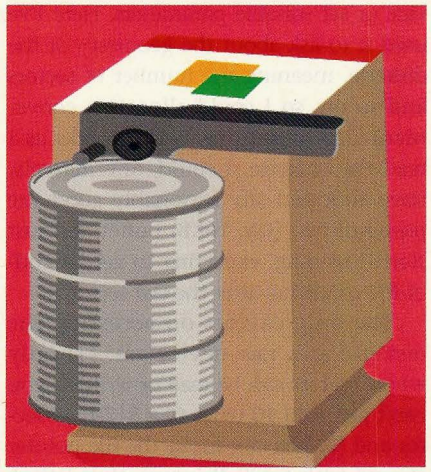
Douglas A. Hamilton

Getting at Your Raw Devices



SCOTT BALDWIN

UNDER DOS OR even OS/2, getting at a raw device—for example, being able to read or write the low-level sectors on a diskette—isn't easy. The programming interface is typically disjointed from other kinds of file I/O. By contrast, NT follows a style more like UNIX's, in which raw devices can be opened by specifying the right name in the filename space and, once the file is open, reading and writing can be done with ordinary ReadFile() and WriteFile() calls.

Under NT, devices are accessed through a pseudo-Universal Naming Convention (UNC) style of prepending \\.\ to the device name, where the dot means *this machine*. To open the diskette, just open \\.\a:. In concept, this is similar to the \dev directory in UNIX, although the analogy isn't perfect. For example, you can't simply list the \\.\ directory to see what devices are available the same way you might list \dev on a UNIX machine; apparently, you just have to know what is there. Also, the use of a UNC style suggests that one ought to be able to access devices on other machines over the network, but, in my experiments, that doesn't work. For example, from my PS/2 (or even from my Olivetti/Mips machine), I expected to be able to open \\mips\a: but got return codes indicating I had tried a bad network name.

But, assuming you know the name of your local device, reading or writing it is pretty straightforward, as we'll demonstrate here with a pair of simple utilities that will let you mass-duplicate a diskette by first reading a whole diskette image onto your hard disk and then writing it back out, formatting as necessary. Here is the first one, diskrd.c, which does the reading, copying to stdout:

```
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>

void error( char *msg )
  {
  DWORD rc;
  fprintf(stderr, "Error:  %s, rc = %d\n",
    msg, rc = GetLastError());
  ExitProcess(rc);
  }

void main( int argc, char **argv )
  {
  HANDLE disk, Stdout;
  DISK_GEOMETRY geometry;
  DWORD cylinder, head, bytes,
    bufsize;
  char devicename[] = "\\\\.\\a:",
    *buffer;

  /* Default is to read the a: drive,
    but user may specify any other
```

```
    drive if desired. */
  if (argc == 2)
    devicename[4] = *argv[1];

  /* Open the drive for reading */
  if ((disk = CreateFile(devicename,
      GENERIC_READ,
FILE_SHARE_READ,
      NULL, OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL,
      NULL)) ==
        INVALID_HANDLE_VALUE)
    error("Couldn't open drive");

  if (!DeviceIoControl(disk,
      IOCTL_DISK_GET_DRIVE_GEOME-
TRY,
      NULL, 0, &geometry,
      sizeof(geometry), &bytes,
      NULL))
    error("Couldn't get drive parameters");

  /* Allocate a buffer equal to one
    track's worth of data. */
  bufsize = geometry.BytesPerSector *
    geometry.SectorsPerTrack;
  buffer = malloc(bufsize);

  /*Get the Stdout handle */
  Stdout =
GetStdHandle(STD_OUTPUT_HANDLE);

  /* Walk through each of the heads
    on all the cylinders, copying
    tracks to stdout. */
  for (cylinder = 0; cylinder++ <
      geometry.Cylinders.LowPart; )
    for (head = 0; head++ <
        geometry.TracksPerCylinder; )
      {
      if (!ReadFile(disk, buffer,
          bufsize, &bytes, NULL) ||
          bytes != bufsize)
        error("ReadFile failed");
      if (!WriteFile(Stdout, buffer,
          bufsize, &bytes, NULL))
        error("WriteFile failed");
      }

  CloseHandle(disk);
  CloseHandle(Stdout);
  ExitProcess(0);
  }
```

To read a disk image in from drive A: and save it as dskimage on your hard disk, type diskrd > dskimage. (Alternatively, you can read the B: drive simply by giving that as an argument.)

Notice that in diskrd.c, I've opened the drive with OPEN_EXISTING. There is a reason for that: It works, and the other choices, such as OPEN_ALWAYS, don't. (The argument presumably is that the device must already exist or you can't open it; you can't just create a new one.)

Also, I've introduced the DeviceIoControl() call here. DeviceIoControl() operations (listed in winioctl.h) let you ask questions about a device or set suitable parameters. Here I've used it to ask about the geometry of the diskette, meaning the number of sectors and so on, so I could allocate a convenient-size buffer and count tracks as I read the diskette. I could, alternatively, have allocated any reasonably good-size power-of-two (say, 16KB) buffer and just started reading, expecting to get an end-of-file condition after the last sector.

The only caveats on accessing the diskette as a raw device are: (1) only whole sectors can be read or written; you can't just read 10 bytes of a 512-byte sector and (2) you can move the file pointer around with SetFilePointer(), thus letting you randomly access individual sectors, although you can position it only on sector boundaries.

