Quanta Helpline **Dilwyn Jones**

Job2Bas Program

With the increasing use of QL emulators prompting a need to transfer ever more files between a QL and an emulator, one solution has been to make use of the ZIP compression utility to protect the QL executable job header as a QL program gets transferred through a non-QL environment.

If you copy a QL program such as Quill into a Windows folder, for example, Windows does not know how to handle the QL executable's header, strips it off, and to all intents and purposes converts what was a program you could EXEC on the QL into a simple data file. Try to execute that in QDOS or SMSQ/E and you might well get an error message such as "bad parameter".

Using QL ZIP to hide away the header does work, but of course you need to know how to use QL ZIP and UNZIP as well as know how to transfer the file through another operating system so that the QL emulator receives the file intact. Plus there is another little snag - if the QL emulator has no Unzip program, you will need a copy of Unzip to Unzip the Zipped copy of Unzip - catch 22.

So I thought I'd attack this problem from another angle and try to see if I could think of a way of allowing QL executable programs to pass through another operating system without losing the file header. This method had to ensure that apart from Toolkit 2 (which most emulators

either have built in or can load a simple Toolkit 2 ROM image) the process needs no other significant extra software.

So, the rather radical solution I cam up with was to convert the QL program to a SuperBASIC program. Yes, that's right, a BASIC program.

SuperBASIC and SBASIC programs can pass through other operating systems as simple data files as long as you don't change the end of line character by doing something like loading the QL BASIC program into a wordprocessor or something like that, which might add carriage return characters to the end of the lines of BASIC and so prevent it from running.

What JOB2BAS does is to store the executable program as a (long) set of DATA statements in a BASIC program, and adds a few lines of BASIC to those DATA statements to allow the original program to be reconstituted just by running the resultant BASIC program on the target computer.

So the process is:

1. Run the JOB2BAS_bas program
2. Tell it which executable program to encode as a BASIC program
3. The output BASIC program is written to a file
4. Transfer this new BASIC program (we might call it OUTPUT_BAS for example) to the target operating system

5. Copy the BASIC program into the target QL emulator
6. Run it on there and tell it what filename to use to save the reconstituted executable program. It remembers the original filename, but you can choose a new filename if you wish.
7. Having saved the executable program file, you can then move any other files needed over as well and proceed to test the software on the emulator.

It is a difficult concept to visualise, but having used the process once or twice (e.g. to transfer a copy of QL UNZIP to the emulator!), you should find it's an easy enough program to use.

There are three options you can change at the start of the program:

1. First line number of the created BASIC program
2. Line increment step from one line to the next
3. The number of items on each line of DATA values (higher values allow longer program to be encoded, but make the output program more difficult to read if you wish to study it)

