Virtual Disk API Programming Guide

VMware Virtual Disk Development Kit 1.1

vmware®

Virtual Disk API Programming Guide
Item: EN-000056-03

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *Virtual Disk API Programming Guide* introduces the Virtual Disk Development Kit and describes how to develop software using the VMware® virtual disk library, which provides a set of system-call style interfaces for managing virtual disks.

To view the current version of this book as well as all VMware API and SDK documentation, go to http://www.vmware.com/support/pubs/sdk_pubs.html.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this guide.

Table 1. Revision History

Revision	Description
20090203	Fourth version for release 1.1 beta2 of the Virtual Disk Development Kit for storage partners.
20080731	Third version for release 1.1 beta of the Virtual Disk Development Kit for storage partners.
20080411	Second version for release 1.0 of the Virtual Disk Development Kit.
20080131	First version of the Virtual Disk Development Kit for partner beta release.

Intended Audience

This guide is intended for developers who are creating applications that manage virtual storage. It assumes knowledge of C and C++ programming.

Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to docfeedback@vmware.com.

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current versions of other VMware books, go to http://www.vmware.com/support/pubs.

Online and Telephone Support

To use online support to submit technical support requests, view your product and contract information, and register your products, go to http://communities.vmware.com/community/developer.

Support Offerings

To find out how VMware support offerings can help meet your business needs, go to http://www.vmware.com/support/services.

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides offerings to help you assess, plan, build, and manage your virtual environment. To access information about education classes, certification programs, and consulting services, go to http://www.vmware.com/services.

Introduction to the Virtual Disk API

1

This chapter introduces VMware virtual disk management and the Virtual Disk Development Kit.

Virtual Disk Management

The Virtual Disk API, or VixDiskLib, is a set of function calls to manipulate virtual disk files in VMDK format (virtual machine disk). Function call semantics are patterned after C system calls for file I/O. This API enables partners and software vendors to manage VMDK directly from their applications.

These library functions can manipulate virtual disk on a VMware Workstation or similar product (hosted disk) or virtual disk contained within a vStorage VMFS volume on an ESX/ESXi server (managed disk). Hosted disk is an VMware term meaning a disk managed by the Workstation host for a guest operating system.

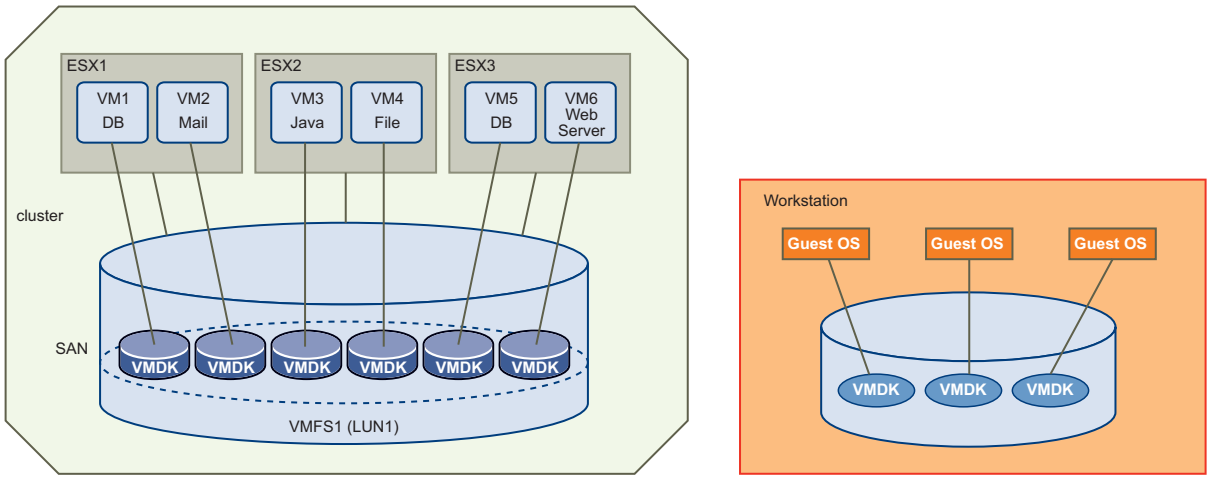
What is Managed Disk?

VMDK format dates back to the early days of VMware Workstation. Virtual machine disk files represent the storage volumes of a virtual machine, and are named with `.vmdk` suffix. On a VMware Workstation host, file systems of each guest OS are kept in VMDK files on the host's physical disk drive.

With the virtual machine file system (VMFS) on ESX/ESXi hosts, VMDK files represent the disk volumes of virtual machines. This is called managed disk. Managed disk is the same thing as `VMFS_FLAT` virtual disk, presented in "Disk Types" on page 15. Functions in the Virtual Disk API support vStorage VMFS, with some exceptions as noted for managed disk.

Table 1-1 compares the arrangement of managed disk (in this case VMDK on a SAN-hosted VMFS file system) and hosted disk (VMDK files on physical disk).

Figure 1-1. Managed Disk and Hosted Disk



VMFS disk can reside on a SAN (storage area network) attached to ESX/ESXi hosts by Fibre Channel or iSCSI. It can also reside on network attached storage, and on directly attached disk. In all cases, the ESX/ESXi host manages physical disk. The Virtual Disk API has no facility to address a storage partition directly. For storage planning, see the whitepaper *VMware Virtual Machine File System: Technical Overview and Best Practices* in the Resources section of the VMware Web site. Follow the configuration advice of your storage vendor.

Virtual Disk Development Kit

The Virtual Disk Development Kit includes the following components:

- Virtual Disk API library functions
- VMware disk utilities: disk mount and virtual disk manager
- Documentation for the above components

Virtual Disk Management Utilities

The Virtual Disk Development Kit includes two command-line utilities for managing virtual disk: disk mount and virtual disk manager. The virtual disk manager is included with Workstation 6.0.x and Server products. Disk mount is available in the Virtual Disk Development Kit and in upcoming products.

Disk Mount Utility

VMware disk mount (`vmware-mount`) is a utility for Windows and Linux hosts. If a virtual disk is not in use, the utility mounts it as an independent disk volume, so it can be examined outside its original virtual machine. You can also mount specific volumes of a virtual disk if the virtual disk is partitioned.

Disk mount is useful because the Virtual Disk API contains no function for making a mounted partition available to other processes. Opening a VMDK is like mounting, but for the calling process only.

See the *VMware DiskMount User's Guide*, which is available on the Web and in the kit.

Virtual Disk Manager Utility

VMware virtual disk manager (`vmware-vdiskmanager`) is a command-line utility for Windows and Linux hosts. It allows you to create, convert, expand, defragment, shrink, and rename virtual disk files. It does not have a facility to create redo logs or snapshots.

See the *VMware Virtual Disk Manager User's Guide*, which is available on the Web and in the kit.

Virtual Disk API

VMware provides graphical tools and command-line utilities to help administrators manage virtual disk. Customers have asked for programmatic interfaces also. This is a standalone wrapper library that helps you develop solutions that integrate into a wide range of VMware products. The Virtual Disk API partly duplicates functionality of the virtual disk management utilities and has additional capabilities:

- It permits random read/write access to data anywhere in a VMDK file.
- It creates and manages redo logs (parent-child disk chaining, or delta links).
- It can read and write disk metadata.
- It is able delete VMDK files programmatically.
- Error explanations are available.
- Many operations are easier to automate with an API than with utilities.

For Windows, the virtual disk kernel-mode driver is 32-bit or 64-bit depending on the underlying system. The user-mode libraries are 32-bit because Windows On Windows 64 can run 32-bit programs without alteration. For Linux, both 32-bit and 64-bit user-mode libraries are provided.

VMware vSphere API to Read and Write VMDK

Version 2.5 of the VMware vSphere API contains some experimental methods to manage VMDK files. See the managed object type `VirtualDiskManager`, which contains about a dozen methods similar to those in the Virtual Disk API documented here.

If you are interested, navigate to VMware SDKs on the Web and click VMware vSphere API Reference Guide for the API 2.5 version. Find `VirtualDiskManager` under All Types.

Virtual Disk Internal Format

A document detailing the VMware virtual disk format is available on request. Navigate to VMware Interfaces Web page, click the **Request** link, and provide your name, organization, and email address. A link to the online PDF document should arrive shortly in your email inbox.

http://www.vmware.com/interfaces/vmdk.html

This *Virtual Disk Format 1.0* document provides useful information about the VMDK format. It uses the term "delta link" for what this manual calls "redo log" or "child" disk.

Solutions Enabled by the Virtual Disk API

When integrated into applications, the Virtual Disk API allows you to manipulate virtual disk images and provide support for VMDK format.

Some tasks can be accomplished either by the virtual disk management utility or by the API:

- Create a new set of new virtual disks and prepare to provision applications.
- Create disk templates for fresh system install, or patch updates, by the IT department.
- Back up a particular volume, or all volumes, associated with a virtual machine.
- Clone the VMDK of a virtual machine and use the cloned copy to perform offline maintenance.
- Manipulate virtual disks to defragment, expand, rename, or shrink the underlying file system image.
- Convert a virtual disk to another format, for example from hosted disk to managed disk.
- Convert a physical disk to a virtual disk (P2V), or a virtual disk to a physical disk (V2P).
- Migrate virtual disks on demand to enable user workforce mobility.

Some solutions can be developed more easily with the Virtual Disk API than with the utilities:

- Scan a VMDK for virus signatures, either live, or first cloning it for off-line scanning. It is not necessary for the antivirus scanner to have knowledge of the underlying file system.
- Search for data in virtual disks across multiple virtual machines.
- Perform data recovery from unresponsive or corrupt virtual machines.
- Verify the integrity of a VMDK and possibly repair the file system image.
- Optimize VMDK images by combining and compacting them.
- Write defragmentation tools that operate on the native file system, not only on 2GB extents.
- Create VMDK saves by backing up the child, compacting the image, and creating a new child.
- Make a plug-in for a forensic analysis tool such as the X-Ways product.
- Develop a tool like VDK, an open-source kernel mode driver that opens (mounts) a VMDK for read-write access on a Windows drive letter.
- Extend VMDK for additional OS support, for example mount capability in BSD.
- Create disk support tools to assist hardware vendors.

Virtual Disk Library Functions

Table 1-1 alphabetically lists function calls in the Virtual Disk API.

Table 1-1. Virtual Disk API Functions

Function	Description
VixDiskLib_Attach	Attaches the child disk chain to the parent disk chain.
VixDiskLib_Cleanup	Remove leftover transports. See "Clean Up After Disconnect" on page 40.
VixDiskLib_Clone	Copies virtual disk to some destination, converting formats as appropriate.
VixDiskLib_Close	Closes an open virtual disk.
VixDiskLib_Connect	Connects to the virtual disk library to obtain services.
VixDiskLib_ConnectEx	Connects to optimum transport. See "Connect to VMware vSphere" on page 40
VixDiskLib_Create	Creates a virtual disk according to specified parameters.
VixDiskLib_CreateChild	Creates a child disk (redo log or delta link) for a hosted virtual disk.
VixDiskLib_Defragment	Defragments a virtual disk.
VixDiskLib_Disconnect	Disconnects from the virtual disk library.
VixDiskLib_Exit	Releases all resources held by the library.
VixDiskLib_FreeErrorText	Frees the message buffer allocated by GetErrorText.
VixDiskLib_FreeInfo	Frees the memory allocated by GetInfo.
VixDiskLib_GetErrorText	Returns the text description of a library error code.
VixDiskLib_GetInfo	Retrieves information about a virtual disk.
VixDiskLib_GetMetadataKeys	Retrieves all keys in the metadata of a virtual disk.
VixDiskLib_GetTransportMode	Gets current transport mode. See "Get Selected Transport Method" on page 40
VixDiskLib_Grow	Grows an existing virtual disk.
VixDiskLib_Init	Initializes the virtual disk library.
VixDiskLib_ListTransportModes	Available transport modes. See "List Available Transport Methods" on page 39.
VixDiskLib_Open	Opens a virtual disk.
VixDiskLib_Read	Reads a range of sectors from an open virtual disk.
VixDiskLib_ReadMetadata	Retrieves the value of a given key from disk metadata.
VixDiskLib_Rename	Renames a virtual disk.
VixDiskLib_Shrink	Reclaims blocks of zeroes from the virtual disk.
VixDiskLib_SpaceNeededForClone	Computes the space required to clone a virtual disk, in bytes.
VixDiskLib_Unlink	Deletes the specified virtual disk.
VixDiskLib_Write	Writes a range of sectors to an open virtual disk.
VixDiskLib_WriteMetadata	Updates virtual disk metadata with the given key/value pair.

Installing the Virtual Disk Development Kit 2

This chapter covers the prerequisites for and installation of the Virtual Disk Development Kit.

