

PRESENTED
FREE WITH



OWNER'S MANUAL

No. 3

QL BARGAINS

★ ALL PRICES INCLUDE VAT AND NEXT DAY DELIVERY BY SECURICOR ★

★ OR FIRST CLASS POST ★

CALL 0267 231246 FOR PERSONAL ASSISTANCE

FULLY TESTED QL VERSION 3 SOFTWARE £199

PRINTER PRICES INCLUDE MIRACLE SYSTEM INTERFACE

MANNESMANN TALLY MT80+	£219
SEIKOSHA SP1000A (Friction Tractor and NLQ)	£239
EPSON LX80	£269
EPSON LX80 WITH TRACTOR FEED	£286
TAXAN KP810	£299
TAXAN KP910	£419
QUEN DATA DAISYWHEEL	£289
Miracle Systems Parallel Interface	£29
Serial Cable	£10
2000 Sheets Fanfold Paper (Supplied with printer)	£16

★ ★ ★ EIDERSOFT ICE ROM — JUST £46 ★ ★ ★

ACCESSORIES

4 Microdrive Cartridges	£8.00
10 Cartridges with Transform Box	£23.00
10 DS/DD 5.25" Disks	£23.00
10 DS/DD 3.25" Disks	£39.00
RIBBONS	
Brother M1009	£5.50
MT80 or Shinwa	£6.00
Quen Data	£6.00

★ ★ BUDGET RAM & DISK UPGRADES ★ ★

CUMANA DISK INTERFACE ONLY £79 512K EXPANDER RAM ONLY £119

CUMANA DUAL 3½" DRIVES + CUMANA INTERFACE ONLY £269

FULL RAM + DISK DRIVE UPGRADE ONLY £379

★ BRIGHT STAR MODEMS ONLY £199 ★

INCLUDES AUTO ANSWER/DIAL

PCML DISK INTERFACE WITH 260K RAM & ROM £249

PCML DISK INTERFACE WITH CUMANA DUAL DRIVE £439

LANGUAGES

METACOMCO ASSEMBLER	£29
METACOMCO BCPL	£49
METACOMCO LISP	£49
METACOMCO PASCAL	£72
METACOMCO 'C'	£82
DIGITAL BASIC COMPILER	£54
DIGITAL FORTH + REVERSI	£26
DIGITAL SUPER MONITOR	£13
TALENT CARTRIDGE DOCTOR	£13

BUSINESS

TR SYSTEMS PAYROLL	£63
DECISION MAKER	£35
PROJECT PLANNER	£35
ENTREPRENEUR	£35
QL HOME FINANCE (BUZZ)	£22
TYPING TUTOR	£20
EIDERSOFT ARCHIVER	£17
EIDERSOFT QSPELL (CART)	£20
EIDERSOFT QSPELL (DISK)	£23

LEISURE

TALENT ZKUL	£13
TALENT WEST	£13
TALENT GRAPHQL	£29
EIDERSOFT ZAPPER	£9
QL SPRITE GENERATOR	£20
QL SUPERBACKGAMMON	£12
PSION CHESS	£17
HYPODRIVE	£13

MICRODEAL HOPPER	£12
MICRODEAL CUTHBERT/SPACE	£12
MICRODEAL CRAZY PAINTER	£11
MICRODEAL FLIGHT SIMULATOR	£16
MICRODEAL LANDS OF HAVOC	£16
QL CAVERN	£12
MATCH POINT	£13

MONITORS (CABLES INCLUDED)

Microvitec Cub 1451/DQ3 Colour	£254
Microvitec Cub 1451/DQ3 with Swivel Stand	£274
Swivel Stand for Microvitec	£25
Philips V7001 18MHz Green (RECOMMENDED FOR THE QL)	£92
Swivel Stand for Philips Monitor	£23

3250 WATT MAINS FILTER

- ★ Provides four protected mains sockets with plug
- ★ Avoid crashes and damage to the QL's components
- ★ 30db suppression 1MHz to 30MHz
- ★ 130 joules spike suppression

MODAPTOR

- ★ Link your QL to any modem with an RS232C socket
- ★ Prestel and Bulletin Board software included

QUICKSHOT II JOYSTICK WITH ADAPTOR

£14

QL JOYSTICK ADAPTOR £5

STRONG COMPUTER SYSTEMS

BRYN COTTAGE, PENIEL, CARMARTHEN, DYFED SA32 7DJ.