Here is the Job2Bas program listing. If you would rather not type it in, it is available to download from my website at http://www.dilwyn.me.uk/arch/index.html , where you will also find a copy of the Unzip program already processed by this program.

```
100 REMark JOB2BAS_bas - convert an executable job to a
BASIC program
```

```
110 REMark by Dilwyn Jones, September 2011
120 :
130 REMark configuration options
140 line_no%        = 1 : REMark first line number of the
outputted S*BASIC program.
150 line_inc%       = 1 : REMark line number increment
steps of S*BASIC program
160 datas_per_line% = 4 : REMark number of DATA values
per line
170 :
180 CLS : CLS #0
190 :
200 INPUT #0,'Enter name of program to convert to BASIC
data > ';ip$
210 IF ip$ = '' THEN STOP
220 :
230 INPUT #0,'Filename of BASIC program to save > ';op$
240 IF op$ = '' THEN STOP
250 :
260 OPEN_IN #3,ip$
270 IF FTYP (#3) <= 0 THEN
280   REMark no need to convert data files or S*BASIC
programs
290   CLOSE #3
300   PRINT #0,'No need to convert this file type.'
310   STOP
320 END IF
330 :
340 file_len   = FLEN(#3) : REMark length of original
program
350 data_space = FDAT(#3) : REMark dataspace of original
program
360 :
370 IF file_len <= 0 OR data_space <= 0 THEN
380   CLOSE #3
390   PRINT #0,'Unsuitable program file.'
400   STOP
410 END IF
420 :
430 REMark how many long words and any extra (1-3 bytes)
in Job file?
440 no_of_words     = INT(file_len/2)
450 oddbytes        = file_len-(2*no_of_words)
460 :
```

```
470 base = ALCHP(file_len) : REMark use RESPR instead if
your system has no ALCHP extension
480 IF base <= 0 THEN
490   CLOSE #3
500   PRINT #0,'Unable to allocate memory to hold the
original job.'
510   STOP
520 END IF
530 :
540 PRINT #0,'Loading'!ip$!'...'
550 LBYTES ip$,base
560 PRINT #0,'Building output S*BASIC program...'
570 :
580 OPEN_NEW #4,op$
590 :
600 REMark comment start of the S*BASIC equivalent...
610 PRINT #4,line_no%&' REMark '&ip$&' as an S*BASIC
program.'
620 line_no% = line_no% + line_inc%
630 :
640 REMark comment how to recreate the Job program file
650 PRINT #4,line_no%&' REMark just RUN this program to
recreate the original Job file'
660 line_no% = line_no% + line_inc%
670 :
680 REMark add code to output BASIC program to recreate
original Job
690 PRINT #4,line_no%&' :'
: line_no% = line_no%+line_inc%
700 PRINT #4,line_no%&' CLS : CLS #0 : RESTORE'
: line_no% = line_no%+line_inc%
710 PRINT #4,line_no%&' READ words,oddbytes'
: line_no% = line_no%+line_inc%
720 PRINT #4,line_no%&' READ orig_name$,orig_dspace'
: line_no% = line_no%+line_inc%
730 PRINT #4,line_no%&' base = ALCHP((2*words)+oddbytes)
: REM or use RESPR()'         : line_no% =
line_no%+line_inc%
740 PRINT #4,line_no%&' FOR a = 0 TO words-1'
: line_no% = line_no%+line_inc%
750 PRINT #4,line_no%&'   READ value : POKE_W
base+(2*a),value'                        : line_no% =
line_no%+line_inc%
760 PRINT #4,line_no%&' END FOR a'
```

```
: line_no% = line_no%+line_inc%
770 PRINT #4,line_no%&' IF oddbytes > 0 THEN'
: line_no% = line_no%+line_inc%
780 PRINT #4,line_no%&'   READ value : POKE
base+(2*words),value'                       : line_no% =
line_no%+line_inc%
790 PRINT #4,line_no%&' END IF'
: line_no% = line_no%+line_inc%
800 PRINT #4,line_no%&' PRINT #0,"Original filename was
";orig_name$'                  : line_no% =
line_no%+line_inc%
810 PRINT #4,line_no%&' INPUT #0,"Save as filename >
";op$'                            : line_no% =
line_no%+line_inc%
820 PRINT #4,line_no%&' IF op$ = "" THEN STOP'
: line_no% = line_no%+line_inc%
830 PRINT #4,line_no%&' PRINT #0,"Saving ";op$'
: line_no% = line_no%+line_inc%
840 PRINT #4,line_no%&' SEXEC
op$,base,2*words+oddbytes,orig_dspace'
: line_no% = line_no%+line_inc%
850 PRINT #4,line_no%&' RECHP base : REMark remove if
using RESPR() above'             : line_no% =
line_no%+line_inc%
860 PRINT #4,line_no%&' PRINT #0,"Program finished"'
: line_no% = line_no%+line_inc%
870 PRINT #4,line_no%&' STOP'
: line_no% = line_no%+line_inc%
880 PRINT #4,line_no%&' :'
: line_no% = line_no%+line_inc%
890 :
900 REMark how many long words and any extra bytes...
910 PRINT #4,line_no%&' DATA '&no_of_words&','&oddbytes&'
: REMark number of LONG WORDS and ODD BYTES at end'
920 line_no% = line_no% + line_inc%
930 :
940 REMark what was the original filename?
950 PRINT #4,line_no%&" DATA '"&ip$&"' : REMark original
Job program's filename."
960 line_no% = line_no% + line_inc%
970 :
980 REMark what was the original dataspace?
990 PRINT #4,line_no%&' DATA '&data_space&' : REMark
original dataspace'
```

```
1000 line_no% = line_no% + line_inc%
1010 PRINT #4,line_no%&' :' : REMark just a spacer line
1020 line_no% = line_no% + line_inc%
1030 :
1040 REMark start to assemble the program data
1050 dpl% = 0 : REMark how many DATA items on current
line so far?
1060 lne$ = line_no%&' DATA '
1070 :
1080 FOR a = base TO base+file_len-1 STEP 2
1090   word = PEEK_W(a) : REMark get a word
1100   IF dpl% >= datas_per_line% THEN
1110     PRINT #4,lne$ : REMark output the line
1120     line_no% = line_no% + line_inc%
1130     lne$ = line_no% & ' DATA '&word
1140     dpl% = 1
1150   ELSE
1160     REMark still room on this line
1170     REMark add a comma before value (unless this is
the first item after DATA)
1180     IF dpl% > 0 THEN lne$ = lne$&','
1190     lne$ = lne$ & word : REMark add to DATA list
1200     dpl% = dpl% + 1
1210   END IF
1220 NEXT a
1230   IF dpl% > 0 THEN PRINT #4,lne$ : REMark part line
to output
1240 END FOR a
1250 :
1260 IF oddbytes THEN
1270   REMark any odd bytes (1 to 3) to add?
1280   line_no% = line_no%+line_inc%
1290   lne$ = line_no%&' DATA '
1300   FOR a = 1 TO oddbytes
1310     IF a > 1 THEN lne$ = lne$&','
1320     lne$=lne$&PEEK(base+file_len-oddbytes)
1330   END FOR a
1340   PRINT #4,lne$
1350 END IF
1360 :
1370 REMark finished, so tidy up
1380 CLOSE #3 : REMark input JOB file
1390 CLOSE #4 : REMark output BASIC file
1400 RECHP base : REMark REM out this line if no RECHP
```

```
command on your system
1410 :
1420 REMark tell user we have finished
1430 PRINT #0,'Program finished.'
1440 PRINT
'Transfer'!op$!'to'!'the'!'required'!'system,'!'then'!'ju
st'!'RUN'!'it'!'to'!'recreate'!'the'!'Job'!'program'!'fil
e!'
1450 STOP
```