# How do I get from a volume to the physical disk that holds it?

devblogs.microsoft.com/oldnewthing/20201021-00

October 21, 2020

Raymond Chen

Last time, we saw how to get from a file path to the volume that holds it. The next step is to get from the volume to the physical disk.

The lazy way is to ask for the device number:

```
STORAGE_DEVICE_NUMBER number;
DeviceIoControl(handle,
    IOCTL_STORAGE_GET_DEVICE_NUMBER,
    nullptr, 0, // no input
    &number, sizeof(number), // output goes here
    &bytesWritten,
    nullptr);
DWORD physicalDriveNumber = number.DeviceNumber;
```

This is lazy for multiple reasons:

- It fails to account for the case where the volume spans multiple physical drives.
- In my experience, if the volume is a CD-ROM drive with no disk in the drive, the call reports that the physical drive number is 0, which is almost certainly incorrect.

In practice, it seems that if the volume spans multiple physical drives, the `IOCTL_ STORAGE_ GET_ DEVICE_ NUMBER` fails (with `ERROR_ INVALID_ FUNCTION` , it seems, which is the Win32 manifestation of the NT status code `STATUS_ INVALID_ DEVICE_ REQUEST` ), so at least you don't get *wrong* answers. You just get *no* answer.

The less lazy (and more likely to be correct) way is to ask the volume for its disk extents. This one is a bit annoying because it returns a variable-sized structure, so you need to ask twice. The first time tells you how big a structure you need, and the second time actually gets the structure.

Since nearly all volumes have only one extent, we can optimize slightly for that case by passing an initial buffer big enough to hold a single extent. If that works, then there's no need to try a second time.

```
VOLUME_DISK_EXTENTS* extents = nullptr;

// Anticipate common case where there is only one extent.
VOLUME_DISK_EXTENTS singleExtent;

// But also have a place to manage allocated data.
std::unique_ptr<BYTE[]> lifetime;

DWORD bytesWritten;
if (DeviceIoControl(handle, IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS,
        nullptr, 0,
        &singleExtent, sizeof(singleExtent),
        &bytesWritten,
        nullptr)) {
  // Worked on the first try. Use the preallocated buffer.
  extents = &singleExtent;
} else {
  VOLUME_DISK_EXTENTS* lastQuery = &singleExtent;
  while (GetLastError() == ERROR_MORE_DATA) {
    assert(RTL_CONTAINS_FIELD(lastQuery, bytesWritten, NumberOfDiskExtents));
    DWORD extentCount = lastQuery->NumberOfDiskExtents;
    DWORD allocatedSize = FIELD_OFFSET(VOLUME_DISK_EXTENTS, Extents[extentCount]);
    lifetime.reset(new BYTE[allocatedSize]);
    lastQuery = (VOLUME_DISK_EXTENTS*)lifetime.get();
    if (DeviceIoControl(handle, IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS,
          nullptr, 0,
          lastQuery, allocatedSize,
          &bytesWritten,
          nullptr)) {
      extents = lastQuery;
      break;
    }
  }
}

if (extents) {
  // process the extents
}
```

The extents tell you which physical drives the volume draws its storage from, and which bytes on those physical drives are devoted to the volume. But for this exercise, we just want the physical drives.

Once you have the physical drive numbers, you can convert them to physical drive handles by building a path of the form `\\.\PhysicalDrive#` where the `#` is the decimal expansion of the drive number.

```
wchar_t physicalDrivePath[80];
wsprintf_s(physicalDrivePath, L"\\\\.\\PhysicalDrive%d", physicalDriveNumber);
driveHandle = CreateFile(physicalVolumePath,
        0, FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
        nullptr, OPEN_EXISTING, 0, nullptr);
```

Okay, great, now you have a physical drive handle.

Next time, we'll see that there's a shortcut available for all this.

**Bonus chatter**: If you are interested only in the first physical drive of a multi-drive volume, you can do it much more simply, because the ioctl will fill in as much of the buffer as it can. Passing a buffer that can hold one physical drive will give you the first physical drive. (Mind you, the drives don't appear to be in any particular, order, so really, you're just grabbing one at random.)

```
wil::unique_hfile GetFirstPhysicalDiskHandleForVolume(HANDLE volume)
{
  VOLUME_DISK_EXTENTS extents;
  if (!DeviceIoControl(volume, IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS,
        nullptr, 0,
        &extents, sizeof(extents),
        &bytesWritten,
        nullptr) && GetLastError() != ERROR_MORE_DATA) {
    THROW_LAST_ERROR();
  }

  wchar_t physicalDrivePath[80];
  swprintf_s(physicalDrivePath, L"\\\\.\\PhysicalDrive%u",
             extents.Extents[0].DiskNumber);

  wil::unique_hfile result{ CreateFile(physicalDrivePath, 0,
                      FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
                      nullptr, OPEN_EXISTING, FILE_FLAG_BACKUP_SEMANTICS, nullptr) };
  THROW_LAST_ERROR_IF(!result);
  return result;
}
```

**Bonus bonus chatter**: It seems that the I/O subsystem can't decide whether the number is a physical *device* number, a physical *disk* number, or a a physical *drive* number.

Raymond Chen

**Follow**