**Directories - David Denham**

**What is a directory?**

Think of your computer as a filing cabinet with several drawers. Each drawer is one of your drives. Within each drawer you can have several folders each holding groups of files. The term "directory" is synonymous with the term "folder" as far as we are concerned.

Within each drive you can have several folders each holding groups of files. For example, one could contain all your letters, another could hold all your spreadsheets, while another could hold your SuperBASIC programs.

Each folder or directory can contain further folders. For example, the one containing your SuperBASIC programs can be further split into Games, Utilities, Music programs and so on. Directories within directories like this are referred to as Sub-Directories.

On an original QL, all files were just lumped together without any real form of grouping. This wasn't too bad when all we had was microdrive cartridges holding about 100 kilobytes each. Then we got floppy disks which could potentially hold dozens if not hundreds of small files. As we filled the disks up, a DIR command could give a very long list of files making it harder and harder to find the file we want. Imagine a disk containing over a hundred files - such long lists of files quickly become unmanageable!

When hard disk systems came along for QLs things got even worse. Now you could probably save hundreds if not thousands of files - something had to change.

And change it did. Add-ons like the Miracle Systems hard disc drive introduced us to Level 2 Filing Systems for the QL. The term Level 2 simply means the next level of QL filing systems - directories. The disc could be split into separate folders allowing you to group files together to make file management a lot easier. Such directories were also called "hard directories". The name came about since the original "soft directories" were just files with a prefix added to the name!

**How Do I Know Which Systems Handle Directories?**

First you have to know that your computer or emulator is able to handle these Level 2 directories. If it has a MAKE_DIR command built in, chances are you can use directories. If the manual doesn't tell you, check this with the EXTRAS command - the list should contain the MAKE_DIR command, or in some cases an equivalent function called FMAKE_DIR.

The situation is perhaps not as simple as we might like though.

An original Trump Card doesn't support level 2 directories or "hard" directories, while later versions supplied by Qubbesoft had an updated ROM allowing use of directories! The table below lists some types of QL systems and whether or not they can support directories. The list is far from complete - if it's not in the list I just don't know!

| System Type | Level 2 Directories? |
|---|---|
| Unexpanded QL | No |
| Microdrives | No |
| QL and Trump Card | Later versions only |
| QL and Gold Card/Super Gold Card | Yes |
| Other floppy disk systems | Mostly no, although updated Level 2 ROM could be fitted to some disk interfaces |
| Aurora | Depends on disk card interface fitted |
| Thor | Some versions - yes, although non-standard |
| Q40/Q60 | Yes |
| Qubide | Yes |
| Miracle Systems Hard Disk | Yes |
| QLay and QL2K | No |
| Qemulator | Registered version only |
| QPC | Yes |
| Smsqmulater | Yes |

Here is a simple little test routine which will just test for the presence of the MAKE_DIR or MAKE_DIR extensions on your system. It needs a ramdisc for the temporary file.

```
1000    DEFine PROCedure Level2_Test
1010    LOCal testing,extension$
1020    OPEN_NEW #3,ram1_tmp
1030    EXTRAS #3
1040    CLOSE #3
1050    OPEN_IN #3,ram1_tmp
1060    REPeat testing
1070    IF EOF(#3)=1 THEN EXIT testing
1080    INPUT #3,extension$
1090    IF extension$ == 'MAKE_DIR' THEN
1100    PRINT'MAKE_DIR command present'
1110    END IF
1120    IF extension$ == 'FMAKE_DIR' THEN
1130    PRINT'FMAKE_DIR function present'
1140    END IF
1150    END REPeat testing
1160    CLOSE #3
1170    DELETE ram1_tmp
1180    END DEFine Level2_Test
```

Just enter the command LEVEL2_TEST and it will look through the list of extensions for MAKE_DIR or FMAKE_DIR, printing the name of whichever it encounters in the list.

