NTFS Curiosities (part 2): Volumes, volume names and mount points

Adi Oltean 17 Apr 2005 1:51 AM 14

In the old days of MS-DOS and Windows 95 you could identify any volume by its drive letter. Things were so easy. But not anymore after we shipped Windows 2000. In W2K you could also have mount points - volumes mounted under certain directories. This feature is similar with the single-root file system namespaces in Unix. The important thing is that mount points behave like normal directories - you can move them around or rename them from explorer. You won't see any significant differences when working with them - a volume mount point appears just as a normal folder. To quickly play with mount points, you can use the mountvol command. This is a simple command-line utility that allows us to create, list or delete mount points. Under the cover, this utility is using the SetVolumeMountPoint API to create a mount point.

Mount points are actually implemented using the NTFS reparse point technology. This allows you to assign a directory to any volume in the system, even to a removable one (so you could create a c:\cdrom folder for your CDROM).

But at this point you might wonder how we can identify a volume in the system, given that we have so many variations of it? The key concept is the **volume name**, a special persistent name assigned to each volume in the system. The format of the volume name is \\?\Volume{GUID}\ and you can see the volume names for every volume in your system if you just run MOUNTVOL without parameters.

What is a volume name?

A volume name gets assigned first time the OS sees the volume in the system. There is an internal OS component called Mount point Manager (in short, MpM) implemented in mountmgr.sys. This driver maintains a persistent database of volumes in the system. In the case of basic disks, the database associates a volume name with a certain associated with a certain partition on a given disk. Since the disks are reliably identified after reboot based on their signatures, MpM can reliably "discover" each volume during boot, and assign it the correct volume name.

A volume name is a MS-DOS device, as you could probably see from the fact that it has a \\?\ prefix. MS-DOS devices are nothing special. They are simply symbolic links that reside in the \\?\ Object Manager namespace, which usually point out to a real device located in the \Device namespace. What makes MS-DOS devices special is that they can be easily accessed from any process in user mode. In other words, you could simply call CreateFile on the volume GUID name, if you need to. You cannot call CreateFile from user mode on a regular device in the form of "\Device\HarrdiskVolume23".

As I said above, being an MS-DOS device, the volume name is just a symbolic link which points back to a real volume device, usually in the form of \Device\HarddiskVolume23. There is another example of an MS-DOS device which is the drive letter. If your volume has the C: drive letter, you will then have a symbolic link called \\?\C: which points to a real volume in the \Device\HarddiskVolumeXX format.

The real" device mentioned above (which I'll call it the "legacy device") is in the form of \Device\HarddiskVolumeXX and is implemented by another component in the operating system called the Volume Manager. There are two volume managers in the OS - the basic volume manager implemented by ftdisk.sys, and the dynamic volume manager implemented in dmio.sys. These are bus drivers that create volume devices as necessary, for example when you create a partition on a new disk.

Wow! So we already have at least three types of devices for a volume: the drive letter MS-DOS device name, the volume name and the volume device name. And that's not all - you can create as many MS-DOS devices you want for a volume through the DefineDosDevice API. But note that these device names that you create with DefineDosDevice are not persistent - they will go away after a reboot.

Translations

Now, given a certain drive letter, or a volume mount point, how I can get the underlying legacy device? The trick is to use the QueryDosDevice API. This API must be used in the following way. First, let's say that your DOS device is \\?\Volume{4bcddd95-9e9e-11d6-b7f4-806e6f6e6963}\. You strip out the \\?\ prefix and the terminating backslash (if any). This way you get to the real DOS device which is "Volume{4bcddd95-9e9e-11d6-b7f4-806e6f6e6963}". Same thing for a drive letter. There, the real DOS device is "C:". Now, you feed this device name as the first parameter to QueryDosDevice and you obtain the legacy volume device. Done!