TELEPHONE: 0267 231246

QL USER

ARCHIVE

The QL's Index Link

EDITOR

Paul Coster BSc

ASSISTANT EDITOR

Paolo Baccanello

ART EDITOR

Mike Spiller

PRINCIPAL AUTHORS

Dr A Carmichael

Adam Denning

Roger Vernon

Christopher Sherwood

ADVERTISING

Phil Baker

PUBLISHER

Neil Wood

QL User,
Priory Court,
30-32 Farrington Lane,
LONDON EC1R 3AU.

Telephone 01-251 6222

Published and Distributed by EMAP
Business & Computer Publications.
Subscriptions, backnumbers and
reprint information from: Carl
Dunne (Magazine Services) on
01-251 6222 Ext 2429.

U.S. Sales Agent - Motorsport, RR1,
Box 200D, Jonesburg, MO 63351.

Typesetting and Make-up by
Time Graphics (Northampton) Ltd

Printing by Riverside Press.

© COPYRIGHT QL USER - 1985

Archive is probably the most difficult of the four Psion packages to really come to grips with, or at least it appears so, especially if you try to understand everything in the User Guide all at once. On the other hand it is a powerful package and probably more than any of the others, realises the potential of using a micro to replace a manual procedure. The purpose of this article is to give you some additional help and ideas, firstly on how to get started, and then on how to translate your particular application into an Archive database - whether its for high-powered scientific research or just keeping track of your record and tape collection. If you've already made a start with Archive the next section will just be revision before getting down to applications.

Archive is a tool for organising information - storing it, retrieving it, sorting it, searching through it and generally managing it. The analogy to a card-index system is a good one since information is stored on *records* (like the cards in a manual system) and the records are collected together in a *file* (like the card-index box). On each record there are a number of *fields* (like the spaces on each card) which are used to contain the information relevant to the particular item (or person) referred to by that record. Fields have a *type* which indicates whether the field is for textual or numeric information. The great advantage of using Archive instead of a manual card-index is that the laborious sorting and searching jobs are carried out automatically. When you add a new record to the file, or delete or amend an existing one, the file is automatically re-sorted. Searching for particular items or groups of items, accumulating totals from individual fields on all or some of the records (if you

have a printer) printing out sorted lists, reports or labels, can all be readily achieved with the Archive database system. Archive also allows you to have several database files open at once, so for more complicated applications it is ideal for linking together related information.

FIRST STEPS

Naturally, before you can use a tool effectively you need to know what it does and the basics of how to use it. This doesn't imply that you need to pore over every detail of the manual - in fact this is probably counter-productive. It's far better to experiment with a few simple examples first.

An obvious contender is the telephone/address list since it is quite straight-forward and probably all of us have one, even if it's only scribbled in the back of our diaries (maybe it is in a card-index box!). Load Archive and try typing the following:

```
Create "Phone" <ENTER>
Surname$      <ENTER>
Firstname$    <ENTER>
Address$      <ENTER>
Telephone$    <ENTER>
endcreate     <ENTER>
```

You have just created your first database file - it really is as simple as that! The effect of the above command is to create a file named "Phone.dbf" (the ".dbf" part of the filename is the default file extension for database files) and each record in the file will have 4 text fields for storing the name, address and 'phone number of the people on the list. To insert some names and addresses in the file just type: **insert** which should cause a blank record to be displayed for you to type in the details. When you have entered a number of records press <F4> to leave insert mode. You could now try a number of the commands like **order** which sorts the file into

ascending or descending alphabetical order on any field (eg. Surname\$), **cls** which clears the screen, **display** which displays the current record, **search**, **find** and **locate** which are different ways of looking for a particular record, or **alter** which is used to change the record on display. There are many more commands on which you will find full details in the User Guide or, if you press <F1>, on Archive's own help pages. It's worth trying as many of the commands out as possible so that you get the feel of what you can do, and if you do this before you type in too much data, you won't feel too bad if you make an error and delete some data accidentally! (Do it after 2 hours typing and it's much more painful - so always remember to back up all your data at least once on another cartridge.) When you have finished using a file, to make sure it is all present and correct on the microdrive cartridge, you must type **close**, or **quit** (which also returns you to SuperBASIC) or **new** (which also clears any procedures or special screen formats from the computer's memory). To examine the file again use the command **look** or, if you want to change the file at all, **open**.