Packaging and Components

The Virtual Disk Development Kit is packaged like other VMware software as a compressed archive for Linux, or an executable installer for Windows. It includes the following components:

- Command-line utilities `vmware-mount` and `vmware-vdiskmanager` in the `bin` directory.
- Header files `vixDiskLib.h` and `vm_basic_types.h` in the `include` directory.
- Function library `vixDiskLib.lib` (Windows) or `libvixDiskLib.so` (Linux) in the `lib` directory.
- HTML reference documentation and sample program in the `doc` directory.

Supported Platforms

You can install the Virtual Disk Development Kit on the following platforms:

- Windows, both 32-bit x86 and 64-bit x86-64 processors:
  - Windows XP (Service Pack 2)
  - Windows 2003 (Server Service Pack 2)
  - Windows Vista
- Linux, separate packages for 32-bit x86 and 64-bit x86-64 processors:
  - Red Hat Enterprise Linux (RHEL) 5
  - Ubuntu Desktop 7.10
  - SUSE Enterprise Server 10 (Service Pack 1)
  - Fedora Core 8

Programming Environments

You can compile the sample program in the following environments:

Visual Studio on Windows

On Windows systems, programmers can use the C compilers in Visual Studio 2003 or Visual Studio 2005. Visual Studio 2008 might work but compatibility cannot be guaranteed.

C++ and C on Linux Systems

On Linux systems, most programmers use the GNU C compiler, version 4 and higher. The sample program compiles with the C++ compiler `g++`, but this package also supports the regular C compiler `gcc`.

Installing the Virtual Disk Development Kit

There is one install package for Windows, one for 32-bit Linux, and one for 64-bit Linux.

To install the package on Windows

1 On the Download page, choose the binary `.exe` for Windows and download it to your desktop.

2 Double-click the new desktop icon.

3 Click **Next**, read and accept the license terms, click **Next** twice, click **Install**, and **Finish**.

To Install the package on Linux

1 On the Download page, choose the binary `tar.gz` for either 32-bit Linux or 64-bit Linux.

2 Unpack the archive:

```
tar xvzf VMware-vix-disklib.*.tar.gz
```

This creates the `vmware-vix-disklib-distrib` subdirectory.

3 Change to that directory and run the installation script as root:

```
cd vmware-vix-disklib-distrib
sudo ./vmware-install.pl
```

4 Read the license terms and type **`yes`** to accept them.

Software components install in `/usr` unless you specify otherwise.

You might need to edit your `LD_LIBRARY_PATH` environment to include the library installation path, `/usr/lib/vmware-vix-disklib/lib32` (or `lib64`) for instance. Alternatively, you can add the library location to the list in `/etc/ld.so.conf` and run `ldconfig` as root.

Target System Connectivity

This section lists supported products and capabilities.

VMware Products

The Virtual Disk Development Kit supports the following VMware products:

- ESX 3.0 and 3.5
- ESXi 3.5 with Foundation License (but not with Base or Core license)
- VMware vCenter 2.0 and 2.5
- ESX 2.5 when connecting through VMware vCenter
- Hosted products including Workstation, ACE, Server, and Player

VMDK Access and Credentials

Local operations are supported by local VMDK. Access to an ESX/ESXi host is authenticated by credentials, so with proper credentials VixDiskLib can reach any VMDK on the ESX/ESXi host. VMware vCenter manages its own authentication credentials, so VixDiskLib can reach any VMDK permitted by login credentials. On all these platforms, VixDiskLib supports the following:

- Both read-only and read/write modes
- Read-only access to disk associated with any snapshot of online virtual machines
- Access to VMDK files of offline virtual machines (vCenter restricted to registered virtual machines)
- Reading of Microsoft Virtual Hard Disk (VHD) format

Virtual Disk API Functions 3

This chapter provides an overview of the Virtual Disk API in two major sections:

- "Virtual Disk and Data Structures" on page 15
- "Library Functions" on page 18

Virtual Disk and Data Structures

VMware offers many options for virtual disk layout, as encapsulated in library data structures.

VMDK File Location

VMDK files are stored in the directory that also holds virtual machine configuration files. On Linux this directory could be anywhere, so it is usually documented as `/path/to/disk`. On Windows this directory is likely to be in `C:\My Documents\My Virtual Machines`, under its virtual machine name.

VMDK files store data representing a virtual machine's hard disk drive. Almost the entire portion of a VMDK file is the virtual machine's data, with a small portion allotted to overhead. If a virtual machine is connected directly to physical disk, the VMDK file stores information about which areas the virtual machine can access.

Disk Types

The following disk types are defined in the virtual disk library:

- `VIXDISKLIB_DISK_MONOLITHIC_SPARSE` – Growable virtual disk contained in a single virtual disk file. This is the default type for hosted disk, and the only setting in the Chapter 4 sample program.
- `VIXDISKLIB_DISK_MONOLITHIC_FLAT` – Preallocated virtual disk contained in a single virtual disk file. This takes a while to create and occupies a lot of space, but might perform the best.
- `VIXDISKLIB_DISK_SPLIT_SPARSE` – Growable virtual disk split into 2GB extents (`s` sequence). These files start small but can grow to 2GB, which is the maximum on old file systems. This type is complicated but very manageable because split VMDK can be defragmented.
- `VIXDISKLIB_DISK_SPLIT_FLAT` – Preallocated virtual disk split into 2GB extents (`f` sequence). These files start at 2GB, so they take a while to create and occupy a lot of space, but available space is huge.
- `VIXDISKLIB_DISK_VMFS_FLAT` – Preallocated virtual disk compatible with ESX 3.0 and later. This is the same as "managed disk" introduced in "Virtual Disk Management" on page 9. VMFS_SPARSE exists but is not supported in this release of the Virtual Disk API.
- `VIXDISKLIB_DISK_STREAM_OPTIMIZED` – Monolithic sparse format and compressed for streaming. Stream optimized format does not support random reads or writes.
- `VIXDISKLIB_DISK_UNKNOWN` – Disk layout is unknown.

Sparse disks employ the copy-on-write (COW) mechanism, in which virtual disk contains no data in places, until copied there by a write. This optimization saves storage space.

Persistence Disk Modes

In **persistent** disk mode, changes are immediately and permanently written to the virtual disk, so that they survive until the next power on.

In **nonpersistent** mode, changes to the virtual disk are discarded when the virtual machine powers off. The VMDK files revert to their original state.

The virtual disk library does not encapsulate this distinction, which is a virtual machine setting.

VMDK File Naming

Table 3-1 further explains the different virtual disk types. The first column corresponds to "Disk Types" on page 15 but without `VIXDISKLIB_DISK` prefix. The third column gives the current names of VMDK files on Workstation hosts. This is an implementation detail; these filenames are currently in use.

NOTE When you open a VMDK file with the virtual disk library, always open the one that points to the others, not the split or flat sectors. The file to open is most likely the one with the shortest name.

For information about other virtual machine files, see section "Files that Make Up a Virtual Machine" in the *VMware Workstation User's Manual*. On ESX/ESXi hosts, VMDK files are type `VMFS_FLAT` or `VMFS_SPARSE`.

Table 3-1. VMDK Virtual Disk Files

Disk Type in API	Virtual Disk Creation on VMware Host	Filename on Host
`MONOLITHIC_SPARSE`	In **Select A Disk Type**, accepting the defaults by not checking any box produces one VMDK file that can grow larger if more space is needed. The `<vmname>` represents the name of a virtual machine.	`<vmname>.vmdk`
`MONOLITHIC_FLAT`	If you select only the **Allocate all disk space now** check box, space is pre-allocated, so the virtual disk cannot grow. The first VMDK file is small and points to a much larger one, whose filename says `flat` without a sequence number.	`<vnname>-flat.vmdk`
`SPLIT_SPARSE`	If you select only the **Split disk into 2GB files** check box, virtual disk can grow when more space is needed. The first VMDK file is small and points to a sequence of other VMDK files, all of which have an `s` before a sequence number, meaning sparse. The number of VMDK files depends on the disk size requested. As data grows, more VMDK files are added in sequence.	`<vmname>-s<###>.vmdk`
`SPLIT_FLAT`	If you select the **Allocate all disk space now** and **Split disk into 2GB files** check boxes, space is pre-allocated, so the virtual disk cannot grow. The first VMDK file is small and points to a sequence of other files, all of which have an `f` before the sequence number, meaning flat. The number of files depends on the requested size.	`<vnname>-f<###>.vmdk`
`MONOLITHIC_SPARSE or SPLIT_SPARSE`	A redo log (or child disk or delta link) is created when a snapshot is taken of a virtual machine, or with the virtual disk library. Snapshot file numbers are in sequence, without an `s` or `f` prefix. The numbered VMDK file stores changes made to the virtual disk `<diskname>` since the original parent disk, or previously numbered redo log (in other words the previous snapshot).	`<diskname>-<###>.vmdk`
`n/a`	Snapshot of a virtual machine, which includes pointers to all its `.vmdk` virtual disk files.	`<vnname>Snapshot.vmsn`

Grain Directories and Grain Tables

`SPARSE` type virtual disks use a hierarchical representation to organize sectors. See the *Virtual Disk Format 1.0* document referenced in "Virtual Disk Internal Format" on page 11. In this context, grain means granular unit of data, larger than a sector. The hierarchy includes:

- Grain directory (and redundant grain directory) whose entries point to grain tables.
  - Grain tables (and redundant grain tables) whose entries point to grains.
    - Each grain is a block of sectors containing virtual disk data. Default size is 128 sectors or 64KB.

Internationalization and Localization

The path name to a virtual machine and its VMDK can be expressed with any character set supported by the host file system, but for portability to other locales, ASCII-only path names are recommended. Future releases are expected to support Unicode UTF-8 path names, based on support in VMware products.

Adapter Types

The library can select the following adapters:

- `VIXDISKLIB_ADAPTER_IDE` – Virtual disk acts like ATA, ATAPI, PATA, SATA, and so on. You might select this adapter type when it is specifically required by legacy software.
- `VIXDISKLIB_ADAPTER_SCSI_BUSLOGIC` – Virtual SCSI disk with Buslogic adapter. This is the default on some platforms and is usually recommended over IDE due to higher performance.
- `VIXDISKLIB_ADAPTER_SCSI_LSILOGIC` – Virtual SCSI disk with LSI Logic adapter. Windows Server 2003 and most Linux virtual machines use this type by default. Performance is about the same as Buslogic.

Data Structures in Virtual Disk API

Here are important data structure objects with brief descriptions:

- `VixError` – Error code of type `uint64`.
- `VixDiskLibConnectParams` – Public types designate the virtual machine credentials `vmxSpec` (possibly through VMware vCenter), the name of its host or server, and the credential type for authentication. For more about `vmxSpec`, see "VMX Specification" on page 18.

  ```
  typedef char * vmxSpec
  typedef char * serverName
  typedef VixDiskLibCredType credType
  ```

  `VixDiskLibConnectParams::VixDiskLibCreds` – Credentials for either user ID or session ID:

  - `VixDiskLibConnectParams::VixDiskLibCreds::VixDiskLibUidPasswdCreds` – String data fields represent user name and password for authentication.
  - `VixDiskLibConnectParams::VixDiskLibCreds::VixDiskLibSessionIdCreds` – String data fields represent the session cookie, user name, and encrypted session key.

- `VixDiskLibCreateParams` – Public types represent the virtual disk (see "Disk Types" on page 15), the disk adapter (see "Adapter Types" on page 17), VMware version (such as Workstation 5 or ESX/ESXi), and capacity of the disk sector.

  ```
  typedef VixDiskLibDiskType diskType
  typedef VixDiskLibAdapterType adapterType
  typedef uint hwVersion
  typedef VixDiskLibSectorType capacity
  ```

- `VixDiskLibDiskInfo` – Public types represent the geometry in the BIOS and physical disk, the capacity of the disk sector, the disk adapter (see "Adapter Types" on page 17), the number of child-disk links (redo logs), and a string to help locate the parent disk (state before redo logs).

  ```
  VixDiskLibGeometry biosGeo
  VixDiskLibGeometry physGeo
  VixDiskLibSectorType capacity
  VixDiskLibAdapterType adapterType
  int numLinks
  char * parentFileNameHint
  ```

- `VixDiskLibGeometry` – Public types specify disk geometry. Virtual disk geometry does not necessarily correspond with physical disk geometry.

  ```
  typedef uint32 cylinders
  typedef uint32 heads
  typedef uint32 sectors
  ```

Library Functions

You can find the *VixDiskLib API Reference* by using a Web browser to open the `doc/index.html` file in the VDDK software distribution. In this section, functions are ordered by how they might be called, rather than alphabetically as in the API reference.