But in practice, you rarely need to deal with legacy devices. It is much more useful to convert a drive letter or a mount point path to a volume name. The operating system provides a convenient function that does just this, called GetVolumeNameForVolumeMountPoint. By the way, I haven't verified, but this must be probably the Win32 API with the longest name!

One more note - the GetVolumeNameForVolumeMountPoint API works only on a path which is either a volume drive letter or the root of a volume mount point. What if you want to get the underlying volume root path for a random path? You need to use the GetVolumePathName which returns the nearest volume root path for a given directory. For example if you have a path like C:\foo\bar\somefolder, and both C:\foo and C:\foo\bar are mount points, then the nearest root is C:\foo\bar.

Enumerations

In Windows 2000 you can enumerate all volumes in the system with FindFirstVolume/FindNextVolume. These APIs are enumerating all the underlying volumes that the MpM knows about.

However, this only enumerates the volume names, not the actual drive letters and paths. Enumerating all volumes in the system can be confusing, given that the same volume might end up having both a drive letter and a mount point at the same time! For example you can have a volume mounted under C:\ and another one under C:\foo\, and another one under C:\foo\bar\. In Windows XP and Windows Server 2003, there is a new API called GetVolumePathNamesForVolumeName which allows you to enumerate all the "display names" of a volume, i.e. the drive letter (if any) and any mount points. Note that you can have volume names with no drive letters or mount points too.

You can also enumerate all mount points on a certain "parent" volume using FindFirstVolumeMountPoint/FindNextVolumeMountPoint. How the OS does that? Normally, you would think that the API enumerates all the directories in that volume and find out what are the mount points. That would be terribly slow, so the OS provides an optimization. On each volume there is a NTFS stream called \$Extend\$Reparse which keeps the list of "child" volume mount points defined on a certain volume. (For more details on this stream or other predefined NTFS streams you might consult the wonderful Microsoft Windows Internals book from Dave Solomon)

In practice, things can get little more complicated if you want to recursively enumerate all the directories **under** a certain path. Say that you want to recursively copy all files under C:\foo to C:\bar. Sounds easy, isn't it? Wrong. What if c:\foo\dir1\ is a mount point? Well, that's not a problem since as I said below, volume mount points look like normal folders to the shell so FindFirstFile/FindNextFile will have no problem enumerating their contents. But - wait - what if we have a **cyclic** volume mount point? In other words, c:\foo\dir1\ is the same volume as C:. In that case you can reach an infinite cycle, which will break at some point anyway since Win32 APIs will work with directory names with less than MAX_PATH (255 characters).

I prepared a quick test below from which we can easily see that even the DIR command gets confused. Oops!

```
P:\>dir /s /b p:\
p:\foo
p:\test
p:\foo\dir1
p:\foo\dir1\foo
p:\foo\dir1\test
p:\foo\dir1\foo\dir1
p:\foo\dir1\foo\dir1\foo
p:\foo\dir1\foo\dir1\test
p:\foo\dir1\foo\dir1\foo\dir1
p:\foo\dir1\foo\dir1\foo\dir1\foo
p:\foo\dir1\foo\dir1\foo\dir1\test
p:\foo\dir1\foo\dir1\foo\dir1\foo\dir1
...
```

So things are not that easy. Consider also that you might end up in a situation where c:\foo\dir1 is one volume which contains another volume mount point named c:\foo\dir1\dir2 which contains another mount point which points back at C:, etc. Solving these enumerations the right way is not an easy task, and is left as an exercise to the reader...

One more thing about FindFirstFile/FindNextFile - you can actually use these APIs to see whether you are in the context of a mount point (although, you can alternatively use the more expensive GetVolumePathNames API to do this). Just get the attributes of a directory with GetFileAttributes. If this directory has the FILE_ATTRIBUTE_REPARSE_POINT attribute, then it holds a mount point.

