDriveIdentifier** is a **HostDrive** object (a property of the **HostInfo** object covered earlier in this chapter). The **mountReadOnly** argument is a Boolean value that specifies whether the linked hard disk should be read-only (true) or read-write (false) within the VM.

Getting a Virtual Hard Disk Object

Throughout the remainder of this chapter, we'll look at ways to work with VHD files. Many of these methods and properties require that you first have a **HardDisk** object. There are several ways in which you can get this reference. If you know the full path and file name of the VHD file, you can use the following:

```
objVirtualMachine.GetHardDisk(imagePath)
```

The only argument, *imagePath*, is simply the fully-qualified path and filename. This method will return a **HardDisk** object, and you'll be ready to go.

But what if you don't have the filename and path of the VHD (or if that's exactly the information you're looking for)? The **VirtualServer** object has you covered there, as well. The following command will search for VHD files:

```
objVirtualMachine.GetHardDiskFiles(AdditionalSearchPaths)
```

By default, this command will walk through all of the known search paths that are configured for the Virtual Server instance and return an array of strings specifying the full pathname to those files. The argument *AdditionalSearchPaths* is optional, and can be used to specify additional paths in which the command should search. Note that, by default, Virtual Server will automatically look in subdirectories of the ones specified.

Getting VHD Details

Once you have access to a **HardDisk** object, you'll be able to get information about the file itself. The properties and their meanings are listed in Table 7.2.

Property	Data Type	Purpose
File	String	Contains the fully-qualified filename and path to the VHD file.
HostDrivIdentifier	String	For linked hard disks, this property returns information about the physical hard disk on the host computer to which the VHD is linked. This property does not apply to other VHD types.
HostDiskFreeSpace	Decimal	Returns the total number of megabytes that are available on the host hard volume on which this VHD resides.
Parent	HardDisk object	For differencing hard disks, this property returns a hard disk object of the parent disk. This property does not apply to other VHD types.
Security	Security object	Returns an object that can be used to query details related to the permissions set on the VHD file.

Table 7.2: Useful Hard Disk-Related Properties.



Property	Data Type	Purpose
SizeInGuest	Decimal	Returns the size (in MB) of the disk, as viewed within the guest operating system. This is equal to the maximum size specified when creating a fixed-size or dynamically-expanding VHD. For fixed-size virtual hard disks, this will be the same as the .SizeOnHost value.
SizeOnHost	Decimal	Returns the physical file size of the VHD file on the host. For dynamically-expanding disks, this size will be less than or equal to the maximum file size specified upon creation. For fixed-size disks, this will be the same as the .SizeInGuest property.
Type	HardDiskType (enumeration)	Returns the type of the VHD file. The enumeration members include: vmDiskType_Dynamic = 0, vmDiskType_FixedSize = 1, vmDiskType_Differencing = 2, vmDiskType_HostDrive = 4

Table 7.2: Useful Hard Disk-Related Properties (continued).

All of this information can be very helpful for managing virtual hard disks. For example, you might want to periodically compare the size of a VHD file on the host against the amount of available free space on the host. If you're running low on disk space, you might need to move the virtual hard disk to another logical volume on the host.

Maintaining VHDs

Virtual hard disks are the real workhorses of virtual machines, since they're the primary method of maintaining the persistence of data. Like physical hard disks, virtual hard disks require some basic maintenance in order to keep them working optimally. In this section, we'll look at these operations (and how easy it is to perform them using scripts and programs).



Caution:

All of the maintenance operations mentioned in this section are potentially very resource-intensive. They can put large amounts of strain on the CPU and disk I/O subsystems, in particular. For this reason, it's highly recommended that you reserve these maintenance operations for off-peak times (an excellent job for automation). Also, various operations might require a lot of temporary disk space (up to twice the size of the VHD that you're working with, in some cases).



Converting VHDs

As mentioned earlier in this chapter, Virtual Server allows you to convert fixed-size VHDs to dynamically-expanding VHDs, and vice versa. This process can be helpful in a number of ways. The simplest example is if you just changed your mind: at first, saving disk space was a higher priority, but now you need the performance of fixed-size hard disks. Other situations include disk-intensive processes (such as installing the guest operating system). You might want to start with a fixed-size disk to get the performance benefits, and then convert it to a dynamically-expanding one later. You can also convert a linked hard disk to either of the other two types of base hard disk types.