When the API reference says that a function supports "only hosted disks," it means virtual disk images hosted by VMware Workstation or similar products. Virtual disk images stored on vStorage VMFS partitions for ESX/ESXi hosts are called "managed disk." When the library accesses virtual disk on vStorage VMFS, all I/O goes through the ESX/ESXi host, which manages physical disk storage. The Virtual Disk API has no direct access to SAN storage.

Start Up

The `VixDiskLib_Init()` and `VixDiskLib_Connect()` functions must appear in all virtual disk programs.

Initialize the Library

`VixDiskLib_Init()` initializes the Virtual Disk API. The first two arguments, 1 and 0, represent major and minor API version numbers. The third, fourth, and fifth arguments specify log, warning, and panic handlers. DLLs and shared objects are located in `libDir`.

```
VixError vixError = VixDiskLib_Init(1, 0, &logFunc, &warnFunc, &panicFunc, libDir);
```

You may call `VixDiskLib_Init()` only once per process, because of internationalization restrictions.

Always call `VixDiskLib_Exit()` at the end of your program to de-initialize.

Connect to a Workstation or Server

`VixDiskLib_Connect()` connects the library to either a local VMware host or a remote server. For hosted disk on the local system, provide null values for most connection parameters. For managed disk on an ESX/ESXi host, specify virtual machine name, ESX/ESXi host name, user name, password, and possibly port.

```
vixError = VixDiskLib_Connect(&cnxParams, &srcConnection)
```

Always call `VixDiskLib_Disconnect()` before the end of your program.

VMX Specification

On ESX/ESXi hosts, the Virtual Machine eXecutable (VMX) is the user-space component of VMware vSphere. On Workstation and hosted products, the `.vmx` file specifies virtual machine configuration. In the connection paramaters, `vmxSpec` can be either a VMX file locator, for example an inventory path to storage, or an inventory path to the virtual machine's `.vmx` file. See "Data Structures in Virtual Disk API" on page 17 for the connection parameters and `vmxSpec` definition. These templates show the syntax of both alternatives:

```
<vmxPathName>?dcPath=<datacenter>&dsName=<dstore>

vmPath=<datacenter>/<path/to/vm>
```

- `<vmxPathName>` is the full path name of the VMX file.
- `<datacenter>` is the inventory path of the datacenter.
- `<dstore>` is the datastore name.

Here are two `vmxSpec` examples demonstrating each of the two syntax templates above. Both would be valid on VMware vCenter Server. The `vixDiskLib.h` include file documents only the first.

```
WinXP/WinXP.vmx?dcPath=Datacenter&dsName=Storage1

vmPath=Datacenter/vm/WindowsXP
```

If you create a folder and move a datacenter into it, be sure to specify `dcPath=FolderName/Datacenter`.

Disk Operations

These functions create, open, read, write, query, and close virtual disk.

Create a New Hosted Disk

`VixDiskLib_Create()` locally creates a new virtual disk, after being connected to the host. In `createParams`, you must specify the disk type, adapter, hardware version, and capacity as a number of sectors. This function supports hosted disk only. To create managed virtual disk, use `VixDiskLib_Clone()`.

```
vixError =
VixDiskLib_Create(appGlobals.connection, appGlobals.diskPath, &createParams, NULL, NULL);
```

Open a Local or Remote Disk

After the library connects to a workstation or server, `VixDiskLib_Open()` opens a virtual disk.

```
vixError =
VixDiskLib_Open(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags, &srcHandle);
```

Read Sectors From a Disk

`VixDiskLib_Read()` reads a range of sectors from an open virtual disk. You specify the beginning sector and the number of sectors. Sector size could vary, but in `<vixDiskLib.h>` it is defined as 512 bytes.

```
vixError = VixDiskLib_Read(srcHandle, i, j, buf);
```

Write Sectors To a Disk

`VixDiskLib_Write()` writes one or more sectors to an open virtual disk. This function expects the fourth parameter `buf` to be `VIXDISKLIB_SECTOR_SIZE` bytes long.

```
vixError = VixDiskLib_Write(newDisk.Handle(), i, j, buf);
```

Close a Local or Remote Disk

`VixDiskLib_Close()` closes an open virtual disk.

```
VixDiskLib_Close(srcHandle);
```

Get Information About a Disk

```
vixError = VixDiskLib_GetInfo(srcHandle, diskInfo);
```

`VixDiskLib_GetInfo()` gets data about an open virtual disk, allocating a filled-in `VixDiskLibDiskInfo` structure (page 17). Some of this information overlaps with metadata (see "Metadata Handling" on page 20).

Free Memory from Get Information

This function deallocates memory allocated by `VixDiskLib_GetInfo()`. Call it to avoid a memory leak.

```
vixError = VixDiskLib_FreeInfo(diskInfo);
```

Error Handling

These functions enhance the usefulness of error messages.

Return Error Description Text

`VixDiskLib_GetErrorText()` returns the textual description of a numeric error code.

```
char* msg = VixDiskLib_GetErrorText(errCode, NULL);
```

Free Error Description Text

`VixDiskLib_FreeErrorText()` deallocates space associated with the error description text.

```
VixDiskLib_FreeErrorText(msg);
```

Metadata Handling

Read Metadata Key from Disk

```
vixError = VixDiskLib_ReadMetadata(disk.Handle(), appGlobals.metaKey, &val[0], requiredLen,
               NULL);
```

Retrieves the value of a given key from disk metadata. The metadata for a hosted VMDK is not as extensive as for managed disk on an ESX/ESXi host. Held in a mapping file, VMFS metadata might also contain information such as disk label, LUN or partition layout, number of links, file attributes, locks, and so forth. Metadata also describes encapsulation of raw disk mapping (RDM) storage, if applicable.

Get Metadata Table from Disk

`VixDiskLib_GetMetadataKeys()` retrieves all existing keys from the metadata of a virtual disk, but not the key values. Use this in conjunction with `VixDiskLib_ReadMetadata()`. Below

```
vixError = VixDiskLib_GetMetadataKeys(disk.Handle(), &buf[0], requiredLen, NULL);
```

Here is an example of a simple metadata table. `Uuid` is the universally unique identifier for the virtual disk.

```
adapterType = buslogic
geometry.sectors = 32
geometry.heads = 64
geometry.cylinders = 100
uuid = 60 00 C2 93 7b a0 3a 03–9f 22 56 c5 29 93 b7 27
```

Write Metadata Table to Disk

`VixDiskLib_WriteMetadata()` updates the metadata of a virtual disk with the given key-value pair. If new, the library adds it to the existing metadata table. If the key already exists, the library updates its value.

```
vixError = VixDiskLib_WriteMetadata(disk.Handle(), appGlobals.metaKey, appGlobals.metaVal);
```

Cloning a Virtual Disk

Compute Space Needed for Clone

This function computes the space required (in bytes) to clone a virtual disk, after possible format conversion.

```
vixError = VixDiskLib_SpaceNeededForClone(child.Handle(), VIXDISKLIB_DISK_VMFS_FLAT, &spaceReq);
```

Clone a Disk by Copying Data

This function copies data from one virtual disk to another, converting (disk type, size, hardware) as specified.

```
vixError = VixDiskLib_Clone(appGlobals.connection, appGlobals.diskPath, srcConnection,
               appGlobals.srcPath, &createParams, CloneProgressFunc, NULL, TRUE);
```

Disk Chaining and Redo Logs

In VMDK terminology, all the following are synonyms: child disk, redo log, and delta link. From the original parent disk, each child constitutes a redo log pointing back from the present state of the virtual disk, one step at a time, to the original. This pseudo equation represents the relative complexity of backups and snapshots:

backup image < child disk = redo log = delta link < snapshot

A backup image (such as on magnetic tape) is less than a child disk because the backup image is merely a data stream. A snapshot is more than a child disk because it also contains the virtual machine state, with pointers to associated file system states on VMDK.

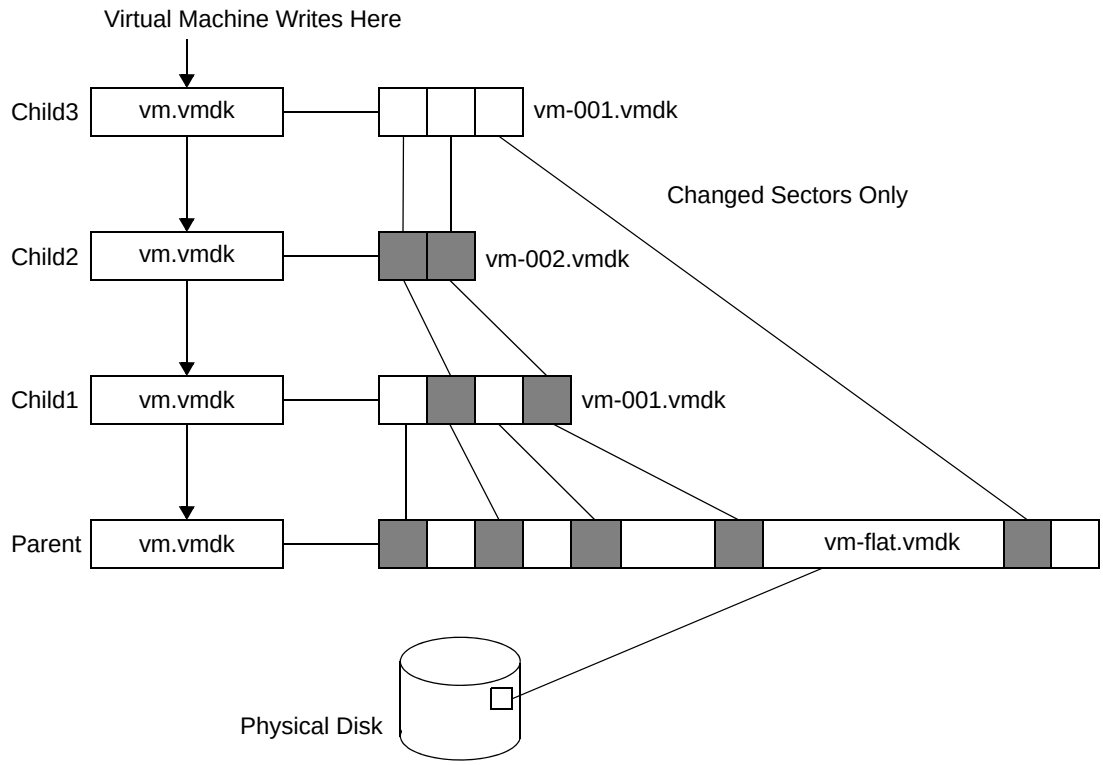
Create Child from Parent Disk

`VixDiskLib_CreateChild()` creates a child disk (or redo log) for a hosted virtual disk. Generally, you create the first child from the parent and create successive children from the latest one in the chain. The child VMDK tracks, in `SPARSE` type format, any disk sectors changed since inception, as illustrated in Figure 3-1.

```
vixError = VixDiskLib_CreateChild(parent.Handle(), appGlobals.diskPath,
              VIXDISKLIB_DISK_MONOLITHIC_SPARSE, NULL, NULL);
```

After you create a child, it is an error to open the parent, or earlier children in the disk chain. In VMware products, the children's `vm.vmdk` files point to redo logs, rather than to the parent disk, `vm–flat.vmdk` in this example. If you must access the original parent, or earlier children in the chain, use `VixDiskLib_Attach()`.