A final observation about volume mount points is that certain file system operations don't work the right way when you have volume mount points. For example if, in explorer, you recursively restrict access on the C: to everyone except local administrators, you might think that you are safe. You are not, actually if C:\foo is a mount point, since the NTFS access control inheritance rules **won't propagate** across volume mount points! So, c:\foo will be widely open to everyone even if you though that you tighten the control on C:\. What you have to do is to recursively enumerate all mount points under C:\ and re-apply the security settings on all underlying volumes.

Are volume names really unique?

There is another quirk on volume names that might interest you. Although the GetVolumeNameForVolumeMountPoint is supposed to return the unique volume name, it doesn't work as expected in certain cases. In rare cases, a volume can end up having two volume names! This weird situation can happen if you migrate a disk from one computer to another. Another scenario is dual boot - remember that in a dual boot, the OS sees the disks of the other stopped instance of the operating system, so it "thinks" as if these disks were moved from one machine to another.

So what is the actual problem? Remember the \$Extend\$Reparse NTFS stream above? That stream contains a list of (folder - volume GUID) associations. In other words, if your C:\ volume contains a mount point at C:\foo, then this stream contains an association between the "\foo" relative path and the volume GUID of the c:\foo volume. Now, let's assume that you moved both C:\ and C:\foo volumes from one machine to another. The MpM will assign a **new** volume GUID to the C:\foo volume. Later, when the C:\ volume also gets surfaced, the C:\foo is present in that enumeration with a different volume GUID. MpM reacts to this by assigning a new volume name to

C:\foo, in order to be consistent with the \$Extend\$Reparse stream contents.

Bottom line is: if you have disk/volume migrations, then you can expect multiple volume names for the same volume. But whatever it happens, there is always a unique volume name for the current boot session. You can obtain this unique name by calling GetVolumeNameForVolumeMountPoint once on your root, get the volume name, and then call GetVolumeNameForVolumeMountPoint again. This will always return the unique volume name.

This trick is very useful for example when you want to check if two volume paths V1 and V2 represent the same volume or not. You get the first volume path (V1), call GetVolumeNameForVolumeMountPoint on it twice in the manner described above, and remember the returned volume name. You do the same thing on V2. In the end, you compare the volume names. If they are equal, then the two volumes are identical.

Hidden volumes

Although this post got rather long, I should mention one more thing. There are certain categories of volumes that are not visible to the MpM.

One class is hidden volumes. These volumes have the special property that no PnP notifications are being sent on arrival. In addition, these volumes do not have volume names (because anyway MpM doesn't know about them). Hidden volumes are used for the VSS Hardware Shadow Copies infrastructure.

Another class of volumes that do not have a volume name are VOLSNAP shadow copy devices. These are devices created by the VOLSNAP.SYS driver in the form \Device\HarddiskVolumeShadowCopyXXX. These devices are not even managed by the volume manager, and no PnP volume arrival notifications are being sent on their arrival. Again, in this case, MpM won't assign a volume name.

Comments

oshah 17 Apr 2005 6:17 AM #
On the subject of cyclic mount points,
you can also get the same thing if your program walks down shell links (shortcuts). One program I'm pointing the finger at is Winamp.

PS ConvertStringSecurityDescriptorToSecurityDescriptor() is longer!

Random Reader 17 Apr 2005 7:01 AM #
Just checking whether a directory has the FILE_ATTRIBUTE_REPARSE_POINT attribute won't necessarily tell you if you're in a mount point though, right? The reparse tag could belong to a Junction, Remote Storage, or a third-party filter. You'd also want to check that dwReserved0 == IO_REPARSE_TAG_MOUNT_POINT.

Reuven 17 Apr 2005 12:27 PM #
The requirement to doubly call GetVolumeNameForVolumeMountPoint twice before is one that few people know about, and causes subtle bugs. Actually, this doesn't completely fix the problem. The mount manager stores the list of volume names in the registry, and it chooses the alphabetically earliest one as its the canonical name. If a new volume name shows up after you call GetVolumeNameForVolumeMountPoint twice, things still won't work!