When you are reasonably confident about using commands the next step is to try to incorporate some in a few simple procedures. Procedures are ways of storing often repeated sets of commands, or of carrying out some processing on the data in the database.

For example, if you type all your address list into the machine it's not very convenient to have to nip up to the computer, load Archive and the database and then locate the required name, every time you want to ring someone up. It's much easier if you have a paper list too.

The advantage the database gives you for storing the address list is that it is easy to change and keep in alphabetical order, and it may be useful for things like typing address labels automatically. So to get the best of both worlds we need to write a procedure to print out the address list whenever an up-to-date copy is required. To do this first enter the command **edit**, to use Archive's screen editor, and type in the following procedure. The language used is actually a dialect of BASIC specifically developed for the database package and incorporating all the commands which can be used directly from the keyboard. Here's the procedure:

```
proc AdList
rem *** Prints the address list on the
rem *** screen or to a printer
input "Send output to printer (y/n)? ";Ans$
spooloff
if Ans$ < > "y" and Ans$ < > "Y"
cls :rem *** Output sent to screen
spoolon screen
endif
first :rem *** Start at the beginning of the file
while not eof( )
lprint "Name: +Firstname$+" +Surname$
lprint "Address: +Address$
lprint "Telephone: +Telephone$
lprint
next :rem *** Go to the next record
endwhile
spooloff
endproc
```

The procedure prints out a list of all the addresses in the database in the order in which they are held in the file. If you used the **order** command this will be alphabetical order, otherwise it's likely to be in an order related to how you typed it in. Having typed in the procedure, press <ESC> twice to exit the editor and use **save** to store the procedure on microdrive, as the procedure will be lost from memory when you **quit** or type **new**. To try it out type **Adlist**.

The procedure demonstrates a few features of Archive's language like the conditional statement (**if**) and the loop statement (**while**) which use a strictly *structured* form – no GOTO's allowed! It may take a little getting used to if you've only used BASIC before, but while it's sometimes a little inconvenient it should result in clearer programs

to read. Remember, anything you can write with a GOTO you can write in a structured form using **if / then / else / endif** or **while / endwhile**. If you need to jump forwards in the program an **if** statement could replace the jump, if you need to jump backwards a **while** could be used. Other additional features of the language include the **trace** command which shows you the statements as they are being executed – very useful for debugging.

Having noted some of the differences with BASIC, writing procedures for Archive is nevertheless in most respects very similar to writing in SuperBasic, and it's not the intention of this article to spend

reports and form lists (for teachers), facts for exam revision (for pupils), quiz games (and other trivial pursuits), catalogues of products, photograph collections, analysis of experimental results or even storing information about your wine cellar!

The first part of the job is to really examine what you want to do and how you think Archive is going to help. It's all too easy to jump in with both feet, spend a few hours bashing the keyboard to set up a database, and probably quite a few more hours typing in the data, only to find that it doesn't do the job you want it to do. Even with thought you may not get it right first time, but you can always add enhancements if you've got the basic structure right. A little time spent examining the problem at the beginning really pays dividends.

Archive allows you to use multiple data files simultaneously, which is a feature you may want to use as it is very useful for referring to a different set of information from a data record. For example, a school form list might well contain the addresses of the pupils, but it would be inconvenient to also store the reports in the same file, and time-consuming to type addresses into the report file. Two files with a simple cross-reference between them is therefore required.

If you do reference other files it is important that you pick up the right record. Surnames are rarely unique, and even with initials or first names can often be duplicated, so it is better to have another field for a reference number. Such a field is known as the primary key to the file, and one value of the primary should key give us only one record.

As well as considering what data you want to store and whether it can conveniently be stored as one or more files, you must consider how the database is going to be used – how information gets in and how it gets out. Ideally data going in should be checked to make sure it is consistent with what is already in the database, and data being output may well be required in the form of a report (for which a procedure will probably be needed) or a special

screen format (which can be designed using the **sedit** command). How much time you spend writing input procedures and so on will really depend on how important it is that mistakes are found during input, and possibly on who inputs the data.