Figure 3-1. Child Disks Created from Parent



Attach Child to Parent Disk

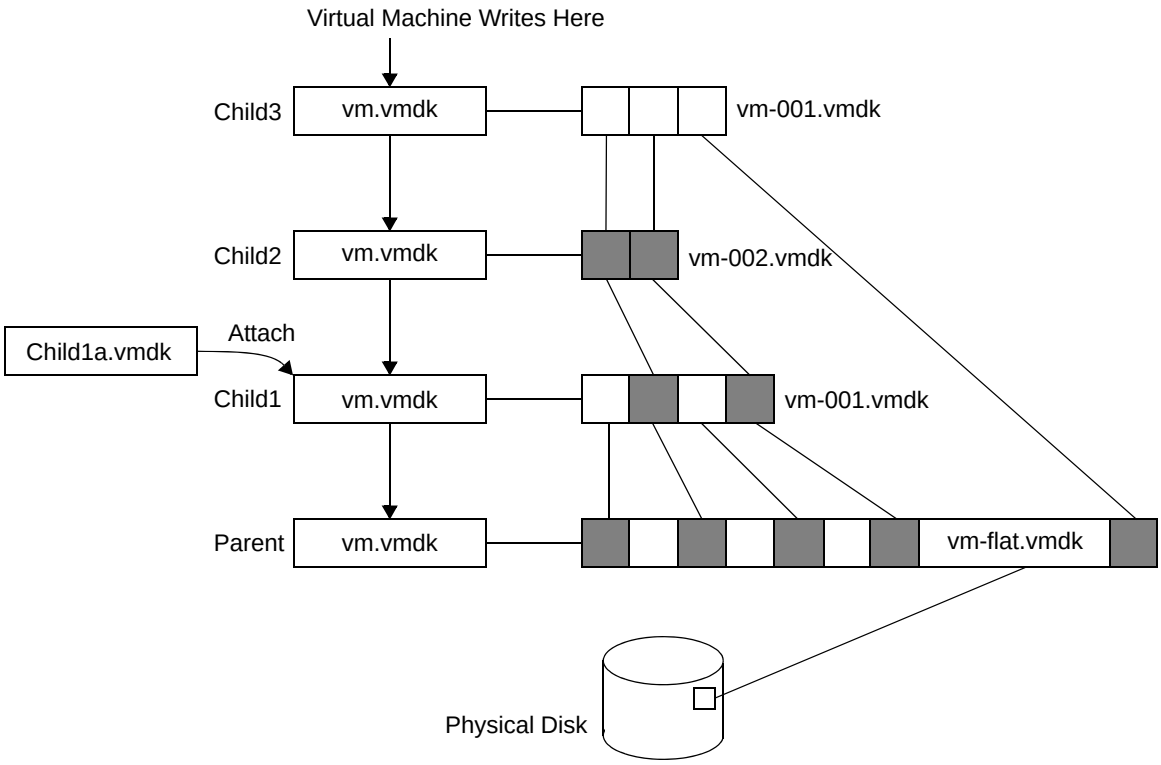
`VixDiskLib_Attach()` attaches the child disk into its parent disk chain. Afterwards, the parent handle is invalid and the child handle represents the combined disk chain of redo logs.

```
vixError = VixDiskLib_Attach(parent.Handle(), child.Handle());
```

For example, suppose you want to access the older disk image recorded by Child1. Attach the handle of new Child1a to Child1, which provides Child1a's parent handle, as shown in Figure 3-2. It is now permissible to open, read, and write the Child1a virtual disk.

The parent-child disk chain is efficient in terms of storage space, because the child VMDK records only the sectors that changed since the last `VixDiskLib_CreateChild()`. The parent-child disk chain also provides a redo mechanism, permitting programmatic access to any generation with `VixDiskLib_Attach()`.

Figure 3-2. Child Disks Created from Parent



Administrative Disk Operations

These functions rename, grow, defragment, shrink, and remove virtual disk.

Rename an Existing Disk

`VixDiskLib_Rename()` changes the name of a virtual disk. Use this function only when the virtual machine is powered off.

```
vixError = VixDiskLib_Rename(oldGlobals.diskpath, newGlobals.diskpath);
```

Grow an Existing Local Disk

`VixDiskLib_Grow()` extends an existing virtual disk by adding sectors. Supports hosted, not managed, disk.

```
vixError =
VixDiskLib_Grow(appGlobals.connection, appGlobals.diskPath, size, FALSE, GrowProgressFunc, NULL);
```

Defragment an Existing Disk

`VixDiskLib_Defragment()` defragments an existing virtual disk. Defragmentation is effective with `SPARSE` type files, but might not do anything with `FLAT` type. In either case, the function returns `VIX_OK`. Supports hosted, not managed, disk.

```
vixError = VixDiskLib_Defragment(disk.Handle(), DefragProgressFunc, NULL);
```

Defragment consolidates data in the 2GB extents, moving it to lower-numbered extents. This is independent of defragmentation tools in the guest OS, such as **Disk > Properties > Tools > Defragmentation** in Windows, or the `defrag` command for the Linux Ext2 file system.

VMware recommends defragmentation from the inside out: first within the virtual machine, then using this function or a VMware defragmentation tool, and finally within the host operating system.

Shrink an Existing Local Disk

`VixDiskLib_Shrink()` reclaims unused space in an existing virtual disk, unused space being recognized as blocks of zeroes. This is more effective (gains more space) with `SPARSE` type files than with pre-allocated `FLAT` type, although `FLAT` files might shrink by a small amount. In either case, the function returns `VIX_OK`.

```
vixError = VixDiskLib_Shrink(disk.Handle(), ShrinkProgressFunc, NULL);
```

In VMware system utilities, "prepare" zeros out unused blocks in the VMDK so "shrink" can reclaim them. In the API, use `VixDiskLib_Write()` to zero out unused blocks, and `VixDiskLib_Shrink()` to reclaim space. Shrink does not change the virtual disk capacity, but it makes more space available.

Unlink Extents to Remove Disk

`VixDiskLib_Unlink()` deletes all extents of the specified virtual disk, which unlinks (removes) the disk data. This is similar to the remove or erase command in a command tool.

```
vixError = VixDiskLib_Unlink(appGlobals.connection, appGlobals.diskPath);
```

Shut Down

All Virtual Disk API applications should call these functions at end of program.

Disconnect from Server

`VixDiskLib_Disconnect()` breaks an existing connection.

```
VixDiskLib_Disconnect(srcConnection);
```

Clean Up and Exit

`VixDiskLib_Exit()` cleans up the library before exit.

```
VixDiskLib_Exit();
```

Capabilities of Library Calls

This section describes limitations, if any.

Support for Hosted Disk

Everything is supported.

Support for Managed Disk

Some operations are not supported:

- For `VixDiskLib_Connect()` to open a connection to managed disk, you must provide valid credentials so the ESX/ESXi host can access the virtual disk.
- For `VixDiskLib_Create()` to create a managed disk on the ESX/ESXi host, first create a hosted type disk, then use `VixDiskLib_Clone()` to convert the hosted virtual disk to managed virtual disk.
- `VixDiskLib_Defragment()` can defragment hosted disks only.
- `VixDiskLib_Grow()` can grow hosted disks only.
- `VixDiskLib_Unlink()` can delete hosted disks only.

Virtual Disk API Sample Code 4

This chapter discusses the VDDK sample program, in the following sections:

Compiling the Sample Program

The sample program is written in C++, although the Virtual Disk API also supports C.

Visual C++ on Windows

Before compiling, set your search path to find the required DLL files. Choose **My Computer > Properties > Advanced > Environment Variables**, select `Path` in the **System Variables** lower list, click **Edit**, and type the following at the end, if it is not already there (assuming you installed in the default location):

```
;C:\Program Files\VMware\VMware Virtual Disk Development Kit\bin
```

If you insert that path earlier in the `Path` system variable, the semicolon goes at the end.

To compile the program, find the sample source `vixDiskLibSample.cpp` at this location:

```
C:\Program Files\VMware\VMware Virtual Disk Development Kit\doc\sample\
```

Double-click the `vcproj` file, possibly convert format to a newer version, and choose **Build > Build Solution**.

To execute the compiled program, choose **Debug > Start Without Debugging**, or type this in a command prompt after changing to the `doc\sample` location given above:

```
Debug\vixdisklibsample.exe
```

SLN and VCPROJ Files

The Visual Studio solution file `vixDiskLibSample.sln` and project file `vixDiskLibSample.vcproj` are included in the sample directory.

C++ on Linux Systems

Find the sample source in this directory:

```
/usr/share/doc/vmware-vix-disklib/sample
```

NOTE Edit `/etc/ld.so.conf` and run `ldconfig` as root, or change your `LD_LIBRARY_PATH` environment to include the library installation path, `/usr/lib/vmware-fix-disklib/lib32` (or `lib64`).

You might need to copy the source `vixDiskLibSample.cpp` and its `Makefile` to a different directory where you have write permission.

Type the `make` command to compile. Then run the application:

```
make
./vix-disklib-sample
```

Makefile

The `Makefile` fetches any packages that are required for compilation but are not installed.

Library Files Required

The virtual disk library comes with dynamic libraries, or shared objects on Linux, because it is more reliable to distribute software that way, compared to using static libraries.

Windows requires the `lib/vixDiskLib.lib` file for linking, and the `bin/*.dll` files at runtime.

Linux uses `.so` files for both linking and running. On Windows and Linux, dynamic linking is the only option.

Usage Message

Running the sample application without arguments produces the following usage message:

```
Usage: vixdisklibsample command [options] diskPath
commands:
 -create : creates a sparse virtual disk with capacity specified by -cap
 -redo parentPath : creates a redo log 'diskPath' for base disk 'parentPath'
 -info : displays information for specified virtual disk
 -dump : dumps the contents of specified range of sectors in hexadecimal
 -fill : fills specified range of sectors with byte value specified by -val
 -wmeta key value : writes (key,value) entry into disk's metadata table
 -rmeta key : displays the value of the specified metada entry
 -meta : dumps all entries of the disk's metadata
 -clone sourcePath : clone source vmdk possibly to a remote site
options: ...
```

Walk-Through of Sample Program

The sample program is the same for Windows as for Linux, with `#ifdef` blocks for Win32.

Include Files

Windows dynamic link library (DLL) declarations are in `process.h`, while Linux shared object (`.so`) declarations are in `dlfcn.h`. Windows offers the `tchar.h` extension for Unicode generic text mappings, not readily available in Linux.

Definitions and Structures

The sample program uses ten bitwise shift operations (1 << 9) to track its nine available commands and the multithread option. The Virtual Disk API has 24 function calls, some for initialization and cleanup.

The following library functions are not demonstrated in the sample program:

- `VixDiskLib_Rename()`
- `VixDiskLib_Defragment()`
- `VixDiskLib_Grow()`
- `VixDiskLib_Shrink()`
- `VixDiskLib_Unlink()`
- `VixDiskLib_Attach()`

The sample program transmits state in the `appGlobals` structure.

Dynamic Loading

The `#ifdef DYNAMIC_LOADING` block is long, starting on line 97 and ending at line 339.

This block contains function definitions for dynamic loading. It also contains the `LoadOneFunc()` procedure to obtain any requested function from the dynamic library and the `DynLoadDiskLib()` procedure to bind it.

This demonstration feature could also be called "runtime loading" to distinguish it from dynamic linking.

To try the program with runtime loading enabled on Linux, add `–DDYNAMIC_LOADING` after `g++` in the `Makefile` and recompile. On Windows, define `DYNAMIC_LOADING` in the project.

Wrapper Classes

Below the dynamic loading block are two wrapper classes, one for error codes and descriptive text, and the other for the connection handle to disk.

The error wrapper appears in `catch` and `throw` statements to simplify error handling across functions.

Wrapper class `VixDisk` is a clean way to open and close connections to disk. The only time that library functions `VixDiskLib_Open()` and `VixDiskLib_Close()` appear elsewhere, aside from dynamic loading, is in the `CopyThread()` function near the end of the sample program.

Command Functions

The print-usage message appears next, with output partially shown in "Usage Message" on page 26.

Next comes the `main()` function, which sets defaults and parses command-line arguments to determine the operation and possibly set options to change defaults. Dynamic loading occurs, if defined. Notice the all-zero initialization of the `VixDiskLibConnectParams` declared structure:

```
VixDiskLibConnectParams cnxParams = {0};
```

For connections to an ESX/ESXi host, credentials including user name and password must be correctly supplied in the `–user` and `–password` command-line arguments. Both the `–host` name of the ESX/ESXi host and its `–vm` inventory path (`vmxSpec`) must be supplied. When set, these values populate the `cnxParams` structure. Initialize all parameters, especially `vmxSpec`, or else the connection might behave unexpectedly.

A call to `VixDiskLib_Init()` initializes the library. In a production application, you can supply appropriate `log`, `warn`, and `panic` functions as parameters, in place of NULL.

A call to `VixDiskLib_Connect()` creates a library connection to disk. If host `cnxParams.serverName` is null, as it is without `–host` command-line argument, a connection is made to hosted disk on the local host. If server name is set, a connection is made to managed disk on the remote server.

Next, an appropriate function is called for the requested operation, followed by error information if applicable. Finally, the `main()` function closes the library connection to disk and exits.

DoInfo()

This procedure calls `VixDiskLib_GetInfo()` for information about the virtual disk, displays results, and calls `VixDiskLib_FreeInfo()` to reclaim memory. The parameter `disk.Handle()` comes from the `VixDisk` wrapper class discussed in "Wrapper Classes" on page 27.

In this example, the sample program connects to an ESX/ESXi host named `esx3` and displays virtual disk information for a Red Hat Enterprise Linux client. For an ESX/ESXi host, path to disk is often something like `[storage1]` followed by the virtual machine name and the VMDK filename.

```
vix-diskLib-sample –info –host esx3 –user admin –password secret "[storage1]RHEL5/RHEL5.vmdk"
capacity          = 8388608 sectors
number of links   = 1
adapter type      = LsiLogic SCSI
BIOS geometry     = 0/0/0
physical geometry = 522/255/63
```

If you multiply physical geometry numbers (522 cylinders * 255 heads per cylinder * 63 sectors per head) the result is a capacity of 8385930 sectors, although the first line says 8388608. A small discrepancy is possible. In general, you get at least the capacity that you requested. The number of links specifies the separation of a child from its original parent in the disk chain (redo logs), starting at one. The parent has one link, its child has two links, the grandchild has three links, and so forth.