What to do about this? Well you could ignore it. Calling GetVolumeNameForVolumeMountPoint twice almost completely fixes the problem; a new volume name showing up in that window is highly unlikely. Another possibility is to call GetDosDevice on both volumes and compare the device names. The device names will always match if the volume is the same, making this a safe comparison.

Adi Oltean 17 Apr 2005 7:20 PM #
>>> Just checking whether a directory has the FILE_ATTRIBUTE_REPARSE_POINT attribute won't necessarily tell you if you're in a mount point though, right? The reparse tag could belong to a Junction, Remote Storage, or a third-party filter. You'd also want to check that dwReserved0 == IO_REPARSE_TAG_MOUNT_POINT.

You are absolutely right.

Roland Kaufmann 18 Apr 2005 10:57 AM #
While we're at the subject; it would be interesting to know what "magic" that must be performed first to allow to specify a native path under \Device\LanmanRedirector as a reparse target to DeviceIoControl().

Adi Oltean 18 Apr 2005 5:38 PM #

>>> While we're at the subject; it would be interesting to know what "magic" that must be performed first to allow to specify a native path under \Device\LanmanRedirector as a reparse target to DeviceIoControl().

What do you mean by "reparse target to DeviceIoControl"? DeviceIoControl is just a method to send IOCTLs/FSCTLs to any device in the system.

ms007 4 May 2005 3:59 PM #

Does anyone know how to pull the volume information for a hidden, or
unidentified volume? I'm mainly looking for the volume label, but
will
also need the disk extents to match them up.

Disk Management views this partition as a Healthy (Unknown Partition),
and also identifies the volume label. I can't see that
GetVolumeInformation works on anything but a Drive Letter or Volume
GUID, or am I mistaken? Any help figuring out how to read the volume
identifying information from a partition would be greatly appreciated!

Adi Oltean 4 May 2005 4:44 PM #

>>> Disk Management views this partition as a Healthy (Unknown Partition), and also identifies the volume label. I can't see that GetVolumeInformation works on anything but a Drive Letter or Volume GUID, or am I mistaken? Any help figuring out how to read the volume identifying information from a partition would be greatly appreciated!

You might want to use the WMI API to enumerate all partitions on a disk, and get their properties. See the last sample code at

http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_tasks__disks_and_file_systems.asp for more details...

Mike Cooper 4 May 2005 7:13 PM #

Any idea how to get a Volume Name (\\?\Vol{GUID}) or drive letter for a given disk XX + partition XX? i.e. If I have a partition table for DiskDrive0 I need to find out what volume/drive letter is on DiskDrive0 Partition 1. I can't seem to find any API's or registry data which gets me there.

Adi Oltean 4 May 2005 9:30 PM #

>>> Any idea how to get a Volume Name (\\?\Vol{GUID}) or drive letter for a given disk XX + partition XX? i.e. If I have a partition table for DiskDrive0 I need to find out what volume/drive letter is on DiskDrive0 Partition 1. I can't seem to find any API's or registry data which gets me there.

I added the response here:

http://blogs.msdn.com/adioltean/archive/2005/05/04/414806.aspx

余啊雷 3 Apr 2008 2:12 PM #

PingBack from http://drinksandbirthdaysblog.info/antimail-ntfs-curiosities-part-2-volumes-volume-names-and-mount/

余啊雷 4 Apr 2008 5:03 PM #

PingBack from http://drinksairportsblog.info/antimail-ntfs-curiosities-part-2-volumes-volume-names-and-mount/

余啊雷 28 May 2008 4:25 PM #

PingBack from http://brainrack.wordpress.com/2008/05/28/broken-and-ill-documented-api-for-windows-mount-points/

余啊雷 29 Jun 2008 7:04 PM #

PingBack from http://ezra.sextalessite.com/whatisavolumelabelforntfs.html