Now here's the corresponding diskwrt.c utility for writing diskettes, skipping the header includes and the error() routine definitions that are the same:

```
void main( int argc, char **argv )
  {
  HANDLE disk, Stdin;
  DISK_GEOMETRY geometry;
  MEDIA_TYPE media;
  FORMAT_PARAMETERS fmt;
  BOOL format_reqd;
  DWORD cylinder, head, bytes,
    bufsize;
  char devicename[] = "\\\\.\\a:",
    *buffer;

  if (argc == 2)
    devicename[4] = *argv[1];

  /* Turn off popups for unformatted
    media. */
  SetErrorMode(SEM_FAILCRITICALER-
RORS);

  /* Open the drive for both
    writing and reading. (If the
    diskette's not formatted,
    we'll need read access to
    determine what media types
    are supported by this
    drive.) */
  if ((disk = CreateFile(devicename,
      GENERIC_WRITE | GENERIC_READ,
      0, NULL, OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL, NULL))
==
        INVALID_HANDLE_VALUE)
    error("Couldn't open drive");

  if (!DeviceIoControl(disk,
      IOCTL_DISK_GET_DRIVE_GEOME-
TRY,
      NULL, 0, &geometry,
      sizeof(geometry), &bytes, NULL))
    error("Couldn't get drive parameters");

  if (format_reqd =
      geometry.MediaType == Unknown)
    {
    /* Diskette isn't formatted.
      Determine what media formats
      are supported, pick the highest
      "standard" density for this
      drive and get set to format
      each track as we go. */
    DISK_GEOMETRY g[10];
    int i, highest;
    if (!DeviceIoControl(disk,
        IOCTL_DISK_GET_MEDIA_TYPES,
        NULL, 0, g, sizeof(g),
        &bytes, NULL))
      error( "Couldn't determine "
        "supported media types");
    for (i = highest = 0;
        i*sizeof(DISK_GEOMETRY) < bytes;
        i++)
      if (g[i].MediaType <
          g[highest].MediaType)
        highest = i;
    geometry = g[highest];
    fmt.MediaType =
      geometry.MediaType;
    fmt.StartHeadNumber = 0;
    fmt.EndHeadNumber =
      geometry.TracksPerCylinder - 1;
    printf("Formatting diskette "
      "as media type = %d\n",
      fmt.MediaType);
    }

  /* Allocate a buffer equal to
```

```
    one track's worth of data. */
  bufsize = geometry.BytesPerSector *
    geometry.SectorsPerTrack;
  buffer = malloc(bufsize);

  /*Get the Stdin handle */
  Stdin = GetStdHandle(STD_INPUT_HAN-
DLE);

  /* Walk through each of the heads
    on all the cylinders, copying
    data to the floppy. */
  for (cylinder = 0; cylinder <
      geometry.Cylinders.LowPart;
      cylinder++)
    {
    if (format_reqd)
      {
      BAD_TRACK_NUMBER badtracks[2];
      fmt.StartCylinderNumber =
        fmt.EndCylinderNumber =
        cylinder;
      if (!DeviceIoControl(disk,
          IOCTL_DISK_FORMAT_TRACKS,
          &fmt, sizeof(fmt),
          badtracks,
          sizeof(badtracks),
          &bytes, NULL) ||
            bytes != 0)
        error( "Couldn't format "
          "the disk");
      }

    for (head = 0; head++ <
        geometry.TracksPerCylinder; )
      {
      if (!ReadFile(Stdin, buffer,
          bufsize, &bytes, NULL) ||
          bytes != bufsize)
        error("ReadFile failed");
      if (!WriteFile(disk, buffer,
          bufsize, &bytes, NULL))
        error("WriteFile failed");
      }
    }

  CloseHandle(disk);
  CloseHandle(Stdin);
  ExitProcess(0);
  }
```

To write out a diskette using the data stored in dskimage, type diskwrt < dskimage. If the diskette isn't already formatted, it will automatically be formatted on the fly, one cylinder at a time as the data is written. To mass-duplicate a diskette, just keep rerunning diskwrt, writing out the same image on a new diskette each time.

In diskwrt.c, I've turned off the pop-ups that normally would appear if I tried opening an unformatted diskette using SetErrorMode(). Also, I requested read, as well as write access, because if we do encounter an unformatted diskette, we can ask what media formats the drive supports. For some reason, you can't do that with just a write handle.

The way to tell if you've got an unformatted diskette is to look at the geometry. If the media type and/or all the other parameters (number of sectors, sector size and so on) are zero, it's unformatted. Deciding what format to use in that case is a little sloppy. The GET_MEDIA_TYPES operation returns a list of all the media formats the drive supports.

There is no way to ask beforehand how many formats that might be, so you just have to pass a buffer you think is bigger than could ever be needed and see what comes back. Also, the list isn't, to my knowledge, in any particular guaranteed order, nor is there an indication of what the preferred, or default, format is for that drive, so you just have to look down the list and pick one. In this example, I've just chosen the lowest-numbered media type, which preferentially picks 1.44MB on 3.5-inch and 1.2MB on 5.25-inch media.

If you've tried writing a program to format a diskette under DOS or OS/2, you'll certainly like the simplification here. Just tell the FORMAT_TRACKS operation what media type you want and the starting and ending heads and cylinders. If any bad tracks are encountered, they'll be listed in a buffer that is returned. By formatting as we go, one cylinder at a time, rather than completely formatting the whole diskette at once, we get a bump in performance, since the heads have to be stepped across to each cylinder only once, not twice. From there, writing to the diskette is simply the inverse of reading it. ■

Douglas Hamilton is president of Hamilton Laboratories (Wayland, Mass.) and author of the Hamilton C Shell, an advanced interactive command processor and tool package for OS/2 and Windows NT. Reach Douglas on WIX as hamilton or care of Editor at the address on page 16.