DoCreate()

This procedure calls `VixDiskLib_Create()` to allocate virtual disk. Adapter type is SCSI unless specified as IDE on the command line. Size is 100MB, unless set by `–cap` on the command line. Because the sector size is 512 bytes, the code multiplies `appGlobals.mbsize` by 2048 instead of 1024. Type is always monolithic sparse and Workstation 5. In a production application, `progressFunc` and callback data can be defined rather than NULL. Type these commands to create a sample VMDK file (the first line is for Linux only):

```
export LD_LIBRARY_PATH=/usr/lib/vmware-vix-disklib/lib32
vix-disklib-sample -create sample.vmdk
```

As a VMDK file, monolithic sparse (growable in a single file) virtual disk is initially 65536 bytes (2 ^ 16) in size, including overhead. The first time you write to this type of virtual disk, as with `DoFill()` below, the VMDK expands to 131075 bytes (2 ^ 17), where it remains until more space is needed. You can verify file contents with the `–dump` option.

DoRedo()

This procedure calls `VixDiskLib_CreateChild()` to establish a redo log. A child disk records disk sectors that changed since the parent disk or previous child. Children can be chained as a set of redo logs.

The sample program does not demonstrate use of `VixDiskLib_Attach()`, which you can use to access a link in the disk chain. `VixDiskLib_CreateChild()` establishes a redo log, with the child replacing the parent for read/write access. Given a pre-existing disk chain, `VixDiskLib_Attach()` creates a related child, or a cousin you might say, that is linked into some generation of the disk chain.

For a diagram of the attach operation, see Figure 3-2, “Child Disks Created from Parent,” on page 22.

Write by DoFill()

This procedure calls `VixDiskLib_Write()` to fill a disk sector with ones (byte value FF) unless otherwise specified by `–val` on the command line. The default is to fill only the first sector, but this can be changed with options `–start` and `–count` on the command line.

DoReadMetadata()

This procedure calls `VixDiskLib_ReadMetadata()` to serve the `–rmeta` command-line option. For example, type this command to obtain the universally unique identifier:

```
vix-disklib-sample -rmeta uuid sample.vmdk
```

DoWriteMetadata()

This procedure calls `VixDiskLib_WriteMetadata()` to serve the `–wmeta` command-line option. For example, you can change the tools version from 1 to 2 as follows:

```
vix-disklib-sample -wmeta toolsVersion 2 sample.vmdk
```

DoDumpMetadata()

This procedure calls `VixDiskLib_GetMetadataKeys()` then `VixDiskLib_ReadMetadata()` to serve the `–meta` command-line option. Two read-metadata calls are needed for each key: one to determine length of the value string and another to fill in the value. See “Get Metadata Table from Disk” on page 20.

In the following example, the sample program connects to an ESX/ESXi host named `esx3` and displays the metadata of the Red Hat Enterprise Linux client's virtual disk. For an ESX/ESXi host, path to disk might be `[storage1]` followed by the virtual machine name and the VMDK filename.

```
vix-diskLib-sample -meta -host esx3 -user admin -password secret "[storage1]RHEL5/RHEL5.vmdk"
geometry.sectors = 63
geometry.heads = 255
geometry.cylinders = 522
adapterType = buslogic
toolsVersion = 1
```

Tools version and virtual hardware version appear in the metadata, but not in the disk information retrieved by "DoInfo()" on page 27. Geometry information and adapter type are repeated, but in a different format. Other metadata items not listed above might exist.

DoDump()

This procedure calls `VixDiskLib_Read()` to retrieve sectors and displays sector contents on the output in hexadecimal. The default is to dump only the first sector numbered zero, but you can change this with the `-start` and `-count` options. Here is a sequence of commands to demonstrate:

```
vix-disklib-sample -create sample.vmdk
vix-disklib-sample -fill -val 1 sample.vmdk
vix-disklib-sample -fill -val 2 -start 1 -count 1 sample.vmdk
vix-disklib-sample -dump -start 0 -count 2 sample.vmdk
od -c sample.vmdk
```

On Linux (or Cygwin) you can run the `od` command to show overhead and metadata at the beginning of file, and the repeated ones and twos in the first two sectors. The `-dump` option of the sample program shows only data, not overhead.

DoTestMultiThread()

This procedure employs the Windows thread library to make multiple copies of a virtual disk file. Specify the number of copies with the `-multithread` command-line option. For each copy, the sample program calls the `CopyThread()` procedure, which in turn calls a sequence of six Virtual Disk API routines.

On Linux the multithread option is unimplemented.

DoClone()

This procedure calls `VixDiskLib_Clone()` to make a copy of the data on virtual disk. A callback function, supplied as the sixth parameter, displays the percent of cloning completed. For local hosted disk, the adapter type is SCSI unless specified as IDE on the command line, size is 200MB, unless set by `-cap` option, and type is monolithic sparse, for Workstation 5. For an ESX/ESXi host, adapter type is taken from managed disk itself, using the connection parameters established by `VixDiskLib_Connect()`.

The final parameter `TRUE` means to overwrite if the destination VMDK exists.

The clone option is an excellent backup method. Often the cloned virtual disk is smaller, because it can be organized more efficiently. Moreover, a fully allocated flat file can be converted to a sparse representation.

Practical Programming Tasks 5

This chapter presents some practical programming challenges not covered in the sample program, including:

Scan VMDK for Virus Signatures

One of the tasks listed in "Solutions Enabled by the Virtual Disk API" on page 11 is to scan a VMDK for virus signatures. Using the framework of our sample program, a function can implement the `–virus` command-line option. The function in Example 5-1 relies on a pre-existing library routine called `SecureVirusScan()`, which typically is supplied by a vendor of antivirus software. As it does for email messages, the library routine scans a buffer of any size against the vendor's latest pattern library, and returns TRUE if it identifies a virus.

Example 5-1. Function to Scan VMDK for Viruses

```
extern int SecureVirusScan(const uint8 *buf, size_t n);
/*
*   DoVirusScan --
*      Scan the content of a virtual disk for virus signatures.
*/
static void
DoVirusScan(void)
{
    VixDisk disk(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags);
    VixDiskLibDiskInfo info;
    uint8 buf[VIXDISKLIB_SECTOR_SIZE];
    VixDiskLibSectorType sector;

    VixError vixError = VixDiskLib_GetInfo(disk.Handle(), &info);
    CHECK_AND_THROW(vixError);
    cout << "capacity = " << info.capacity << " sectors" << endl;
    // read all sectors even if not yet populated
    for (sector = 0; sector < info.capacity; sector++) {
        vixError = VixDiskLib_Read(disk.Handle(), sector, 1, buf);
        CHECK_AND_THROW(vixError);
        if (SecureVirusScan(buf, sizeof buf)) {
            printf("Virus detected in sector %d\n", sector);
        }
    }
    cout << info.capacity << " sectors scanned" << endl;
}
```

This function calls `VixDiskLib_GetInfo()` to determine the number of sectors allocated in the virtual disk. The number of sectors is available in the `VixDiskLibDiskInfo` structure, but normally not in the metadata. With `SPARSE` type layout, data can occur in any sector, so this function reads all sectors, whether filled or not. `VixDiskLib_Read()` continues without error when it encounters an empty sector full of zeroes.

The following difference list shows the remaining code changes necessary for adding the `–virus` option to the `vixDiskLibSample.cpp` sample program:

```
43a44
> #define COMMAND_VIRUS_SCAN      (1 << 10)
72a74
> static void DoVirusScan(void);
425a429
>     printf(" –virus: scan source vmdk for virus signature \n");
519a524,525
>         } else if (appGlobals.command & COMMAND_VIRUS_SCAN) {
>             DoVirusScan();
564a571,572
>         } else if (!strcmp(argv[i], "–virus")) {
>             appGlobals.command |= COMMAND_VIRUS_SCAN;
```

Creating Virtual Disks

This section discusses the types of local VMDK files and how to create virtual disk for a remote ESX/ESXi host.

Creating Local Disk

The sample program presented in Chapter 4 creates virtual disk of type `MONOLITHIC_SPARSE`, in other words one big file, not pre-allocated. This is the default for VMware Workstation, and is ideal for modern file systems, all of which support files larger than 2GB, and can hold more than 2GB of total data. This is not true of legacy file systems, such as FAT16 on MS-DOS until Windows 95, or the ISO9660 file system commonly used to write files on CD. Both are limited to 2GB per volume, although FAT was extended with FAT32 before NTFS.

However, a `SPLIT` virtual disk might be safer than the `MONOLITHIC` variety, because if something goes wrong with the underlying host file system, some data might be recoverable from uncorrupted 2GB extents. VMware products do their best to repair a damaged VMDK, but having a split VMDK increases the chance of salvaging files during repair. On the downside, `SPLIT` virtual disk involves higher overhead (more file descriptors) and increases administrative complexity.

When required for a FAT16 file system, here is how to create `SPLIT_SPARSE` virtual disk. The change is simple: the line highlighted in boldface. The sample program could be extended to have an option for this.

```
static void DoCreate(void)
{
   VixDiskLibAdapterType adapter = strcmp(appGlobals.adapterType, "scsi") == 0 ?
                                   VIXDISKLIB_ADAPTER_SCSI_BUSLOGIC : VIXDISKLIB_ADAPTER_IDE;
   VixDiskLibCreateParams createParams;
   VixError vixError;
   createParams.adapterType = adapter;
   createParams.capacity = appGlobals.mbSize * 2048;
   createParams.diskType = VIXDISKLIB_DISK_SPLIT_SPARSE;
   vixError = VixDiskLib_Create(appGlobals.connection, appGlobals.diskPath, &createParams,
               NULL, NULL);
   CHECK_AND_THROW(vixError);
}
```

NOTE You can split VMDK files into smaller than 2GB extents, but created filenames still follow the patterns shown in Table 3-1, “VMDK Virtual Disk Files,” on page 16.

This one-line change to `DoCreate()` causes creation of 200MB split VMDK files (200MB being the capacity set on the previous line) unless the `–cap` command-line argument specifies otherwise.

Creating Remote Disk

As stated in "Support for Managed Disk" on page 23, `VixDiskLib_Create()` does not support managed disk. To create a managed disk on the remote ESX/ESXi host, first create a hosted disk on the local Workstation, then convert the hosted disk into managed disk with `VixDiskLib_Clone()` over the network.

To create remote managed disk using the sample program, type the following commands:

```
./vix-disklib-sample -create -cap 1000000 virtdisk.vmdk
./vix-disklib-sample -clone virtdisk.vmdk -host esx3i -user root -password secret vmfsdisk.vmdk
```

It might be useful to write a virtual-machine provisioning application using the virtual disk library to perform the following steps:

1 Create a hosted disk VMDK with 2GB capacity, using `VixDiskLib_Create()`.

2 Write image of the guest OS and application software into the VMDK, using `VixDiskLib_Write()`.

3 Clone the hosted disk VMDK onto the VMFS file system of the ESX/ESXi host.

```
vixError = VixDiskLib_Clone(appGlobals.connection, appGlobals.diskPath,
                            srcConnection, appGlobals.srcPath,
                            &createParams, CloneProgressFunc, NULL, TRUE);
```

In this call, `appGlobals.connection` and `appGolbals.diskpath` represent the remote VMDK on the ESX/ESXi host, while `srcConnection` and `appGlobals.srcPath` represent the local hosted VMDK.

4 Power on the new guest OS to get a new virtual machine.

On Workstation, the `VixVMPowerOn()` function in the VIX API does this. For ESX/ESXi hosts, you must use the `PowerOnVM_Task` method. As easy way to use this method is in the VMware vSphere Perl Toolkit, which has the `PowerOnVM_Task()` call (non-blocking), and the `PowerOnVM()` call (synchronous).

5 Provision and deploy the new virtual machine on the ESX/ESXi host.

Special Consideration for ESX/ESXi Hosts

No matter what virtual file type you create in Step 1, it becomes type `VIXDISKLIB_DISK_VMFS_FLAT` in Step 3.

Working with Virtual Disk Data

The virtual disk library reads and writes sectors of data. It has no interface for character or byte-oriented I/O.