The syntax for performing a conversion is as follows:

```
objHardDisk.Convert(convertedDiskImagePath, convertedDiskImageType)
```

Where:

- ▶ **convertedDiskImagePath** — This is the fully-qualified path and filename for the new VHD that will be created. The term “conversion” might be a little misleading, as the actual operation creates a new VHD for you and keeps the old one. Of course, you can manually delete the old VHD after the conversion process is complete.
- ▶ **convertedDiskImageType** — This is the type of VHD you want to create (fixed-size or dynamically-expanding). The possible values are of the **HardDiskType** enumeration (covered earlier in this chapter).

Compacting VHDs

When you delete a significant amount of data from a dynamically-expanding virtual hard disk, the size of the physical VHD file on the host’s file system does not automatically decrease in size (that’s why they’re not also called “dynamically-shrinking” files). That’s where the process of compacting a VHD file comes in. The compacting process will remove all of the unused space from the physical disk file and will reduce the amount of space used on the host’s physical disk. It’s important to note that fixed-size VHDs cannot directly be compacted. They must first be converted to dynamically-expanding hard disks in order to perform this operation.



Virtual Server also includes a Virtual Disk Precompactor utility that should be run from within the virtual machine before you perform a compact operation. For more details, see the Virtual Server user's guide or release notes.

The process of compacting a VHD couldn't be much easier. All you need to do is execute the **.Compact** method of the appropriate virtual hard disk object:

```
objHardDisk.Compact()
```

While it's not directly related to compacting VHDs (and there's no direct automated way to accomplish this), it's a good idea to routinely defragment all of your guest file systems, as well as your host file system. Refer to the guest operating system's documentation for information on how to do this.

Merging Differencing Disks

The process of merging a differencing disk can be used to create a single disk out of a child and a parent VHD. Figure 7.7 provides an overview of the two different ways in which this can be accomplished.

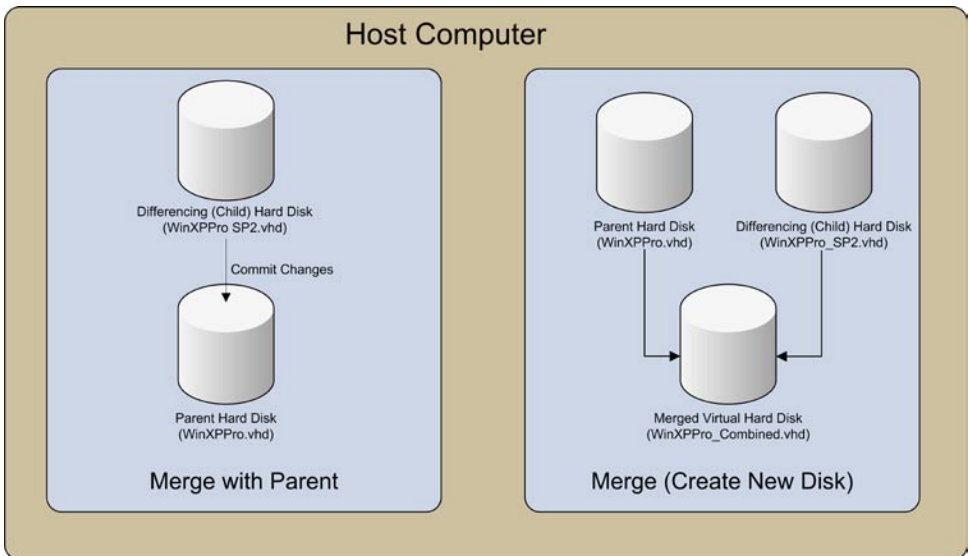


Figure 7.7: Merging a Differencing Disk to a New Disk, or with its Parent.

