QUANTA HELPLINE

Dilwyn Jones

Members wishing to submit helpline requests via email can use the email address helpline@quanta.org.uk or if you prefer to use traditional post, please send the helpline request to me via the address printed inside the front cover of the newsletter.

Obviously, we cannot guarantee to answer every query we receive, but we will do our best! Where we have been unable to answer the queries, we may print the help request as an open request in the newsletter to ask if any of the readers can come up with a solution. And, of course, if readers feel that they have a better solution than we came up with, or would like to correct any errors we make, please write to us!

Q. We are forever being told to make backups. Which is the best program to use to backup my QL files?

A. There are quite a few QL backup programs available from my website at http://www.dilwyn.me.uk/backup/index.html as well as some backup programs in the Quanta library. I don't have a particular favourite to recommend, as I have only ever used Winback. All presumably have their merits and some work better than others on some particular types of QL systems. All of them date back to at least 2003 (some even older than that) so it is quite possible that some may fail to work properly on modern systems.

I use my own very simple QL backup routine as I use QPC2. This emulator uses a QXL.WIN filing system, so to make a backup, just copy the QXL.WIN file onto a CD or DVD in Windows. Simple. Done.

If you would like the basis of a backup routine to experiment with to develop your own backup routine, try this one. It will copy all files from a given drive onto another drive, which could be a separate partition on a Qubide drive for example, or if you use a computer with two hard disk drives, it makes it possible to make a complete backup (directory names and all) onto the second hard drive. Equally, it would work with ED disks containing directories and a lot of software – you could make a backup copy on a second disk in drive 2.

```
100 REMark copy entire disk, subdirectories and all
110 CLS : CLS #0
120 INPUT #0,'Copy from which drive > ';dr$
130 INPUT #0,'Copy to which drive > ';tdr$
140 Extended_DIR dr$,''
150 :
160 DEFine PROCedure Extended_DIR (drive$,directory$)
170   LOCal loop,ch,d$,fp,n$
180   ch = FOP_DIR (drive$&directory$) : REMark open channel to
directory
190   IF ch < 0 THEN RETurn : REMark unable to open directory
200   fp = 14 : REMark file position in directory for filename
210   REPeat loop
220     BGET #ch\fp : IF EOF(#ch) THEN CLOSE #ch : EXIT loop
230     GET #ch,d$ : REMark get directory entry name
240     IF LEN(d$) > 0 THEN
250       REMark a filename length of 0 may be a deleted file
260       BGET #ch\fp-9 : REMark file type byte
270       IF CODE(INKEY$(#ch)) = 255 THEN
```

```
280         REMark need to create this directory in destination
290         PRINT'  Creating ';d$;'...' : MAKE_DIR tdr$&d$
300         REMark follow the directory tree
310         Extended_DIR drive$,d$
320       ELSE
330         REMark next line can be used to list all files
340         PRINT'Copying ';d$;'...' : COPY dr$&d$ TO tdr$&d$
350       END IF
360     END IF
370     fp = fp + 64
380   END REPeat loop
390 END DEFine Extended_DIR
```

The program uses Toolkit 2 extensions such as FOP_DIR to open channels to the directory of a disk, and other extensions such as GET to fetch strings from the directory and BGET to position the directory file pointer and handle byte values. On level 2 filing systems it also needs the MAKE_DIR extension o recreate directories on the backup drive.

Importantly, the program shows how to step through a QL filing system's directory one file at a time (each entry is 64 bytes long), read the filenames and check if what it is reading is a directory name.

Line 180 opens a channel to the source drive, e.g. to WIN1_ . The FOP_DIR function returns a value to the variable ch which, if positive, is the channel number opened. A negative value might mean either an error or the end of the directory has been reached, so the procedure ends.

Line 200 sets a start offset of 14 for the variable fp, which starts us going from the 14th byte of a directory, and line 220 tries to position the file pointer there, or returns if we have strayed beyond the end fo the last entry in the directory. The structure of the 64 byte directory is shown below, but for now just accept this is where the filename entry starts, at an offset of 14 bytes from the start of a directory entry. Of course, if we have gone beyond the end of the file we give up and return.

Line 230 then tries to fetch the filename fo this file from the directory entry into the filename string d$. If this returns a null string (a file with no name) it usually means that the particular entry in the directory might be a deleted file, for example. If we get a blank name, the IF clause from line 240 to 360 is skipped and we try to move on to the next directory entry – line 370 adjusts the file pointer variable fp.

If this is a valid file or directory name, line 260 fetches the file type byte to check if it is a filename or directory name. If a directory name, it would have a value of 255 on most systems, although this may vary on Thor computer systems which used a different value for a directory entry.