If you prefer a function to test if there's MAKE_DIR or FMAKE_DIR on the system, try this:

PRINT IsItLevel2 will return 1 if there is a MAKE_DIR or FMAKE_DIR extension.

```
2000    DEFine FuNction IsItLevel2
2010    LOCal testing,extension$
2020    OPEN_NEW #3,ram1_tmp
2030    EXTRAS #3
2040    CLOSE #3
2050    OPEN_IN #3,ram1_tmp
2060    lev2=0
2070    REPeat testing
2080    IF EOF(#3)=1 THEN EXIT testing
2090    INPUT #3,extension$
2100    IF extension$ == 'MAKE_DIR' OR extension$ == 'FMAKE_DIR' THEN
2110    lev2 = 1 : EXIT testing
2120    END IF
2130    END REPeat testing
2140    CLOSE #3
2150    DELETE ram1_tmp
2160    RETurn lev2
2170    END DEFine IsItLevel2
```

**Creating Directories**

Directories are created with the MAKE_DIR command like this:

```
MAKE_DIR drive_directory_
```

So to make a directory on FLP1_ to hold BASIC programs we use:

```
MAKE_DIR FLP1_BASIC_
```

If the system has the FMAKE_DIR function, it makes it easier to cope with error trapping:

```
LET errorcode=FMAKE_DIR(FLP1_BASIC_)
```

FMAKE_DIR will return a negative error code if something went wrong. This error code may be one of the following:

-7 = Not Found (drive not available)
-8 = Already Exists (directory of that name exists already)
-9 = In Use (directory of that name exists already)
-15 = Bad Parameter (cannot handle directories on this drive)

**Some Rules**

A few rules to note:

1. The directory name should end with an underscore character, although many systems will add this automatically if it is missing.

2. The name of the drive and directory does not have to be in quotes unless it contains a non-ascii character such as a full stop or space. The name can always be in quotes if you prefer, or may also be a string variable, e.g. LET d$="Flp1_Games_":MAKE_DIR d$

3. Most systems will let you include an underscore within the directory name, but this is poor practice. An underscore may mean the end of a directory name, so on some systems it may cause a mild confusion as to where the directory name ends. I tend to steer clear of underscore characters within a directory name. It makes life easier if you just use letters and numbers.

Most systems will allow spaces if you want to make long name more readable, but of course the name has to be in quotes. I find it best to avoid spaces and other non-alphanumeric characters.

4. Name lengths. The sum length of both the directory name and a file name may not exceed 36 characters. So it is best to keep directory names short and meaningful, especially when using sub-directories within other directories, remembering that the underscore characters also count towards the maximum length of 36.

5. Like QL filenames, directory names are usually case insensitive - it doesn't matter if you use upper case, lower case or a mix. The only exception is some emulators which are subject to the case sensitivity of the file system on which they run. Linux, for example.

**Saving Files To Directories**

To save a file to a directory, we just include the directory name before the filename. We can save a BASIC program called mygame_bas into a directory called BASIC on drive WIN1_

```
SAVE win1_Basic_MyGame_bas or, of course SAVE "win1_Basic_MyGame_bas"
```

Copying a file from floppy disc to hard disc is just as simple:

```
COPY Flp1_MyGame_bas TO win1_Basic_MyGame_bas
```

If your system has the WCOPY command, you can use this to bulk copy files, e.g. to copy all files from Flp1_ to Win1_Basic_ just use:

```
WCOPY Flp1_ TO Win1_Basic_
```

Equally, if you wanted to copy files back from a hard disc directory back to a floppy disc:

```
WCOPY Win1_Basic_ TO FLP1_
```

What the command does is try to work out the differences between where the files are stored and copied to and change the full filename accordingly.

WCOPY can also use a simple form of wild card name, so that only files matching that wild card are copied. Supposing we have a disk full of all sorts of files and we wished to copy just the Quill documents to a directory called Win1_Quill_ and just the Abacus spreadsheets to Win1_Abacus_

```
WCOPY Flp1__doc TO Win1_Quill__doc
WCOPY Flp1__aba TO Win1_Abacus__doc
```

The wildcard is the extension - anything ending with _doc or _aba in these examples.