You can programmatically merge a differencing disk with its parent using one of two different methods. To copy the contents of the base hard disk type to the differencing disk, you can use the **Merge** method:

```
objHardDisk.Merge()
```

Alternatively, if you want to create a new file that contains the combined content of both the differencing disk and its parent, you can use the following method:

```
objHardDisk.MergeTo(newDiskImagePath, newDiskImageType)
```

Where:

- ▶ **newDiskImagePath** — This is the fully-qualified path and filename of the new VHD that will be created.
- ▶ **newDiskImageType** — This is the type of VHD you want to create (fixed-size or dynamically-expanding). The possible values are of the `HardDiskType` enumeration (covered earlier in this chapter).



Undo “disks” are actually merged through the process of committing. For information about committing undo disks, see Chapter 6.

Code Samples: Maintaining Virtual Hard Disks

The following code samples cover some steps related to managing and maintaining virtual hard disk files. Specifically, the following tasks are performed:

1. Creation of a new dynamically-expanding virtual hard disk with a maximum size of 500 MB.
2. Compacting the Virtual Hard Disk.
3. Converting the dynamically-expanding VHD to a fixed-size VHD.



```

On Error Resume Next 'Continue on errors

VHDFilename = "C:\VirtualServer\VHD\NewHardDisk.vhd"
Set objVirtualServer = CreateObject("VirtualServer.Application")

'Create a new dynamically-expanding VHD
objVirtualServer.CreateDynamicVirtualHardDisk VHDFilename, 500
WScript.Sleep (10000) 'Pause 10 seconds for the hard disk to be created

'Compact the new VHD
Set objHardDisk = objVirtualServer.GetHardDisk(VHDFilename)
objHardDisk.Compact()

'Convert the new VHD to a fixed-size VHD
objHardDisk.Convert "C:\VirtualServer\VHD\FixedSizeDisk.vhd", 1

WScript.Echo "Operation complete."

```

Listing 7.1: Maintaining Virtual Hard Disks Using VBScript.

```

Imports Microsoft.VirtualServer.Interop

Namespace ScriptingVirtualServer
    Public Class Listing_07_02
        Public Shared Function MaintainVHD() As String
            Dim VHDFilename As String = "C:\VirtualServer\VHD\NewHardDisk.vhd"
            Dim objVirtualServer As New VMVirtualServer

            'Create a new dynamically-expanding VHD
            objVirtualServer.CreateDynamicVirtualHardDisk(VHDFilename, 500)
            System.Threading.Thread.Sleep(10000) 'Pause for VHD creation

            'Compact the new VHD
            Dim objHardDisk As VMHardDisk = objVirtualServer.
                GetHardDisk(VHDFilename)
            objHardDisk.Compact()

```



```

        'Convert the new VHD to a fixed-size VHD
        objHardDisk.Convert("C:\VirtualServer\VHD\FixedSizeDisk.vhd",
➤ VMHardDiskType.vmDiskType_FixedSize)

        Return "Operation complete."
    End Function
End Class
End Namespace

```

Listing 7.2: Maintaining Virtual Hard Disks Using VB.NET.

```

using Microsoft.VirtualServer.Interop;

namespace VirtualServerSampleCode_CSharp
{
    namespace ScriptingVirtualServer
    {
        public class Listing_07_03
        {
            public static string MaintainVHD()
            {
                string VHDFilename = @"C:\VirtualServer\VHD\NewHardDisk.vhd";
                VMVirtualServer objVirtualServer = new VMVirtualServer();

                //Create a new dynamically-expanding VHD
                objVirtualServer.CreateDynamicVirtualHardDisk(VHDFilename,
➤ 500);

                System.Threading.Thread.Sleep(10000); //Pause for VHD creation

                //Compact the new VHD
                VMHardDisk objHardDisk = objVirtualServer.GetHardDisk(
➤ VHDFilename);

                objHardDisk.Compact();
            }
        }
    }
}

```



```

        //Convert the new VHD to a fixed-size VHD
        objHardDisk.Convert(@"C:\VirtualServer\VHD\FixedSizeDisk.vhd",
        VMHardDiskType.vmDiskType_FixedSize);

        return "Operation complete.";
    }
}
}
}
}

```

Listing 7.3: Maintaining Virtual Hard Disks Using C#.

