Windows Disk Drive Recovery with Ada95 – An Application Note

Karl Nyberg
Grebyn Corporation
P. O. Box 47
Sterling, VA 20167-0047
1-703-406-4161
karl@nyberg.net

ABSTRACT

This note is a little documentary case of using Ada95 to develop a small utility to recover data from a FAT (File Allocation Table) formatted file system on a Windows 98 disk drive.

1. INTRODUCTION

A friend of mine mentioned that he had been installing an additional disk drive on his computer and in the process had somehow lost everything he had on his original disk. (I later found that a worm targeting his firewall security software, BlackICE, may have actually been responsible for the damage.) I offered to take a look at it and see if I could recover any data off of the drive, having had a modicum of success in doing similar things before, particularly with Solaris drives. In the process, I created an Ada interface to the underlying format of the file system and herein describe that format in Ada95 terms and its use in recovering the data.

2. Window 98 File System - FAT32

The drive in question was formatted using Microsoft FAT32 (File Allocation Table - 32 bits, there are also 12 bit and 16 bit versions, where the number refers to the size of the table), formally documented in the "Microsoft Extensible Firmware Initiative FAT32 File System Specification" [FAT32]. The remainder of this section describes that file format.

A FAT file system consists of four regions:

1. Reserved Region (Boot Sector)
2. File Allocation Table Region
3. Root Directory Region (not present on FAT32)
4. Directory and File Data Region

2.1 Boot Sector

The boot sector is the first sector of data on the disk. Generally, this is the first 512 bytes (stored in BPB_BytsPerSec, a variable name, as all shown below, chosen to mimic the specification), but, we are cautioned in the documentation, this isn't necessarily so...

The various fields in the Boot_Sector record shown below (types are System.Unsigned_Types from GCC 3.4 20031210) have to do with such issues as the drive's geometric characteristics as well as the type of file system stored on the drive. Separate structures are stored on drives formatted for other formats (FAT12 and FAT16), but were not defined and are not described here, as they were not of interest.

The data on the drive being recovered in the boot sector had been damaged in some manner so that this information was useless during recovery. Data from other drives was read to understand how the data fit together, what data was actually necessary in order to recover data from the drive and to confirm that the algorithm worked on valid data.

```
type Boot_Sector is record
   BS_JmpBoot      : String (1 .. 3);
   BS_OEMName      : String (1 .. 8);
   BPB_BytsPerSec  : Short_Unsigned       := 0;
   BPB_SecPerClus  : Short_Short_Integer  := 0;
   BPB_ResvdSecCnt : Short_Unsigned       := 0;
   BPB_NumFATs     : Short_Short_Integer  := 0;
   BPB_RootEntCnt  : Short_Unsigned       := 0;
   BPB_TotSec16    : Short_Unsigned       := 0;
   BPB_Media       : Short_Short_Integer  := 0;
   BPB_FATSz16     : Short_Unsigned       := 0;
   BPB_SecPerTrk   : Short_Unsigned       := 0;
   BPB_NumHeads    : Short_Unsigned       := 0;
   BPB_HiddSec     : Unsigned             := 0;
   BPB_TotSec32    : Unsigned             := 0;
   BPB_FATSz32     : Unsigned             := 0;
   BPB_ExtFlags    : Short_Unsigned       := 0;
   BPB_FSVer       : Short_Unsigned       := 0;
   BPB_RootClus    : Unsigned             := 0;
   BPB_FSInfo      : Short_Unsigned       := 0;
   BPB_BkBootSec   : Short_Unsigned       := 0;
   BPB_Reserved    : String (1 .. 12);
   BS_DrvNum       : Short_Short_Integer  := 0;
   BS_Reserved1    : Short_Short_Integer  := 0;
   BS_BootSig      : Short_Short_Integer  := 0;
   BS_VolID        : Unsigned             := 0;
   BS_VolLab       : String (1 .. 11);
   BS_FilSysType   : String (1 .. 8);
end record;
```

2.2 File Allocation Table Region

The File Allocation Table (FAT) Region is a number of entries that indicate what locations on the disk contain data for individual items (be they directories or files). Each entry in the FAT contains either the location of the following FAT entry or a value (mod 2 ** 28 = 16#FFFFFFF# or 16#FFFFFF8#) that indicates this entry is the final one for this data item (directory or file). Each data item referred to contains one cluster's worth of data.

Generally, for a defragmented (optimized layout for sequential access, not necessarily for disk geometry access) file system,

entries will be sequential - a data item stored at location 10 .. 12 will have FAT entries that appear as:

FAT Location	Value
10	11
11	12
12	16#FFFFFFF#

There are two copies of the FAT stored on the disk, one following the other. While the documentation doesn't say so, it appeared from analysis that the entries for the two copies are not identical. What appears to be the case is that when the two entries differ, one will have the value 16#76F676F6# and the other will have the correct value. Since the FAT was being used as a read-only mechanism to restore the data here rather than as a general-purpose file system, this issue wasn't investigated further.

2.3 Root Directory Region (not on FAT32)

2.4 Directory and File Data Region

The remainder of the disk contains the data for the directory entries and the actual contents of files stored in the directories.

2.4.1 Directory Structure

The directory structure consists of two types of entries - short names (sometimes also known as the 8.3 filenames, from the number of characters available for the name and the extension) and long filenames. The file system is designed so that all names are unique with short filenames, even though long filenames may not be unique to eight characters with the same extension.