PHILASOPHY

Consider another example – collecting stamps. Depending on the size of your collection you might want to store data on each of your stamps, or just those in a particular album, or maybe one data record per page or per album. Depending on how you want to use the data and what sort of reports you want the computer to give you when all the data has been entered, you might want to store the purchase price of stamps, their estimated value, catalogue cost, condition, date of issue, and so. You might even want to store details of stamps that you'd like to own one day! You will need to take into account the total number of records you want to create, as if you have an unexpanded QL and sort your files on one field, the maximum would be about 1000 records. If you collect stamps and wish to use Archive to catalogue your collection you may choose to store slightly different data from that which I that chosen, but in any case here is an attempt at setting up data files for this application.

Looking at the data to be stored you will see that it falls into two categories: data about particular stamps in the collection, and data about particular types of stamps (of which there may be a number of examples in the collection). This suggests that two data files will be needed. The following two procedures define suitable database files:

proc NewTypes;Filename\$

```
proc NewTypes;Filename$
rem *** Creates a new file for storing
rem *** data about types of stamps
create Filename$ logical "T"
Type_Ref_No
Stamp_Names
Country$
Issue_Date$
Mint_Value
Used_Value
Comments$
endcreate
order Type_Ref_No$
endproc
```

too much time on this aspect. As practice in writing procedures try writing one to help inputting the data. For example after asking for each field it could change the surname to upper-case, firstname to initial upper-case followed by lower-case, and check the phone number contains only numbers. Use the **append** command to enter each record into the database.

APPLIED SCIENCE

What about other more complicated applications? The really good thing about Archive is that you can tailor it to your own interest and use it to store and organise data how you want it. There is no limit to what you could apply it to: stock control, mailing lists and lending libraries are some obvious suggestions, but what about butterfly collecting, gardening, school

proc NewCollection; Filename\$

```
proc NewCollection;Filename$
  ree *** Creates a new file for stamp collection
  create Filename$ logical "S"
    Stamp_Ref_No
    Stamp_Type
    Page_No
    Position_No
    Mint
    Purchase_Dt$
    Cost
    Sale_Value
    Comments$
  endcreate
```

Having run these procedures with suitable file names as parameters we can start entering data about stamps. However a data entry procedure might be useful which could search for the appropriate stamp type name and if the type had not yet been entered allow both files to be updated together.

Other useful procedures would be one to print the catalogue of the whole collection and one to print a summary report of the total value of the collection. Here are my suggestions for these procedures, firstly to output the value of the collection:

proc FindValue

```
proc FindValue
  ree *** To show total value of album
  spoolon screen
  cls :CenPrint;"Total Cost and Value",80
  CenPrint;"-----",80; lprint
  let Total_Val=0; let Total_Cost=0
  all "S"
    let Total_Val=Total_Val+S.Sale_Value
    let Total_Cost=Total_Cost+S.Cost
  endall
  CenPrint;"Cost of collection = "+dec(Total_Cost,2,8),80
  CenPrint;"Estimated sale value of collection = "+dec(Total_Val,2,8),80
  spooloff : print : print :YesNo;"Okay?"
endproc
```

proc Catalogue

```
proc Catalogue
  ree *** Prints catalogue of collection to screen or printer
  print :YesNo;"Do you want the catalogue sent to printer?"
  spooloff : if not yes: spoolon screen : endif
  let yes=0
  while not yes
    YesNo;"Print catalogue in Country/Date order?"
    if yes
      use "T": order Country$a,Issue_Date$a; first
      use "S": order Stamp_Type$a;Heading;l
      while not eof("T")
        use "S": locate T.Type_Ref_No
        while S.Stamp_Type=T.Type_Ref_No and not eof("S")
          PrinStamp: next "S": endwhile
        next "T": endwhile
      lprint rept("-",79)
    else
      YesNo;"Print catalogue in Page/Position order of album?"
      if yes
        use "S": order Page_No$a,Position_No$a
        use "T": order Type_Ref_No$a
        Heading;2: first "S"
        while not eof("S")
```

```
      use "T": locate S.Stamp_Type
      PrinStamp: next "S": endwhile
      lprint rept("-",79)
    endif
  endwhile
  print :YesNo;"Okay?"
endwhile
Reorder: spooloff
endproc
```