Reading and Writing Local Disk

Demonstrating random I/O, this function reads a sector at a time backwards through a VMDK. If it sees the string "VmWare" it substitutes the string "VMware" in its place and writes the sector back to VMDK.

```
#include <string>
static void DoEdit(void)/
{
    VixDisk disk(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags);
    uint8 buf[VIXDISKLIB_SECTOR_SIZE];
    VixDiskLibSectorType i;
    string str;
    for (i = appGlobals.numSectors; i >= 0; i--) {
        VixError vixError;
        vixError = VixDiskLib_Read(disk.Handle(), appGlobals.startSector + i, 1, buf);
        CHECK_AND_THROW(vixError);
        str = buf;
        if (pos = str.find("VmWare", 0)) {
            str.replace(pos, 5, "VMware");
            buf = str;
            vixError = VixDiskLib_Write(disk.Handle(), appGlobals.startSector + i, 1, buf);
            CHECK_AND_THROW(vixError);
        }
    }
}
```

Reading and Writing Remote Disk

The function is similar to this for remote virtual disk on ESX/ESXi hosts, but calls `VixDiskLib_Connect()` with authentication credentials instead of passing NULL parameters.

```
if (appGlobals.isRemote) {
          cnxParams.vmxSpec = NULL;
          cnxParams.serverName = appGlobals.host;
          cnxParams.credType = VIXDISKLIB_CRED_UID;
          cnxParams.creds.uid.userName = appGlobals.userName;
          cnxParams.creds.uid.password = appGlobals.password;
          cnxParams.port = appGlobals.port;
       }
       VixError vixError = VixDiskLib_Init(1, 0, NULL, NULL, NULL, NULL);
       CHECK_AND_THROW(vixError);
       vixError = VixDiskLib_Connect(&cnxParams, &appGlobals.connection);
```

Deleting a Disk (Unlink)

The function to delete virtual disk files is `VixDiskLib_Unlink()`. It takes two arguments: a connection and a VMDK filename.

```
vixError = VixDiskLib_Unlink(appGlobals.connection, appGlobals.diskPath);
```

Effects of Deleting a Virtual Disk

When you delete a VMDK, you lose all the information it contained. In most cases, the host operating system prevents you from doing this when a virtual machine is running. However, if you delete a VMDK with its virtual machine powered off, that guest OS becomes unbootable.

Renaming a Disk

The function to rename virtual disk files is `VixDiskLib_Rename()`. It takes two arguments: the old and the new VMDK filenames.

```
vixError = VixDiskLib_Rename(oldGlobals.diskpath, newGlobals.diskpath);
```

Effects of Renaming a Virtual Disk

The server expects VMDK files of its guest OS virtual machines to be in a predictable location. Any file accesses that occur during renaming might cause I/O failure and possibly cause a guest OS to fail.

Working with Disk Metadata

With vStorage VMFS on ESX/ESXi hosts, disk metadata becomes important because it stores information about the raw disk mapping (RDM) and interactions with the containing file system.

Managing Child Disks

In the Virtual Disk API, redo logs are managed as a parent-child disk chain, each child being the redo log of disk changes made since its inception. Trying to write on the parent after creating a child results in an error. The library expects you to write on the child instead. See Figure 3-2, "Child Disks Created from Parent," on page 22 for a diagram.

Creating Redo Logs

Ordinarily a redo log is created by a snapshot of the virtual machine, allowing restoration of both disk data and the virtual machine state.

For example, you could write an application to create new redo logs, independent of snapshots, at 3:00 AM nightly. This allows you to re-create data for any given day. When you create a redo log while the virtual machine is running, the VMware host re-arranges file pointers so the primary VMDK, `<vmname>.vmdk` for example, keeps track of redo logs in the disk chain.

To re-create data for any given day

1 Locate the <vmname>–<NNN>.vmdk redo log for the day in question.

<NNN> is a sequence number. You can identify this redo log by its timestamp.

2 Initialize the virtual disk library and open the redo log to obtain its parent handle.

3 Create a child disk with the VixDiskLib_Create() function, and attach it to the parent:

```
vixError = VixDiskLib_Attach(parent.Handle(), child.Handle());
```

4 Read and write the virtual disk of the attached child.

Virtual Disk in Snapshots

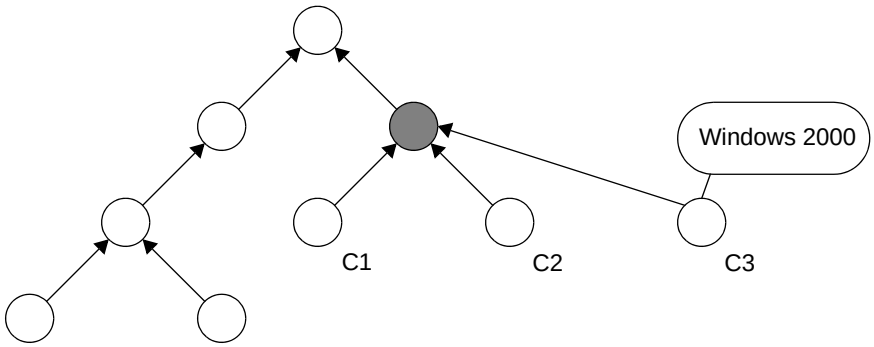
The Virtual Disk API provides the following features to deal with the disk component of snapshots:

- Attaching an arbitrary child in a disk chain
- Opening read-only virtual disks
- Ability to open snapshot disk on ESX/ESXi hosts through VMware vCenter

Windows 2000 Read-Only File System

Another use of parent-child disk chaining is to create read-only access for Windows 2000, which has no option for mounting a read-only file system. In Figure 5-1, the gray circle represents a virtual disk that must remain read-only because it has children. In this example, you want the Windows 2000 virtual machine to use that virtual disk, rather than the newer ones C1 and C2. Create new child disk C2, attach to the gray virtual disk as parent, and mount C3 as the virtual disk of the Windows 2000 guest OS.

Figure 5-1. Attaching Virtual Read/Write Disk for Windows 2000



Interfacing With the VIX API

The VIX API is a popular, easy-to-use developer interface for VMware Workstation and other hosted products. See the Support section of the VMware Web site for information about the VIX API:

http://www.vmware.com/support/developer/vix-api

The *VIX API Reference Guide* includes function reference pages for C++, Perl, and COM, a component object model for Microsoft C#, VBScript, and Visual Basic. Most of the function reference pages include helpful code examples. Additionally, the above Web page includes examples for power on and off, suspending a virtual machine, taking a snapshot, asynchronous use, and a polling event pump.

Virus Scan all Hosted Disk

Suppose you want to run the antivirus software presented in "Scan VMDK for Virus Signatures" on page 31 for all virtual machines hosted on a VMware Workstation. Here is the high-level algorithm for an VIX-based application that would scan hosted disk on all virtual machines:

1 Write an application including both the Virtual Disk API and the VIX API.

2 Initialize the virtual disk library with `VixDiskLib_Init()`.

3 Connect VIX to the Workstation host with `VixHost_Connect()`.

4 Call `VixHost_FindItems()` with item-type (second argument) `VIX_FIND_RUNNING_VMS`.

This provides to a callback routine (fifth argument) the name of each virtual machine, one at a time. To derive the name of each virtual machine's disk, append "`.vmdk`" to the virtual machine name.

5 Write a callback function to open the virtual machine's VMDK.

Your callback function must be similar to the `VixDiscoveryProc()` callback function shown as an example on the `VixHost_FindItems()` page in the *VIX API Reference Guide*.

6 Instead of printing "Found virtual machine" in the callback function, call the `DoVirusScan()` function shown in "Scan VMDK for Virus Signatures" on page 31.

7 Decontaminate any infected sectors that the virus scanner located.

Interfacing With VMware vSphere

The VMware vSphere API is a developer interface for ESX/ESXi hosts and VMware vCenter. See the Support section of the VMware Web site for information about the VMware vSphere SDK:

http://www.vmware.com/support/developer/vc-sdk

The *Developer's Setup Guide* for the VMware vSphere SDK 2.5 has a chapter describing how to set up your programming environment for Microsoft C#. Some of the information applies to C++ also.

The *Programming Guide* for the VMware vSphere SDK 2.5 contains sample applications written in Java, but no examples in C++. You might find the Java examples helpful.

ESX/ESXi hosts and the VMware vSphere API use a programming model based on Web services, in which clients generate Web services description language (WSDL) requests that pass over the network as XML messages encapsulated in simple object access protocol (SOAP). On ESX/ESXi hosts or VMware vCenter, the vSphere layer answers client requests, possibly passing back SOAP responses. This is a very different programming model than the object-oriented function-call interface of C++ and the VIX API.

Virus Scan All Managed Disk

Suppose you want to run the antivirus software presented in "Scan VMDK for Virus Signatures" on page 31 for all virtual machines hosted on an ESX/ESXi host. Here is the high-level algorithm for a VMware vSphere solution that can scan managed disk on all virtual machines:

1 Using the VMware vSphere Perl Toolkit, write a Perl script that connects to a given ESX/ESXi host.

2 Call `Vim::find_entity_views()` to find the inventory of every `VirtualMachine`.

3 Call `Vim::get_inventory_path()` to get the virtual disk name in its appropriate resource.

The VMDK filename is available as `diskPath` in the `GuestDiskInfo` data object.

4 Using Perl's `system(@cmd)` call, run the extended `vixDiskLibSample.exe` program with `–virus` option.

For ESX/ESXi hosts you must specify `–host`, `–user`, and `–password` options.

5 Decontaminate any infected sectors that the virus scanner located.

Flexible Transport for Virtual Disk

After the release of VDDK 1.0, customers and partners requested additional features to support SAN and to help increase I/O performance. When reading managed disk, VDDK 1.0 required access over the network, through an ESX/ESXi host. Now it is possible to access virtual disk data directly on a storage device, LAN-free. To transparently select the most efficient transport method, a new set of APIs is available, including:

- `VixDiskLib_ConnectEx()` – Establishes a connection using the best transport protocol available for accessing a given machine's virtual disk. Replaces `VixDiskLib_Connect()` in your application.
- `VixDiskLib_ListTransportModes()` – Lists transport modes that the virtual disk library supports.

These new virtual disk interfaces are discussed in the section "APIs to Select Transport Methods" on page 39. Protocols available to `VixDiskLib_ConnectEx()` are presented in "Virtual Disk Transport Methods," below.

NOTE Library routines for flexible transport are implemented but marked experimental in VDDK 1.1 beta2.

Virtual Disk Transport Methods

VMware supports file-based or image-level backups of virtual machines hosted on an ESX/ESXi host with SAN or iSCSI storage. VMware virtual machines can read data directly from shared VMFS LUNs, so backups are highly efficient and do not put significant load on production ESX/ESXi hosts or the virtual network.

This VDDK release makes it possible to integrate storage-related applications, including backup, using an API rather than a command-line interface. VMware has developed back-ends that enable efficient access to data stored on ESX/ESXi server farms. Third party vendors now have access to these data paths (internally called VixTransport) through the virtual disk library. The motivation behind this flexible transport library was to provide the most efficient transport method available, to help developers maximize application performance.

Currently VMware supports the transport methods discussed below: file, SAN, HotAdd, and LAN (NBD).

File

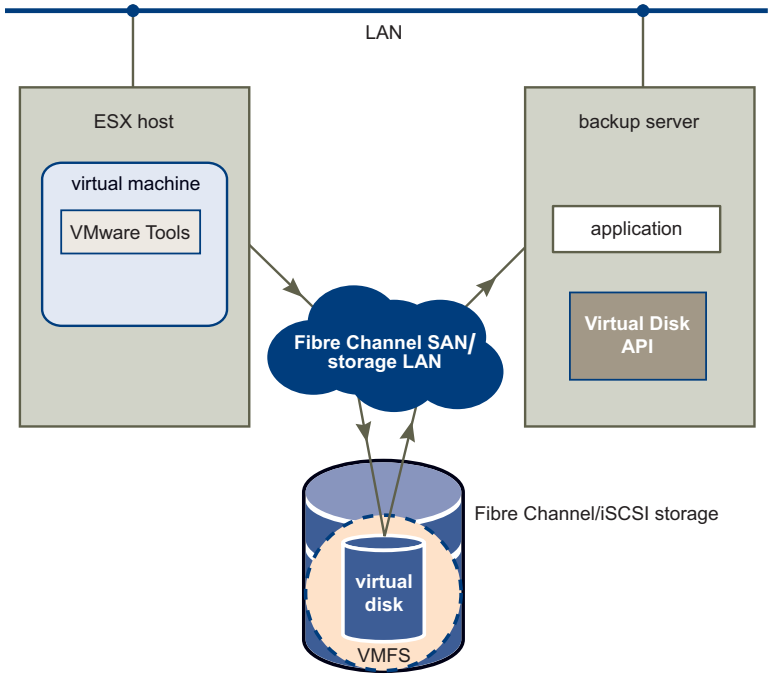
The library reads virtual disk data from `/vmfs/volumes` on ESX/ESXi hosts, or from the local filesystem on hosted products. This file transport method is built into the virtual disk library, so it is always available.

SAN

In this mode, the virtual disk library obtains information from an ESX/ESXi host about the layout of VMFS LUNs, and using this information, reads data directly from the SAN or iSCSI LUN where a virtual disk resides. This is the fastest transport method for applications deployed on a SAN-connected ESX/ESXi host.