Managing Hard Disk Attachments

So far, we have focused on the topics of creating new virtual hard disks, and the operations required to maintain them. What we haven't yet covered is the important topic of putting VHDs to use by attaching them to a VM. The basic relationship between the objects is that a virtual machine can contain several hard disk controllers, each of which can connect to one or more virtual hard disks. In this section, we'll look at how all of these object types come together.

Comparing SCSI and IDE Controllers

Virtual Server supports two different types of virtual hard disk controllers: IDE and SCSI. It's important to note that these options are completely independent of the underlying host hard disk controller architecture. For example, you can have virtual IDE hard disks on a server that has only SCSI connections, and you can create virtual SCSI disks on a computer that has only physical IDE connections. Other disk controller types such as Serial ATA (SATA) or Serial-Attached SCSI (SAS) will work as well.

Table 7.3 compares the features of SCSI and IDE hard disk connections. The main benefit of IDE connections is that they have the widest support in guest operating systems. Although they may not be officially supported, this means that you can run all the old "classics," like older versions of Linux and MS-DOS, without requiring the installation of any special drivers. The virtual CD/DVD-ROM device is attached to one of the IDE controller channels, leaving up to three more available for use.



Feature	IDE	SCSI
Maximum number of controllers per VM	2	4
Maximum number of channels per controller (usable)	2	7
Total number of connected devices	4	28
Maximum size of a connected VHD file	127 GB	2.0 TB (~2000 GB)
Supports multiple concurrent IO Operations?	No	Yes
Supports virtual CD/DVD-ROM Devices	Yes	No

Table 7.3: Comparison of IDE and SCSI Controller Features.

SCSI-based virtual hard disk connections provide several advantages. First, since the SCSI architecture allows multiple concurrent I/O operations to execute on the bus at the same time, it has a performance advantage. On virtual machines where disk I/O is a bottleneck, the improvement could be up to 20%. The second advantage is that up to four SCSI devices (each with 7 usable channels) can be added to a VM. This means that up to 28 total VHDs can be attached to the SCSI controllers alone.

Since VHD files are independent of the type of hard disk controller they're connected to, you can move a VHD between IDE and SCSI controllers as often as you wish. There's only one potential exception: if you will be moving the boot partition of an operating system between two connections, you must be sure that both IDE and SCSI drives are available in order for this to work.

Tech Tip:

If you're planning to move a virtual machine between Virtual Server and Virtual PC, you should know that Virtual PC does not provide support for SCSI hard disk connections. One potential workaround for this problem is to install on Virtual Server, and include both IDE and SCSI devices in a guest virtual machine. During installation, this will ensure that both IDE and SCSI drivers are available in the guest OS.

Managing SCSI Controllers

As with physical machines, in order to add disk controllers or hard disk connections, a virtual machine must be turned off. The two types of supported disk controllers—IDE and SCSI—are managed in different ways in Virtual Server. Since all virtual machines will always have two IDE controllers, there's no need for methods to enumerate, add, or remove them. There will always be two IDE controllers (numbered 0 and 1), and each controller will have two channels (called Primary and Secondary).

The situation with SCSI adapters is different. By default, new virtual machines will not have any SCSI adapters at all. To enumerate the SCSI controllers that are currently attached to a VM, you can use the following property:

```
objVirtualMachine.SCSIControllers()
```

This property returns a collection of **SCSIController** objects (if any). There are two main configuration options for SCSI controllers. This first specifies whether the SCSI bus is shared (this is generally enabled to allow for clustering support). The second option is to specify the SCSI ID of the controller itself. SCSI controllers have eight total channels (numbered 0 through 7). The SCSI controller itself uses a channel, which may be either 6 or 7 (the default). Figure 7.8 provides an example of creating managing SCSI controllers using the Virtual Server Administration Web Site.