proc Heading;N

```
proc Heading;N
  ree *** Output heading for catalogue

  let Ord$="Country/Issue-date"
  if N=2: let Ord$="Page/Position": endif
  cls :CenPrint;"STAMP CATALOGUE for album: "+Album_Name$,79
  lprint :CenPrint;"printed in "+Ord$+" order",79; lprint
  lprint : lprint rept("-",79);BlankLine
  lprint "iPage Pos.i STAMP NAME i Country i Issue Date i
          Cost i ValueiCat.Vali"
  lprint "i-----i-----i-----i-----i"
  lprint "i-----i-----i-----i-----i"
  BlankLine:BlankLine
endproc
```

proc PrinStamp

```
proc PrinStamp
  ree *** Outputs details of one stamp (current record)
  lprint "i";dec(S.Page_No,0,4);dec(S.Position_No,0,4);"
  i";:CenPrint;T.Stamp_Name$,17
  lprint "i";:CenPrint;T.Country$,13; lprint "i";
  CenPrint;T.Issue_Date$,12; lprint "i";
  CenPrint;dec(S.Cost,2,6)+" i"+dec(S.Sale_Value,2,6),14
  lprint "i";
  if S.Mint:CenPrint;dec(T.Mint_Value,2,6),7
  else :CenPrint;dec(T.Used_Value,2,6),7; endif
  if S.Comments$<>"" or T.Comments$<>""
    lprint "i"; lprint "i" i";
    Cenprint;S.Comments$+" / "+T.Comments$,49; endif
  lprint "i";BlankLine
endproc
```

Notice that the "Catalogue" line in the table, and the details of procedure calls other procedures individual stamps. to output the heading, a blank

proc AltStamp

```
*proc AltStamp
  ree *** Alter existing stamp record
  Which
  YesNo;"Do you wish to change the stamp type?"
  if yes: cls :GetType: let S.Stamp_Type=T.Type_Ref_No: endif
  use "S": alter : return
endproc

proc BlankLine
  ree *** Blank line in catalogue
  lprint "i" i" i" i" i" i"
  lprint "i" i" i" i" i" i"
endproc

proc Bye
  ree *** Closes all files that may be open
  while 1: close : endwhile
endproc
```

proc GetType

```

ren *** Finds appropriate stamp type (or enters it)
print : print "What is the type of stamp to be entered?"
use "T": let yes=0
while not yes
  print "Reply with stamp Type_Ref_No (if known)"
  print "      or part of Stamp_Name (not numerical)"
  print "      or press <ENTER> to enter new stamp type."
  print : input "Stamp type: ";Ans$
  if Ans$="":InpType: else
    let Ref_No=val(Ans$)
    if Ref_No<>0: locate Ref_No: display : else
      find Ans$: display
      while not yes
        if found(): sprint : print tab 12; " :YesNo;"OK (y/n)? "
        if not yes: continue : endif
        else : print "NOT FOUND *** ": let yes=1: endif
      endwhile
    endif
  endif
  sprint :YesNo;"Is this stamp type the correct one?": cls
endwhile
endproc

```

proc InpStamp

```

ren *** Input new stamp record
cls : print "INPUT"
use "S": last :ren *** Set up Ref. No.
let S.Stamp_Ref_No=S.Stamp_ref_No+1
GetType: let S.Stamp_Type=T.Type_Ref_No
let S.Mint=0: let S.Page_No=0: let S.Purchase_Dt$=""
let S.Cost=0: let S.Sale_Value=0: let S.Comments$=""
let S.Position_No=0
use "S": append : alter : return
endproc

```

proc InpType

```

ren *** Input a new stamp type
YesNo;"Are you sure a new stamp type is needed?"
if not yes: return : endif
let OK=1: use "T": last
let T.Type_Ref_No=T.Type_Ref_No+1
let Stamp_Name$="": let Country$="Great Britain"
let Issue_Date$="YYYY/MM/DD": let Mint_Value=0
let Used_Value=0: let Comments$=""
append : alter : return
endproc

```

proc Menu

```

ren *** Display and execute options
Reorder
while 1
  cls : print : print "Choose options:"
  print : print tab 10;"(I) Insert new stamp"
  print tab 10;"(A) Alter stamp"
  print tab 10;"(C) Output catalogue"
  print tab 10;"(V) Estimate value of collection"
  print : print tab 10;"(X) Exit (and close files)"
  let AS=inkey(): let AS=getkey()
  if upper(AS)="I":InpStamp: endif
  if upper(AS)="A":AltStamp: endif
  if upper(AS)="C":Catalogue: endif
  if upper(AS)="V":FindValue: endif
  if upper(AS)="X": error bye: stop : endif
endwhile
endproc