2.4.1.1 Short File Names - 8.3

For the current application, which was merely restoring the files, only the name, location of the first cluster (high and low bits) and the file size were utilized.

```
type Directory is record
   DIR_Name          : String (1 .. 11);
   DIR_Attr          : Short_Short_Integer  := 0;
   DIR_NTRes         : Short_Short_Integer  := 0;
   DIR_CrtTimeTenth  : Short_Short_Unsigned := 0;
   DIR_CrtTime       : Short_Unsigned       := 0;
   DIR_CrtDate       : Short_Unsigned       := 0;
   DIR_LstAccDate    : Short_Unsigned       := 0;
   DIR_FstClusHI     : Short_Unsigned       := 0;
   DIR_WrtTime       : Short_Unsigned       := 0;
   DIR_WrtDate       : Short_Unsigned       := 0;
   DIR_FstClusLO     : Short_Unsigned       := 0;
   DIR_FileSize      : Unsigned             := 0;
end record;
```

2.4.1.2 Long File Names

For directories (or files) with long filenames, these names are stored using multiple entries (all directory filename entries are 32 bytes long). The individual components of the filename (Name1, Name2, and Name3) are catenated together in sequential entries for each occurrence to make up the long filename.

```
type Long_Directory is record
   LDIR_Ord          : Short_Short_Integer := 0;
   LDIR_Name1        : String (1 .. 10);
   LDIR_Attr         : Short_Short_Integer := 0;
   LDIR_Type         : Short_Short_Integer := 0;
   LDIR_Chksum       : Short_Short_Integer := 0;
   LDIR_Name2        : String (1 .. 12);
   LDIR_FstClusLO    : Short_Unsigned      := 0;
   LDIR_Name3        : String (1 .. 4);
end record;
```

2.4.2 File Contents

Once the file names were determined, the corresponding data for the file was read. Again, this was performed in 8K data sectors, with the final write being a partial sector to complete the size of the file's data.

3. Program Execution

The utility was developed as two passes - the first one that simply listed the directories and files contained and the second one that actually recovered the data in the files. This was done as much for the iterative learning process as anything.

Both passes perform the same walking of the directory structure. In the first pass, a script was created with directory names were printed with "mkdir" commands and file names were printed with "touch". This script could be executed to create the appropriate populated directory tree for the second pass. (It was also useful during development, to confirm directory and file names, especially with embedded spaces and special characters.)

During the second pass, only the leaves of the tree (the files) received any additional processing. The file size was obtained from the directory record and the appropriate number of bytes copied.

4. Results

4.1 Development Approach

The development approach was essentially iterative – a little requirements analysis, a little design, a little coding, a little testing. Software for the first pass (parsing the directory structures) was developed until a full understanding of the format and contents of the drive were completed

4.2 Timing

The original disk drive tested was a 14GB drive, whose contents were copied from the existing drive to a much larger drive using the UNIX "dd" command. This was done so as to use the original drive as little as possible in the event that some parts of the drives components were failing and might not last long. Execution of the listing phase of the program to determine the directory structures and file names took about one minute. The file extraction phase required an additional twenty minutes. (All times were run on a Dell PowerEdge 400SC with a 3.2GHz Pentium processor under RedHat Linux 9.0 running kernel 2.4.20-8). As these times were considered "reasonable", no additional effort in improving the performance of the program was undertaken. For comparison, images from other file systems of approximately 2GB ran in 20 seconds for the listing phase and 3 – 4 minutes for the extraction phase. Times were somewhat dependent upon the mix between the number of directories and individual files contained within the images.

4.3 Effectiveness

Many times I've heard people say that they don't choose Ada because they're looking for something that will let them get their

little project done quickly and don't want to have the "overhead" that Ada requires. This project took a couple weeks of occasional evenings to generate about 1000 lines (including blanks, comments, etc.). Mind you, there's no 2167A documentation, users guide, man pages or attempts to meet or maintain any SEI Level N anything. It just worked.

5. Additional / Future Work

There are a number of areas where additional work could be done to extend the usefulness of this effort.

File timestamps (creation time / date, write time / date, last access date) were not used when writing the data. Doing so wasn't considered important here, since recovery of content was the issue. Adding such features would be straightforward, possibly requiring an interface to the underlying operating system's file structures and system calls. With such an interface, it might even be reasonable to expect to reduce the whole process to a single pass.

Support for other FAT types, namely FAT12 and FAT16, were not implemented because they weren't of current interest. These could easily enough be added with variant records within the Boot_Sector record.

Ada's Direct_IO package was used to read in the (8K byte) sectors. However, as the implementation used was limited in the file size, a "virtual" implementation was developed. This virtual file system split the recovered image into individual files of 500 megabytes (again, using the UNIX "dd" command) and creating an interface that selected the appropriate file and set the index accordingly. Bytes were then copied from these files into a temporary file for use. Another option would have been to modify the compiler to permit larger files with Direct_IO.

Some sort of interactive recovery program that would allow walking the directory tree and selecting files or one that would allow the specification of individual files or directories to be selected on the command line could also be added to the utility.

It might have been possible to recreate the information in the boot sector and try to get the drive to boot by itself. There is usually an additional copy stored at a secondary location in the disk drive, but in the case of interest here, the backup boot sector had also been overwritten. However, by this time my friend had already reloaded his operating system on another disk drive and was only interested in a few pieces of data.

The software is available at http://www.grebyn.com/software for those interested in using or modifying it.

6. REFERENCES

[1] http://www.microsoft.com/hwdev/download/hardware/fatgen103.pdf