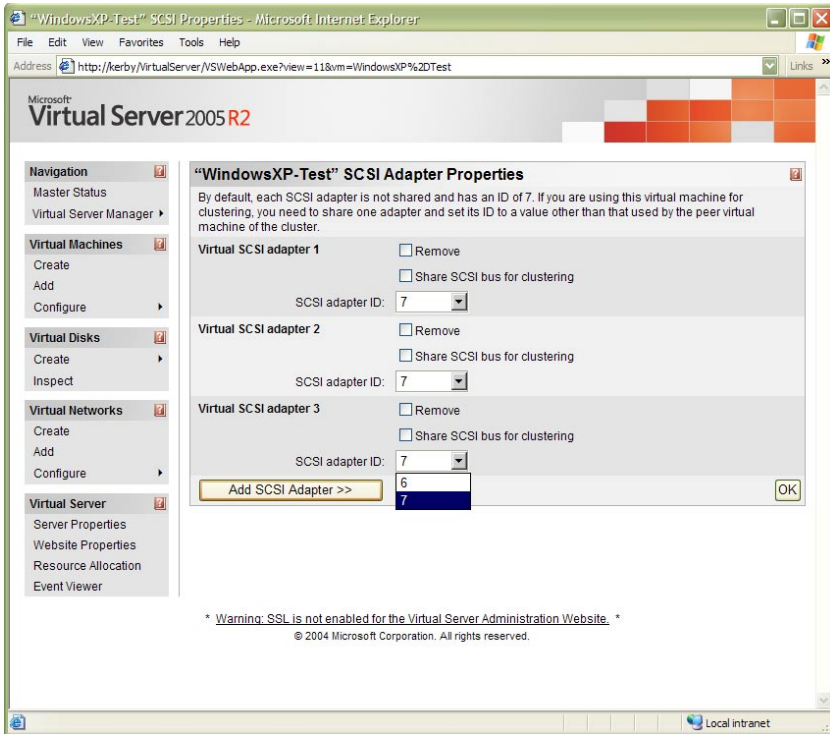


Figure 7.8: Managing SCSI Controllers Using the Virtual Server Administration Web Site.

To configure a SCSI controller, you can use the following method:

```
objSCSIController.Configure(isBusShared, SCSIID)
```

The *isBusShared* argument is a Boolean value that specifies whether or not the SCSI bus should be shared (more information about the purpose of this is available in the *Virtual Server Administrator's Guide*). The second option is the ID of the SCSI adapter itself (“6” or “7”). To obtain this information about the **SCSIController** object, you can use the following two properties:

```
objSCSIController.IsBusSharedobjSCSiController.SCSIID
```

To add a new SCSI controller to the virtual machine, use the following command:

```
objVirtualMachine.AddSCSIController()
```

Note that the command does not take any parameters. Virtual Server will automatically add a new SCSI controller to the virtual machine (or create the first one, if one doesn't yet exist).

Removing a SCSI controller can be done by passing a **SCSIController** object to the appropriate method:

```
objVirtualMachine.RemoveSCSIController(SCSIControllerObject)
```

Managing Hard Disk Connections

Now that we've covered information about virtual hard disks and hard disk controllers, it's time to link the two. A hard disk connection links a **HardDisk** object (which is actually a VHD file) to a hard disk controller. On a physical computer, you can visualize this as a standard IDE or SCSI connection cable that runs between a hard disk drive and the controller port to which it is attached. For virtual machines, you can add and remove connections as long as the VM is powered off.

The **.HardDiskConnections** property of the **VirtualMachine** object returns a collection of **HardDiskConnection** objects. The API command is:

```
objVirtualMachine.HardDiskConnections()
```

You can loop through the returned collection of **HardDiskConnection** objects to get details about which VHDs are connected to the VM, and to which controller they're attached. Table 7.4 provides a listing of the properties of a **HardDiskConnection** object.

HardDiskConnection Property	Data Type	Purpose
BusType	busType (enumeration)	The DriveBusType enumeration value for the connection. The enumeration contains the following three values: vmDriveBusType_Invalid = -1, vmDriveBusType_IDE = 0, vmDriveBusType_SCSI = 1
BusNumber	Long	The number of the hard disk controller. If you are using an IDE bus type, then the value can be either 0 or 1 (since only two IDE controllers are allowed). If you attaching to a SCSI bus type, the value can be 0, 1, 2, or 3 (since up to four SCSI controllers are allowed).
DeviceNumber	Long	The device number on the specified bus. For IDE devices, this can be either 0 or 1 (specifying the primary or secondary channel on the IDE controller). For SCSI devices, this can be 0 through 6 (specifying to which of the seven channels the device will be attached).
HardDisk	HardDisk object	Returns a HardDisk object representing a connected hard disk (if any).
UndoHardDisk	HardDisk object	Returns a HardDisk object representing an undo disk (if any).