```

proc Start

```

ren *** Start-up procedure
mode 1,8: cls
YesNo;"Does the Stamp Catalogue file exist (stamp types)?"
input "Name for Stamp Catalogue file: ";N$
if yes:YesNo;"Need to modify?"
  if yes: open N$ logical "T": else look N$ logical "T": endif
  else :NewTypes;N$: endif
YesNo;"Does the Stamp Album file exist?"
input "Name for Stamp Album file: ";Album_Name$
if yes:YesNo;"Need to modify?"
  if yes: open Album_Name$ logical "S"
  else : look Album_Name$ logical "S": endif
  else :NewCollection;Album_Name$: endif
Menu
endproc
proc Which
ren *** Identifies the stamp for change
let yes=0: use "S": display
while not yes
  input "Stamp reference number? ";Ref_No
  locate Ref_No: sprint
  print "
  YesNo;"Correct stamp? "
endwhile
endproc

```

proc YesNo;P\$

```

proc YesNo;P$
ren *** Gets Y or N from keyboard and sets "yes"
while 1
  print P$: let Q$=lower(getkey())
  let yes=(Q$="y")
  if instr("ny",Q$): print " "+Q$
  return : endif
  print : endwhile
endproc

```

All the procedures also make use of the following more general purpose procedures for outputting a centred line of text, reordering the files, and obtaining a yes or no reply to a prompt.

proc CenPrint;S\$,Ln

```

proc CenPrint;S$,Ln
ren *** Centres or truncates string in given length
if len(S$)>Ln: lprint S$(1 to Ln): return : endif
let T=int((Ln-len(S$))/2)
lprint rept(" ",T);S$;rept(" ",Ln-T-len(S$));
endproc

```

proc Reorder

```

proc Reorder
ren *** Puts the files back in Ref. No. order
use "S": order Stamp_Ref_Noja
use "T": order Type_Ref_Noja
endproc

```

The procedures make use of the logical names of the files "S" and "T" given when the files were created. Whenever the files are opened the same logical names should be specified in the open or look command.

ARCHIVE

3 IN 1

DELTA DISK INTERFACE

FOR THE

QL

**SPECIAL
PACKAGE**

From

£210

UNIQUE 3 IN 1 FEATURES

- Double Density Disk Interface
- Centronic Printer Interface
- Memory Expansion

Delta 64 — 64 K extra RAM
Delta 128 — 128 K extra RAM

- Double density
- Support up to 4 Drives
- RAM Disk facility
- Centronic printer interface built in
- Compatible with Microdrive

- Compatible with most QL Software
- Professional quality and reliability
- Upgradable to

DELTA 64 - 64 K RAM
DELTA 128 - 128 K RAM



Technology Research Limited

Unit 18, Central Trading Estate, Staines, Middlesex TW18 4XE Tel: 0784 63547 Telex: 896691 TLXIR G

PLEASE SEND ME:

	Delta Basic	Qty	Delta 64	Qty	Delta 128	Qty	P&P
Interface only	£129.50		£175.00		£199.50		£4.00
Interface & 5¼" Single drive 40 track D.S.	£210.00		£245.00		£269.00		£8.00
Interface & 3½" Single drive 80 track D.S.	£250.00		£290.00		£310.00		£8.00
Interface & 3½" Twin drive 80 track D.S.	£350.00		£390.00		£410.00		£8.00
					Total		

*All prices include VAT and apply to UK only

Name _____

Address _____

Postcode _____

DIPPING IN

Assorted printer driver and DIP switch settings for the QL

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

Shinwa CP80 (type1)

None
9600
CR,LF
FF
ESC,"@,0
none
ESC,"E
ESC,"F
ESC,"-,1
ESC,"-,0
ESC,"S,1
ESC,"T,ESC,"H
ESC,"S,0
ESC,"T,ESC,"H
"£,129

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

Brother KP22

none
300
CR,LF
FF
none
none
none
none
none
none
none
none
none
none
"£,156

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1
Translate 2

Brother KP44

none
1200
CR,LF
FF
none
none
none
none
ESC,"E
ESC,"R
ESC,"U
ESC,"D
ESC,"D
ESC,"U
"£,35
"!,"",8,".