DIR List

When you DIR a disc containing directories, the name of a directory is shown on most systems by having the "->" characters after a name.

```
DIR WIN1_
might give

WIN1 QDOS
35840/40960 sectors
Boot
Basic ->
Games ->
Xchange ->
```

In this example, I have a boot program written in Superbasic plus three directories called Basic, Games and Xchange. Note how the directories have -> after the name, and how the last '_' is not normally shown.

It is quite normal not to put important system files such as a boot program not into any directory. This is called putting a file in the 'root directory' or 'base directory'.

You may also come across some programs which may not work correctly from a directory, so these can be put in a root directory if required as long as you remember to keep the number of files down to reduce clutter.

**Sub-Directories**

If we want to create sub-directories within an existing directory, that's quite easy. Just work out what you want and make sure the main directory exists first. In an example, I'm going to create a directory to hold a copy of Xchange, and within that, four sub-directories to hold Quill, Abacus, Easel and Archive files.

First, I create the main directory called Xchange:

```
MAKE_DIR Win1_Xchange_
```

Next, I create the four sub-directories to go inside the Xchange directory, simply by adding the names of the sub-directories to the name of the main one. I'll call these sub-directories Quill_, Abacus_, Archive_ and Easel_

```
MAKE_DIR Win1_Xchange_Abacus_
MAKE_DIR Win1_Xchange_Archive_
MAKE_DIR Win1_Xchange_Easel_
MAKE_DIR Win1_Xchange_Quill_
```

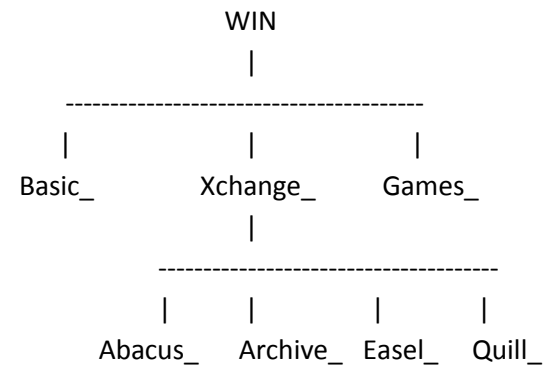
I happened to put those in alphabetical order, but there is no real need to do this unless you want them to appear in alphabetical order in a DIR list.

If I now do a DIR of Win1_ the list is still shown the same as the first example above. But if I now do a DIR of Win1_Xchange_ I see the new sub-directories:

```
WIN1 QDOS
35840/40960 sectors
Xchange_Abacus ->
Xchange_Archive ->
Xchange_Easel ->
Xchange_Quill ->
```

So you can see that if you use DIR to list the content of a directory, it shows what's in that directory and not anything in the main directory before it. This brings us to their hierarchy - the first directory is referred to as the "top level" directory.

When you have directories within directories, you can think of the whole structure as a "tree", with various branches leading away from the root or top level. You can draw a diagram a bit like a family tree (an upside down tree) like this. This will help us envisage what a directory structure is like when it comes to "navigation".




Sub-directories within other directories are sometimes referred to as "nested directories".

**Deleting Directories**

From time to time we may need to delete a directory, either because we have finished using it, or we have realised we made a typing mistake and misnamed it. Suppose I'd intended to create a directory called Win1_Toys_ and accidentally typed MAKE_DIR Win1_Tyos_. The directory can be deleted with a simple DELETE command as long as you take care to type the name correctly to avoid accidentally deleting a similarly named directory!

```
DELETE Win1_Tyos_
```

However, you cannot normally delete a directory name if it already contains some files. You would need to delete those files first. So if I had a directory named Win1_Texts_ which contained three plain text files, I'd need to delete all three of those files before I could delete the directory called Win1_Texts_. A quick but potentially dangerous way of deleting them would be to use the WDEL (Wildcard Delete) command to delete all its files first:

```
WDEL Win1_Texts_
then DELETE Win1_Texts_
```

If you have sub-directories within a directory, you will need to delete the content of the deepest directories first, then delete the sub-directory names, then delete all the files in the parent or higher directory and work your way backward until you reach the one you want to remove. Taking Win1_Xchange_ in the example tree above, in order to remove Win1_Xchange_ you'd need to delete the content of all four sub-directories first, then delete all four sub-directories, then delete any remaining files in Win1_Xchange_ before finally removing Win1_Xchange_. It is a bit tedious with a lot to type in, but perhaps that's no bad thing as it forces you to do it step by step to avoid accidentally deleting everything!