Table 7.4: Properties of the HardDiskConnection Object.

To add a new hard disk connection, use the following command:

```
objVirtualMachine.AddHardDiskConnection(hardDiskPath, busType, busNumber,
deviceNumber)
```

Where:

- ▶ **hardDiskPath** — The fully-qualified path and file name for the VHD file you want to attach.
- ▶ **busType** — Specifies whether the connection will be to an IDE bus type (0) or a SCSI bus type (1).
- ▶ **busNumber** — Specifies the number of the controller. For IDE devices, this number can be either 0 (for the primary channel) or 1 (for the secondary channel). For SCSI devices, the number 0 represents the first controller, 1 represents the second, etc.
- ▶ **deviceNumber** — Once you've specified the busNumber, you can specify to which channel on the bus you want to add the new VHD. For IDE devices, the options are 0 or 1. For SCSI devices, the options are 0 through 6 (assuming that the SCSI controller itself is using ID 7).

Once the command executes, a new virtual hard disk will be available for use by the guest operating system. Most modern operating systems will automatically detect the new hard disk during the boot process. If the hard disk is already properly formatted and has data on it, it should be ready for use immediately. Otherwise, you might need to partition and/or format the hard disk.

If you want to change the bus location of an existing **HardDiskConnection** you can use the following command:

```
objHardDiskConnection.SetBusLocation(busType, busNumber, deviceNumber)
```

Removing a hard disk connection requires you to get a **HardDiskConnection** object and then pass it to the appropriate **VirtualMachine** method, as follows:

```
objVirtualMachine.RemoveHardDiskConnection(HardDiskConnectionObject)
```

Code Samples: Managing Hard Disk Connections

The code samples in this section walk through the process of creating a VHD and then attaching it a new VM's hard disk controllers. Specifically, the code automates the following operations:

1. Create a new VM called "Test VM."
2. Creating two dynamically-expanding virtual hard disks. Each VHD will be a fixed-size disk with a size of 50 MB.
3. Attach the first VHD to the first IDE channel.
4. Create a new SCSI hard disk controller for the VM.
5. Attach the second VHD to the first SCSI controller.

```
On Error Resume Next 'Continue on errors
```

```
VMFolderPath = "C:\VirtualServer\VHDTTest"
```

```
Set objVirtualServer = CreateObject("VirtualServer.Application")
```

```
'Create a new VM
```

```
Call objVirtualServer.CreateVirtualMachine("VHD Test VM", VMFolderPath)
```



```
Set objVirtualMachine = objVirtualServer.FindVirtualMachine("VHD Test VM")
```

```
'Create two new Virtual Hard Disks
```

```
VHD01FileName = VMFolderPath & "\Disk01.vhd"
```

```
objVirtualServer.CreateFixedVirtualHardDisk VHD01FileName, 50
```

```
WScript.Sleep (5000) 'Pause for five seconds
```

```
VHD02FileName = VMFolderPath & "\Disk02.vhd"
```

```
objVirtualServer.CreateFixedVirtualHardDisk VHD02FileName, 50
```

```
WScript.Sleep (5000) 'Pause for five seconds
```

```
'Attach Disk01 to the the first IDE channel
```

```
Call objVirtualMachine.AddHardDiskConnection (VHD01FileName, 0, 0, 0)
```

```
'Add a SCSI Controller to the VM
```

```
objVirtualMachine.AddScsiController
```

```
'Attach Disk02 to the first available SCSI channel
```

```
Call objVirtualMachine.AddHardDiskConnection (VHD02FileName, 1, 0, 0)
```

```
WScript.Echo "Operation complete."
```