SAN mode requires applications to run on a physical machine (a backup server, for example) with access to FibreChannel or iSCSI SAN containing the virtual disks to be accessed. This is an efficient data path, as shown in Figure A-1, because no data needs to be transferred through the production ESX/ESXi host. If the backup server is also a media server, with optical media or tape drives, backups can be made entirely LAN-free.

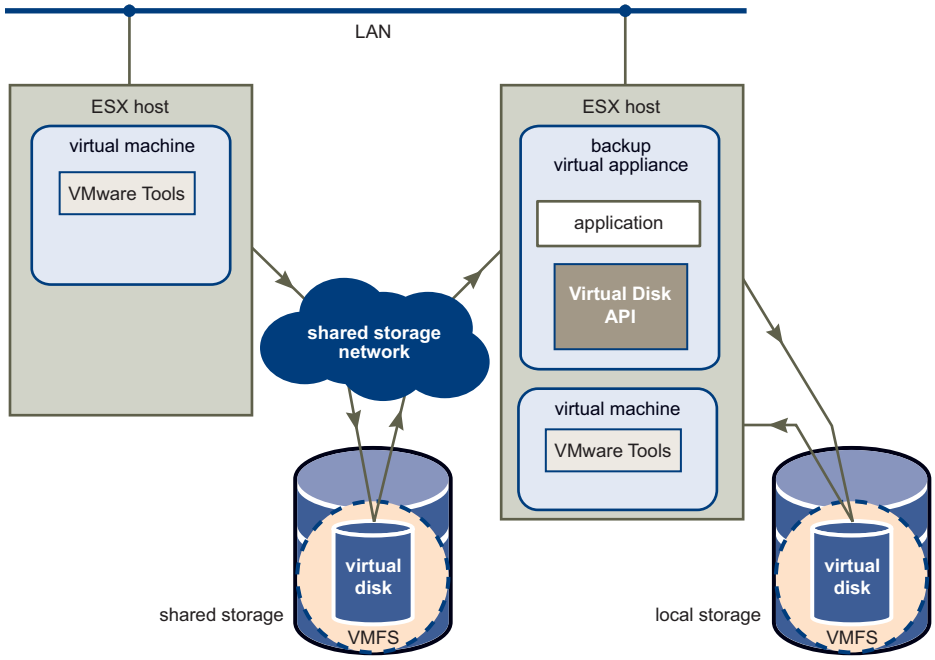
Figure A-1. SAN Transport Mode for Virtual Disk



HotAdd

If the application runs in a virtual machine, it can create a linked-clone virtual machine from the backup snapshot and read the linked clone's virtual disks for backup. This involves a SCSI hot-add on the host where the application is running – disks associated with the linked clone are hot-added on the virtual machine. VixTransport handles the temporary linked clone and hot attachment of virtual disks. VixDiskLib opens and reads the hot-added disks as a "whole disk" VMDK (virtual disk on the local host).

Figure A-2. HotAdd Transport Mode for Virtual Disk



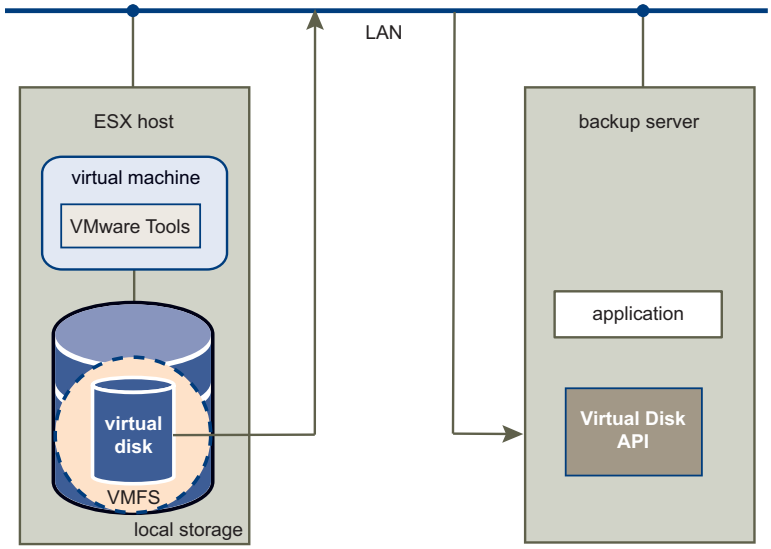
Running the backup server on a virtual machine has two advantages: it is easy to move a virtual machine to a new media server, and it can also back up local storage without using the LAN, although this incurs more overhead on the physical ESX/ESXi host than when using SAN transport mode.

SCSI hot-add is a good way to get virtual disk data from guest virtual machines directly to the ESX/ESXi host on which they are running.

LAN (NBD)

When no other transport mode is available, storage applications can uses LAN transport for data access, either NBD or NBDSSL. NBD (network block device) is a Linux kernel module that treats storage on a remote host as a block device. NBDSSL encrypts all data passed over the TCP/IP connection. The LAN transport method is built into the virtual disk library, so it is always available.

Figure A-3. LAN (NBD) Transport Mode for Virtual Disk



In this mode, the ESX/ESXi host reads data from storage and sends it across a network to the backup server. For LAN transport, virtual disks cannot be larger than 1TB each. As its name implies, this transport mode is not LAN-free, unlike SAN and HotAdd transport. However, LAN transport offers the following advantages:

- The ESX/ESXi host can use any storage device, including local storage or NAS.
- The backup server could be a virtual machine, so you can use a resource pool and scheduling capabilities of VMware vSphere to minimize the performance impact of backup. For example, you can put the backup server in a different resource pool than the production ESX/ESXi hosts, with lower priority for backup.
- If the ESX/ESXi host and backup server are on a private network, you can use unencrypted data transfer, which is faster and consumes fewer resources than NBDSSL. If you need to protect sensitive information, you have the option of transferring virtual machine data in an encrypted form.

Licensing

Currently the flexible transport license for VDDK includes all transport types. In the future, licensing might be based on transport type and read/write capability.

APIs to Select Transport Methods

This section summarizes the new APIs for selecting transport method.

List Available Transport Methods

The `VixDiskLib_ListTransportModes()` function returns the currently supported transport methods as a colon-separated string value, currently "`file:san:hotadd:nbd`" where `nbd` indicates LAN transport. When available, SSL encrypted NBD transport is shown as `nbdssl`.

```
printf("Transport methods: %s\n", VixDiskLib_ListTransportModes());
```

Connect to VMware vSphere

`VixDiskLib_ConnectEx()` connects the library to managed disk on a remote ESX/ESXi host or through VMware vCenter. For hosted disk on the local system, it works the same as `VixDiskLib_Connect()`. `VixDiskLib_ConnectEx()` takes three additional parameters:

- Boolean indicating `TRUE` for read-only access, often faster, or `FALSE` for read/write access. If connecting read-only, later calls to `VixDiskLib_Open()` are always read-only regardless of the `openFlags` setting.
- Name of the snapshot to back up.
- Preferred transport method, or `NULL` to accept the defaults.

```
VixDiskLibConnectParams cnxParams = {0};
if (appGlobals.isRemote) {
    cnxParams.vmName = vmName;
    cnxParams.serverName = hostName;
    cnxParams.credType = VIXDISKLIB_CRED_UID;
    cnxParams.creds.uid.userName = userName;
    cnxParams.creds.uid.password = password;
    cnxParams.port = port;
}
VixError vixError = VixDiskLib_ConnectEx(&cnxParams,
                        TRUE,
                        “snapshot-47",
                        NULL,
                        &connection);
```

The snapshot name is required for SAN and HotAdd transport methods, and to access virtual disks of a powered-on virtual machine.

Get Selected Transport Method

The `VixDiskLib_GetTransportMode()` function returns the transport method selected for `diskHandle`.

```
printf(“Selected transport method: %s\n”, VixDiskLib_GetTransportMode(diskHandle));
```

Clean Up After Disconnect

If virtual machine state was not cleaned up correctly after connection shut down, `VixDiskLib_Cleanup()` removes extra state for each virtual machine. Its three parameters specify connection, and pass back the number of virtual machines cleaned up, and the number remaining to be cleaned up.

```
int numCleanedUp, numRemaining;
VixError vixError = VixDiskLib_Cleanup(&cnxParams,
                        &numCleanedUp,
                        &numRemaining);
```

Updating Applications for Flexible Transport

To update your applications for flexible transport, follow these steps:

1 Find all instances of `VixDiskLib_Connect()`.

2 Except for instances specific to hosted disk, change all these to `VixDiskLib_ConnectEx()`.

3 Add parameters in the middle:

   a `TRUE` for high performance read-only access, `FALSE` for read/write access.

   b Snapshot name, if applicable.

   c `NULL` to accept transport method defaults (recommended).

4 Find `VixDiskLib_Disconnect()` near the end of program, and for safety add a `VixDiskLib_Cleanup()` call immediately afterwards.

5 Compile with the new flexible-transport-enabled version of VixDiskLib.

Developing Backup Applications

The flexible transport functions are useful for backing up or restoring data on virtual disks managed by VMware vSphere. Backup is based on the snapshot mechanism, which provides a data view at a certain point in time, and allows access to quiescent data on the parent disk while the child disk continues changing.

A typical backup application follows this algorithm:

- Possibly through VMware vCenter, contact the ESX/ESXi host containing the target virtual machine.
- Ask the ESX/ESXi host to produce a snapshot of the target virtual machine.
- Using the vSphere API, capture the virtual machine configuration (`VirtualMachineConfigInfo`) and the changed block information (with `queryChangedDiskAreas`).
- Using flexible transport functions and VixDiskLib, access and save data in the snapshot.
- Ask the ESX/ESXi host to delete the backup snapshot.

A typical back-in-time disaster recovery or file-based restore follows this algorithm:

- Possibly through VMware vCenter, contact the ESX/ESXi host containing the target virtual machine.
- Ask the ESX/ESXi host to halt and power off the target virtual machine.
- Using flexible transport functions, restore a snapshot from saved backup data.
- For disaster recovery to a previous point in time, have the virtual machine revert to the restored snapshot. For file-based restore, mount the snapshot and restore requested files.

The technical note *Designing Backup Applications for VMware vSphere* presents these algorithms in more detail and includes code samples.

Backup and Recovery Example

The VMware vSphere API method `queryChangedDiskArea` returns a list of disk sectors that changed between an existing snapshot, and some previous time identified by a change ID.

The `queryChangedDiskAreas` method takes four arguments, including a snapshot reference and a change ID. It returns a list of disk sectors that changed between the time indicated by the change ID and the time of the snapshot. If you specify change ID as * (star), `queryChangedDiskAreas` returns a list of allocated disk sectors so your backup can skip the unallocated sectors of sparse virtual disk.

Suppose that you create an initial backup at time *T1*. Later at time *T2* you take an incremental backup, and another incremental backup at time *T3*. (You could use differential backups instead of incremental backups, which would trade off greater backup time and bandwidth for shorter restore time.)

For the full backup at time T1:

1 Keep a record of the virtual machine configuration, `VirtualMachineConfigInfo`.

2 Create a snapshot of the virtual machine, naming it **snapshot_T1**.

3 Obtain the change ID for each virtual disk in the snapshot, **changeId_T1** (per VMDK).

4 Back up the sectors returned by `queryChangedDiskAreas(..."*")`, avoiding unallocated disk.

5 Delete **snapshot_T1**, keeping a record of **changeId_T1** along with lots of backed-up data.

For the incremental backup at time T2:

1 Create a snapshot of the virtual machine, naming it **snapshot_T2**.

2 Obtain the change ID for each virtual disk in the snapshot, **changeId_T2** (per VMDK).

3 Back up the sectors returned by `queryChangedDiskAreas(snapshot_T2,... changeId_T1)`.

4 Delete **snapshot_T2**, keeping a record of **changeId_T2** along with backed-up data.

For the incremental backup at time T3:

1 Create a snapshot of the virtual machine, naming it **snapshot_T3**.

At time T3 you can no longer obtain a list of changes between T1 and T2.

2 Obtain the change ID for each virtual disk in the snapshot, **changeId_T3** (per VMDK).

3 Back up the sectors returned by `queryChangedDiskAreas(snapshot_T3,... changeId_T2)`.

A differential backup could be done with `queryChangedDiskAreas(snapshot_T3,... changeId_T1)`.

4 Delete **snapshot_T3**, keeping a record of **changeId_T3** along with backed-up data.

For a disaster recovery at time T4:

1 Create a new virtual machine with no guest operating system installed, using configuration parameters you previously saved from `VirtualMachineConfigInfo`. You do not need to format the virtual disks, because restored data includes formatting information.