The sequence of BASIC commands you would need to delete Win1_Xchange_ above would be:

```
1. Remove sub-directory content:
WDEL Win1_Xchange_Abacus_
WDEL Win1_Xchange_Archive_
```

WDEL Win1_Xchange_Easel_
WDEL Win1_Xchange_Quill_

2. Delete sub-directories:
DELETE Win1_Xchange_Abacus_
DELETE Win1_Xchange_Archive_
DELETE Win1_Xchange_Easel_
DELETE Win1_Xchange_Quill_

3. Delete any other files left in Win1_Xchange_:
WDEL Win1_Xchange_

4. Finally delete Win1_Xchange_ itself:
DELETE Win1_Xchange_

**Default Directories**

It can be a bit tedious typing in very long drive and directory names, so Toolkit 2 added commands to set default directories. There are three of them in all - one called PROG_USE to set default directories for programs you can execute (e.g. Xchange or QL Quill), DATA_USE for ordinary data files and BASIC programs and DEST_USE for the default destination when copying files.

1. PROG_USE drive_directory_

Suppose you keep your executable programs in a directory called PROGS_ on Win1_. You can set PROG_USE Win1_Progs_ to make EXEC and EX commands start the programs from there by default without having to add a drive and directory. Let us compare two EX commands:

EX Win1_Games_MyGame_bin - this starts the game called MyGame_bin from Win1_Games_, in other words you specified a drive and directory called Win1_Games_

EX Utility_Task - here you have specified no drive and directory, so it looks on the one specified in the last PROG_USE statement automatically (called the Program Default Directory), in this case Win1_Progs_. Saves a bit of typing when you have grouped all of your programs in the same place.

2. DATA_USE drive_directory_

Suppose you keep your BASIC programs in Win1_Basic_. After DATA_USE Win1_Basic_ the computer will look for a Basic program or indeed any data file in this directory unless you explicitly state where:

LOAD Win1_Xchange_MyBasicProg_bas - this one specifies a drive and directory, so it will look for the file there.

LOAD MyBasicProg_bas - this doesn't specify a drive or directory, so it checks the DATA_USE default value. In other words, it realises you didn't specify anything, looks up the default (called the Data Default Directory) which was Win1_Basic_ so ends up loading the file from there.

Given that data files are more likely to be spread across many directories, this is less useful in this particular context, but another use is for the DIR command. If you had set DATA_USE WIN1_ then enter the command DIR with no drive name, it looks up the Data Default Directory value (Win1_) and tries to do a DIR of that, in this case like a DIR WIN1_.

3. DEST_USE drive_directory_

Sets the default destination directory when files are being copied, moved or renamed, usually when these commands which would otherwise be used with two names this provides the second or destination name. If the destination name does not end with a '_' it usually means it's a non-directory device, such as a serial port or parallel printer port.

This can be useful. Set DEST_USE "PAR" on a system with a parallel printer port and this can be useful for copying files to a printer, using either COPY or SPL (spooler) commands:

DEST_USE PAR
COPY win1_plain_txt (sends it to PAR printer port)
SPL win1_PLAIN_txt (uses the SPL spooler job to print a file to a PAR printer port in the background).

I tend to find it best to group all my executable programs into one directory and point PROG_USE to that one. I also set DATA_USE to point to my main drive, WIN1_. DEST_USE is just used for the default printer port. My boot program contains lines like this:

```
200 PROG_USE Win1_Progs_
210 DATA_USE Win1_
220 DEST_USE Par
```

Many systems impose a limit of 32 characters on a default name. This is in keeping with the maximum filename length of 36 characters. After all, if you have a long 32 character default name, it only leaves 4 characters for the rest of the filename, so you can understand why!