Listing 7.4: Maintaining Virtual Hard Disk Connections Using VBScript.

```
Imports Microsoft.VirtualServer.Interop
```

```
Namespace ScriptingVirtualServer
```

```
    Public Class Listing_07_05
```

```
        Public Shared Function ConnectVHD() As String
```

```
            Dim VMFolderPath As String = "C:\VirtualServer\VHDDTest"
```

```
            Dim objVirtualServer As New VMVirtualServer
```

```
            'Create a new VM
```

```
            objVirtualServer.CreateVirtualMachine("VHD Test VM", VMFolderPath)
```

```
            Dim objVirtualMachine As VMVirtualMachine
```

```
            objVirtualMachine = objVirtualServer.FindVirtualMachine("VHD
```

```
Test VM")
```



```

'Create two new Virtual Hard Disks
Dim VHD01FileName As String = VMFolderPath & "\Disk01.vhd"
objVirtualServer.CreateFixedVirtualHardDisk(VHD01FileName, 50)
System.Threading.Thread.Sleep(5000) 'Wait five seconds

Dim VHD02FileName As String = VMFolderPath & "\Disk02.vhd"
objVirtualServer.CreateFixedVirtualHardDisk(VHD02FileName, 50)
System.Threading.Thread.Sleep(5000) 'Wait five seconds

'Attach Disk01 to the first IDE channel
objVirtualMachine.AddHardDiskConnection(VHD01FileName,
☛VMDriveBusType.vmDriveBusType_IDE, 0, 0)

'Add a SCSI adapter to the new virtual machine
objVirtualMachine.AddSCSIController()

'Attach Disk02 to the first available SCSI channel
objVirtualMachine.AddHardDiskConnection(VHD02FileName,
☛VMDriveBusType.vmDriveBusType_SCSI, 0, 0)

Return "Operation complete."
End Function
End Class
End Namespace

```

Listing 7.5: Maintaining Virtual Hard Disk Connections Using VB.NET.

```

using Microsoft.VirtualServer.Interop;

namespace VirtualServerSampleCode_CSharp
{
    namespace ScriptingVirtualServer
    {
        public class Listing_07_06
        {

```

```

public static string MaintainVHD()
{
    string VMFolderPath = @"C:\VirtualServer\VHDDTest";
    VMVirtualServer objVirtualServer = new VMVirtualServer();

    //Create a new VM
    objVirtualServer.CreateVirtualMachine("VHD Test VM", VMFolderPath);
    VMVirtualMachine objVirtualMachine = objVirtualServer.
    FindVirtualMachine("VHD Test VM");

    //Create two new Virtual Hard Disks
    string VHD01FileName = VMFolderPath + @"\Disk01.vhd";
    objVirtualServer.CreateFixedVirtualHardDisk(VHD01FileName,50);
    System.Threading.Thread.Sleep(5000); //Wait five seconds

    string VHD02FileName = VMFolderPath + @"\
    Disk02.vhd";
    objVirtualServer.CreateFixedVirtualHardDisk(VHD02FileName,50);
    System.Threading.Thread.Sleep(5000); //Wait five seconds

    //Attach Disk01 to the first IDE channel
    objVirtualMachine.AddHardDiskConnection(VHD01FileName,
    VMDriveBusType.vmDriveBusType_IDE, 0, 0);

    //Add a SCSI adapter to the new virtual machine
    objVirtualMachine.AddSCSIController();

    //Attach Disk02 to the first available SCSI channel
    objVirtualMachine.AddHardDiskConnection(VHD02FileName,
    VMDriveBusType.vmDriveBusType_SCSI,0,0);

```



```
        return "Operation complete.";
    }
}
}
```

Listing 7.6: Maintaining Virtual Hard Disk Connections Using C#.

Summary

In this chapter we looked at one of the most flexible and powerful aspects of working with VMs—the virtual disk architecture. We started with an overview of the different types of VHDs, including fixed-size, dynamically-expanding, linked, and differencing disks. Then we looked at the tasks of converting and compacting VHDs to maintain performance and minimize wasted disk space. We brought this information together by looking at how you can programmatically attach VHDs to virtual machines. All of these concepts and methods can help you build complex VM configurations with just a few lines of code.

Compliments of...

mann
PUBLISHING GROUP



www.mannpublishing.com