2 Restore data from the backup at time T3. Keep track of which disk sectors you restore.

3 Restore data from the incremental backup at time T2, skipping any sectors already recovered.

With differential backup, you can skip copying the T2 backup.

4 Restore data from the full backup at time T1. The reason for working backwards is to get the newest data while avoiding unnecessary data copying.

5 Power on the recovered virtual machine.

Virtual Disk Mount API

B

After the release of VDDK 1.0, customers and partners requested an API to support local and remote mounting of virtual disks. The `vmware-mount` command does this, but analogous library routines were not provided.

In upcoming releases, the vixMntapi library will be packaged with the VDDK, and installed in the same directory as VixDiskLib. However VixMntapi involves a separate library for loading.

CAUTION The new virtual disk mount routines are implemented but marked experimental in VDDK 1.1 beta2. Interfaces may change in upcoming releases, and backward compatibility is not guaranteed.

In VDDK 1.1 beta2, the `vixMntapi` library is available for Windows only, not Linux.

The VixMntapi Library

The VixMntapi library supports guest operating systems on multiple platforms. On POSIX systems it requires FUSE mount, available on recent Linux systems, and freely available on the SourceForge Web site.

Header File

Definitions are contained in the following header file, installed in the same directory as `vixDiskLib.h`:

```
#include "vixMntapi.h"
```

Types and Structures

This section summarizes the important types and structures.

Operating System Information

The `VixOsInfo` structure encapsulates the following information:

- Family of the guest operating system, `VixOsFamily`, one of the following:
  - Windows NT-based
  - Windows 9x DOS-based
  - Linux
  - Netware
  - Solaris
  - FreeBSD
  - OS/2
  - Mac OS X (Darwin)
- Major version and minor version of the operating system

- Whether it is 64-bit or 32-bit
- Vendor and edition of the operating system
- Location where the operating system is installed

Disk Volume Information

The `VixVolumeInfo` structure encapsulates the following information:

- Type of the volume, `VixVolumeType`, one of the following:
  - Basic partition
  - GPT – GUID Partition Table, used by Extensible Firmware Interface (EFI) disk.
  - Dynamic volume
  - LVM – Logical Volume Manager disk storage.
- Whether the volume is mounted on the proxy
- Path to the volume mount point on the proxy, or NULL if the volume is not mounted
- Number of mount points for the volume in the guest, 0 if the volume is not mounted
- Mount points for the volume in the guest

Function Calls

To obtain these functions, load the vixMntapi library separately from the vixDiskLib library. On Windows, compile with the `vixMntapi.lib` library so your program can load the `vixMntapi.dll` runtime.

The remainder of this section lists the available function calls in the vixMntapi library. Under parameters, [in] indicates input parameters, and [out] indicates output parameters.

All functions that return `vixError` return `VIX_OK` on success, otherwise a suitable VIX error code.

VixMntapi_Init()

Initializes the VixMntapi library.

```
VixError
VixMntapi_Init(uint32 majorVersion,
               uint32 minorVersion,
               VixDiskLibGenericLogFunc *log,
               VixDiskLibGenericLogFunc *warn,
               VixDiskLibGenericLogFunc *panic,
               const char *libDir,
               const char *tmpDir);
```

Parameters:

- `majorVersion` [in] and `minorVersion` [in] API major and minor version numbers.
- `log` [in] Callback function to write log messages.
- `warn` [in] Callback function to write warning messages.
- `panic` [in] Callback function to report fatal errors.
- `libDir` [in]
- `tmpDir` [in]

VixMntapi_Exit()

Cleans up the VixMntapi library.

```
void
VixMntapi_Exit();
```

VixMntapi_OpenDiskSet()

Opens the set of disks for mounting. All the disks for a dynamic volume or logical volume management (LVM) volume must be opened together.

```
VixError
VixMntapi_OpenDiskSet(VixDiskLibHandle diskHandles[],
                      int numberOfDisks,
                      uint32 openMode,
                      VixDiskSetHandle *handle);
```

The `VixDiskLibHandle` type, defined in `vixDiskLib.h`, is the same as for the `diskHandle` parameter in the `VixDiskLib_Open()` function, but here it is an array instead of a single value.

Parameters:

- `diskHandles` [in] Array of handles to open disks.
- `numberOfDisks` [in] Number of disk handles in the array.
- `openMode` [in] One of the following:
  - `VIXMNTAPI_FLAG_MOUNT_READ_ONLY`
  - `VIXMNTAPI_FLAG_MOUNT_NON_PERSISTENT`
  - `VIXMNTAPI_FLAG_MOUNT_DEFAULT`
- `handle` [out] Disk set handle to be filled in.

VixMntapi_CloseDiskSet()

Closes the disk set.

```
VixError
VixMntapi_CloseDiskSet(VixDiskSetHandle diskSet);
```

Parameter:

- `diskSet` [in] Handle to an open disk set.

VixMntapi_GetVolumeHandles()

Retrieves handles to the volumes in the disk set.

```
VixError
VixMntapi_GetVolumeHandles(VixDiskSetHandle diskSet,
                           int *numberOfVolumes,
                           VixVolumeHandle *volumeHandles[]);
```

Parameters:

- `diskSet` [in] Handle to an open disk set.
- `numberOfVolumes` [out] Number of volume handles needed.
- `volumeHandles` [out] Array of volume handles to be filled in.

VixMntapi_FreeVolumeHandles()

Frees memory allocated by `VixMntapi_GetVolumeHandles()`.

```
void
VixMntapi_FreeVolumeHandles(VixVolumeHandle *volumeHandles);
```

Parameter:

- `volumeHandles` [in] Volume handle to be freed.

VixMntapi_GetOsInfo()

Retrieves information about the default operating system in the disk set.

```
VixError
VixMntapi_GetOsInfo(VixDiskSetHandle diskSet,
                    VixOsInfo **info);
```

Parameters:

- `diskSet` [in] Handle to an open disk set.
- `info` [out] OS information to be filled in.

VixMntapi_FreeOsInfo()

Frees memory allocated by `VixMntapi_GetOsInfo()`.

```
void
VixMntapi_FreeOsInfo(VixOsInfo* info);
```

Parameter:

- `info` [in] OS info to be freed.

VixMntapi_MountVolume()

Mounts the volume. After mounting the volume, use `VixMntapi_GetVolumeInfo()` to obtain the path to the mounted volume.

```
VixError
VixMntapi_MountVolume(VixVolumeHandle volumeHandle,
                      Bool isReadOnly);
```

Parameters:

- `volumeHandle` [in] Handle to a volume.
- `isReadOnly` [in] Whether to mount the volume in read-only mode. Does not override `openMode`.

VixMntapi_DismountVolume()

Unmounts the volume.

```
VixError
VixMntapi_DismountVolume(VixVolumeHandle volumeHandle,
                      Bool force);
```

Parameters:

- `volumeHandle` [in] Handle to a volume.
- `force` [in] Force unmount even if files are open on the volume.

VixMntapi_GetVolumeInfo()

Retrieves information about a volume. Some of the volume information is available only if the volume is mounted, so this must be called after calling `VixMntapi_MountVolume()`.

```
VixError
VixMntapi_GetVolumeInfo(VixVolumeHandle volumeHandle,
                      VixVolumeInfo **info);
```

Parameters:

- `volumeHandle` [in] Handle to a volume.
- `info` [out] Volume information to be filled in.

VixMntapi_FreeVolumeInfo()

Frees memory allocated in `VixMntapi_GetVolumeInfo()`.

```
void
VixMntapi_FreeVolumeInfo(VixVolumeInfo *info);
```

Parameter:

- `info` [in] Volume info to be freed.

Virtual Disk API Errors

C

Finding Error Code Documentation

For a list of Virtual Disk API error codes, see the online reference guide *Introduction to the VixDiskLib API*:

- Windows – `C:\Program Files\VMware\VMware Virtual Disk Development Kit\doc\intro.html`
- Linux – `/usr/share/doc/vmware-vix-disklib/intro.html`

In a Web browser, click the **Error Codes** link in the upper left frame, and click any link in the lower left frame. The right-hand frame displays an alphabetized list of error codes, with explanations.

Association With VIX API Errors

Most error codes in the Virtual Disk API are shared with the VMware VIX API, which explains the `VIX` prefix. For information about the VIX API, including its online reference guide to functions and error codes, see the Support section of the VMware Web site.

The following errors were introduced with the Virtual Disk API, or with new versions of the VIX API, so they are not found in the online documentation. Some of these involve virtual disk operations, while others involve connecting to a remote VMware Server.

```
VIX_E_BUFFER_TOOSMALL
VIX_E_CANNOT_CONNECT_TO_HOST
VIX_E_DISK_CANTSHRINK
VIX_E_DISK_CID_MISMATCH
VIX_E_DISK_INVAL
VIX_E_DISK_INVALIDCHAIN
VIX_E_DISK_INVALIDPARTITIONTABLE
VIX_E_DISK_INVALID_CONNECTION
VIX_E_DISK_KEY_NOTFOUND
VIX_E_DISK_NEEDKEY
VIX_E_DISK_NEEDSREPAIR
VIX_E_DISK_NEEDVMFS
VIX_E_DISK_NOINIT
VIX_E_DISK_NOIO
VIX_E_DISK_NOKEY
VIX_E_DISK_NOKEYOVERRIDE
VIX_E_DISK_NOTENCDESC
VIX_E_DISK_NOTENCRYPTED
VIX_E_DISK_NOTNORMAL
VIX_E_DISK_NOTSUPPORTED
VIX_E_DISK_OPENPARENT
VIX_E_DISK_OUTOFRANGE
VIX_E_DISK_PARTIALCHAIN
VIX_E_DISK_PARTMISMATCH
VIX_E_DISK_RAWTOOBIG
VIX_E_DISK_RAWTOOSMALL
VIX_E_DISK_SUBSYSTEM_INIT_FAIL
VIX_E_DISK_TOOMANYOPENFILES
VIX_E_DISK_TOOMANYREDO
```

VIX_E_DISK_UNSUPPORTEDDISKVERSION
VIX_E_HOST_DISK_INVALID_VALUE
VIX_E_HOST_DISK_SECTORSIZE
VIX_E_HOST_FILE_ERROR_EOF
VIX_E_HOST_NBD_HASHFILE_INIT
VIX_E_HOST_NBD_HASHFILE_VOLUME
VIX_E_HOST_NETBLKDEV_HANDSHAKE
VIX_E_HOST_NETWORK_CONN_REFUSED
VIX_E_HOST_SERVER_NOT_FOUND
VIX_E_HOST_SOCKET_CREATION_ERROR
VIX_E_HOST_TCP_CONN_LOST
VIX_E_HOST_TCP_SOCKET_ERROR
VIX_E_NOT_ALLOWED_DURING_VM_RECORDING
VIX_E_NOT_ALLOWED_DURING_VM_REPLAY
VIX_E_NOT_FOR_REMOTE_HOST

Open Virtual Machine Format

D

Open Virtualization Format (OVF) is a relatively new industry standard for describing virtual machines in XML format. Companies that contributed to the standard include Dell, HP, IBM, Microsoft, VMware, and XenSource. As VMware increases its support for this standard, partners are encouraged to develop solutions that incorporate OVF.

The OVF specification describes a secure, portable, efficient, and flexible method to package and distribute virtual machines and components. It originated from the Distributed Management Task Force (DMTF) after vendor initiative. See the Virtual Appliances section of the VMware Web site for an introduction:

http://www.vmware.com/appliances/learn/ovf.html

OVF includes a mechanism for describing virtual disks.

OVF Tool

VMware currently provides the OVF Tool, a graphical user interface that allows third parties to create OVF images. See the Communities section of the VMware Web site for the user's guide:

http://www.vmware.com/resources/techresources/1013

A similar OVF packaging method is included with recent versions of ESX/ESXi.

OVF Library

At some point, an OVF library will be packaged for use with the VMware Virtual Disk Development Kit and other VMware development platforms.

Glossary

D

differential backup
Saving system data changed since the last full backup, so only two restore steps are necessary.

E

extent
In the context of VMDK, a split portion of virtual disk, usually 2GB.

F

flat
Space in a VMDK is fully allocated at creation time (pre-allocated). Contrast with sparse.

H

hosted disk
A virtual disk stored on a hosted product, such as VMware Workstation, for its guest operating system.

I

incremental backup
Saving system data changed since the last backup of any type.

M

managed disk
A virtual disk managed by an ESX/ESXi host or VMware vCenter, contained within a vStorage VMFS volume.

monolithic
The virtual disk is a single VMDK file, rather than a collection of 2GB extents. Contrast with split.

S

sparse
Space in a VMDK is allocated only when needed to store data. Contrast with flat.

split
The virtual disk is a collection of VMDK files containing 2GB extents. Contrast with monolithic.

Index

V

W