If it was a value indicating a directory name type (255), line 310 tries to create a directory of the same name on the destination drive, then the routine recursively calls itself with details of the new directory. Don't worry too much if you do not know what the term Recursion means – recursion has been known to give experienced programmers a headache!

If it wasn't a directory name, it must be a filename, so line 340 copies the file from the source directory to the destination (backup) one.

The program then continues to step through the directory one entry at a time until all is done. Obviously, since this is a full backup, there must be at last as much free space on the backup drive as there was on the original.

Here is the list of entries in a standard 64 byte file header. What I have referred to above as opening a channel to the 'directory' using FOP_DIR is really just a list of the 64 byte header which each file has.

QDOS file headers consist of a 64 byte file data block in the directory.

Filenames can be up to 36 characters long (not including the drive name).

Executable files have file type 1 and an associated dataspace value, while
ordinary data files (including BASIC programs) are file type 0. The offset values from the start of each 64 byte header are given in hexadecimal values preceded by a '$' character, with the decimal equivalent in brackets under it for those not familiar with hexadecimal numbers.

```
OFFSET LENGTH    DESCRIPTION
$00    long      file length (in bytes)
 (decimal 0)

$04    byte      file access key (not implemented on original QLs)
 (decimal 4)

$05    byte      file type
 (decimal 5)     0=a data file or SuperBASIC program
                 1=an executable file
                 2=SROFF relocatable object file
                 255=directory
                 3=Thor directory
                 4=font file in 'The Painter'
                 5=pattern file in The Painter
                 6=4 colour mode compressed picture in The Painter
                 11=8 colour mode compressed picture in The Painter

$06    8 bytes   if file type is an executable file (type 1), the
first long
 (decimal 6)     word contains the default dataspace size of the
program in
                 bytes

$0E    word     length of filename
 (decimal 14)

$10    36 bytes characters of the filename
 (decimal 16)

$34    long     file update date
 (decimal 52)

$38    long     file reference date
 (decimal 56)

$3C    long     file backup date
```

```
(decimal 60)
```

Hopefully, using the above table you will be able to relate what the program is doing to what is where in each file header.

This is just a very basic and short backup program which blindly copies everything it comes across – hopefully it will give you the basis of an idea how to writer a backup program to cater for your needs. But wouldn't it be better (and faster!) if it only had to copy files which had changed since the date of the last time you made a backup?

Well, with a little modification, it should be possible to get it to do this!

Looking at the file header table above, we can see that towards the end of the 64 byte header there is a set of 'date' values. Most QL systems imprint the date the file was last changed in the long word 52 bytes on from the start of the header – it is referred to as the 'file update date'. This value is a 32 bit number which is the same value as the clock setting on a QL when the file was updated, in other words, like the value you get from the DATE function, which in turn is the number of seconds since the start of 1961, which is how the QL clock works, the date being the number of seconds which have elapsed since the start of 1961. From this, in turn, you can work out how long the QL clock can keep working, since it is limited by the maximum number of seconds a 32 bit integer can hold starting from the beginning of 1961, although you have to remember that it depends on whether the system concerned uses signed integers or unsigned ones.

Anyway, back to reality and we can see that by reading the file update date of the original file (the source file) and then reading the same information from its existing backup if there is one, we need only copy the file if it has been changed since the original file's update date. So we could surround line 340 with a test of both update dates and if the source file's value is higher than the backup we could use an IF statement along the lines of IF source file update date is greater than backup file update date THEN COPY the file.

Backup programs which take a full backup on the first run, and then backups of files altered or created since then are called Incremental Backups. Winback is an example of this – it lets you make incremental backups onto many floppy disks, even splitting large files in sections spanning more than one disk, although to be fair it is a LONG time since I last used that facility.

In practice, the act of making a backup of a file may update the value of the update date, making this not as easy as it appears. One answer is to make use of the backup date entry in a file, setting this in the source file when a backup copy is made so you only have to check a file's update date and its backup date to check if you need to recopy it. That, my friends, is a little job for you to tinker with to keep you busy until the next issue!

The important thing is that backups are made regularly. For a home user, one backup a week might suffice (if you are not a heavy home user who does not accumulate too much important data during a week), whereas in business making backups is often if not usually a daily essential. After all, if your computer fails or gets stolen or other such mishap and all your records are on computer, you have basically lost all company information since the last backup. Businesses will also store a copy "offsite" e.g. your boss takes it home with him/her to make sure there's a second backup copy in

another location, so that should something happen such as a fire at your office which leaves you without your precious data backups, you can always get your boss to bring in those backups he/she kept safely at home (didn't they….?) to help get you up and running in a temporary office.