**Checking The Settings**

Sometimes you may forget what defaults you have set. This is where the three functions DATAD$, PROGD$ and DESTD$ come in. These request and return the relevant information from the system:

PRINT PROGD$ returns the default set with PROG_USE
PRINT DATAD$ returns the default set with DATA_USE
PRINT DESTD$ returns the default set with DEST_USE

An additional command DLIST prints a list of all three settings.
Using these extensions you'll quickly realise that the system can only have one default for the entire computer. In other words, when you set PROG_USE "Win1_" it applies to all programs. It is not possible for each program to have a separate setting. Also, not all programs understand these defaults - life is not always easy! In a future article I'll show how to work around these limitations for awkward programs.

**Default Defaults**

When you first start a QL or emulator, these defaults are usually set to one of the drives in the system and no directory. If your system has only floppy disc drives and no hard disc drive, it will usually set PROG_USE to FLP1_ and DATA_USE to FLP1_ (on some systems with two disc drives DATA_USE may be set to FLP2_), and DESTD$ is usually a serial port or printer device such as PAR or SER1

**Navigation**

Some extensions are usually provided to navigate the defaults up and down a directory tree. Personally, I rarely (if ever) find the need to use these, but they are there. Normally, these three commands work on the DATA_USE default name, but if the PROG_USE name happens to be the same they will affect that too.

**DDOWN name** - moves down the tree, adds the name given to the default, so if the default happened to be Win1_ and the command was DDOWN Xchange the default would now become Win1_Xchange_. A subsequent DDOWN Quill would change the default to Win1_Xchange_Quill_

**DUP** - moves up the tree by removing the last directory name, so if the default was set to Win1_Xchange_Abacus_, a DUP would make it Win1_Xchange_ and a second DUP would make it just Win1_

**DNEXT** - moves up and then down a different part of the directory tree.

**Separating Directory And Filename**

In some ways, the QL filing system is a bit restrictive. Apart from the 36 character total name length limit, it's also hard for a program to work out which part of a name is a directory and which is the filename part.

Most operating systems use specific symbols in a filename to separate the directory and filename, for example on Windows we might save something to C:\QLfiles\letter.txt. Here we know that the drive name is C:, the directory name is indicated by the backslash characters (directory called QLfiles) and the filename is the bit after that last backslash, here "letter.txt". The QL is a bit more simplistic in this respect and the directory separator character can also be part of the filename,

potentially making it hard to work out which is the directory name part, and which is the filename part.

There is a solution, but it's not obvious and requires a little bit of devious code to ask the system which is the directory, then subtract from the full filename (or full 'path' name as it's sometimes referred to).

The answer lies in the use of the FNAME$ function, which returns the name of a file normally, or of the directory part if you open a channel using the FOP_DIR command:

```
LET f$="Win1_Xchange_test_doc"
LET channel=FOP_DIR(f$)
PRINT FNAME$(#channel)
CLOSE #channel
```

This prints Xchange, the name of the directory containing the file called TEST_doc. Note that it doesn't return Xchange_ only the part before the underscore, so you need to take that into account. Also, not including the drive name.

Separating the directory path and the filename needs a bit of juggling like this (I wrote an article about Directory Names back in Volume 10 Issue 4 of QL Today).

First, we use FNAME$ to return the full path and filename without the drive name:

```
LET f$="Win1_Xchange_test_doc"
LET ch=FOP_IN(f$)
LET fullname$=FNAME$(#ch)
CLOSE #ch
LET ch=FOP_DIR(f$)
LET dirname$=FNAME$(#ch)
IF dirname$<>"" THEN LET dirname$=dirname$&'_'
CLOSE #ch
LET filename$=fullname$(LEN(dirname$)+1 TO LEN(fullname$)
PRINT filename$
```

which (eventually!) prints "test_doc"

**Conclusion**

Directories are a very useful and sometimes underused facility on modern QL systems. They take a bit of getting used to but once you've got used to them you'll wonder how you ever managed without them. The main advice is to remember that the length of a directory name and filename combined must not exceed 36 characters and try to plan ahead what names you intend to use because if you change your mind later it can be a very fiddly job to make big changes.