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

Brother HR-5

Space
300
CR,LF
FF
ESC,CR,"P
none
ESC,"E
ESC,"F
ESC,"-,1
ESC,"-,0
ESC,"S,0
ESC,"S,1
ESC,"S,1
ESC,"S,0
"£,35

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

Qume LP20

none
9600
CR,LF
FF
ESC,CR,"P
none
ESC,"Q
ESC,"R
ESC,"I
ESC,"J
ESC,"U
ESC,"D
ESC,"D
ESC,"U
"£,35

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

Smith Corona L-1000

none
1200
CR,LF
FF
ESC,"C,ESC,"9
none
DEF
DEF
ESC,"C
ESC,"R
none
none
none
none
"£,35

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1
Translate 2
[Red On]
[Red Off]

Brother HR15/25/35

None
9600
CR,LF
FF
ESC,CR,"P
none
ESC,"W
ESC,"&
ESC,"E
ESC,"R
ESC,"U
ESC,"D
ESC,"D
ESC,"U
"£,35
35,92,8,61
ESC,"A
ESC,"B

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Translate 1

SPG 8010

none
9600
CR,LF
FF
ESC,"R,4
none
ESC,"d
ESC,"e
ESC,"f,1
ESC,"f,0
ESC,"h,0
ESC,"h,1
"£,"\$

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off

Riteman II

EVEN
9600
CR
FF
ESC, "@"
none
ESC, "E
ESC, "F
ESC, "-", 1
ESC, "-", 0
ESC, "4
ESC, "5
ESC, "W, 1
ESC, "W, 0
"£, ESC, "I, "33, 6, ESC, "I, 48

OK Writer

none
1200
CR, LF
FF
24, ESC, 0
none
ESC, "H
ESC, "I
ESC, "C
ESC, "D
ESC, 28
ESC, 0
ESC, 31
ESC, 0
"£, 35

Diablo

none
9600
CR, LF
FF
none
none
ESC, 33
ESC, 34
ESC, "X
ESC, "Y
ESC, "G, 1
ESC, "Z
ESC, "G, 0
ESC, "Z
"£, 35

Brother M1009

SPACE
9600
CR, LF
FF
ESC, "R, 3
none
ESC, "E
ESC, "F
ESC, "-", 1
ESC, "-", 0
ESC, "S, 0
ESC, "T
ESC, "S, 1
ESC, T
"£, 35

Juki 6100

EVEN
300
CR, LF
FF
CR, ESC, 9, 0, ESC, "9
none
ESC, "W
ESC, "&

Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1
[NLQ On]

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1
[NLQ On]
[NLQ Off]

PRINTER

Parity
Baud Rate
End of Line Code
End of Page Code
Preamble Code
Postamble Code
Bold On
Bold Off
Underline On
Underline Off
Subscript On
Subscript Off
Superscript On
Superscript Off
Translate 1

ESC, "E
ESC, "R
ESC, 30, 6, "U, ESC, 30, 8
ESC, 30, 6, "D, ESC, 30, 8
ESC, 30, 6, "D, ESC, 30, 8
ESC, 30, 6, "U, ESC, 30, 8
"£, 35

Kaga Taxan & Canon PW1080A

EVEN
9600
CR, LF
FF
ESC, "R, 3
none
ESC, "E
ESC, "F
ESC, "-", 1
ESC, "-", 0
ESC, "S, 1
ESC, "T
ESC, "S, 0
ESC, "T
"£, 35
ESC, "("

Quen Data DWP 1120

SPACE
300
CR, LF
FF
ESC, 26, "I
none
ESC, "W
ESC, "&
ESC, "R
ESC, "D
ESC, "U
ESC, "U
ESC, "D
"£, 35

Epson LX-80

none
9600
CR, LF
FF
ESC, "@"
ESC, "R, 0
ESC, "E
ESC, "F
ESC, "-", 1
ESC, "-", 0
ESC, "S, 1
ESC, "T
ESC, "S, 0
ESC, "T
"£, 35
ESC, "x, 1
ESC, "x, 0

Daisystep 2000

none
1200
CR, LF
FF
none
none
ESC, "Q
ESC, "&
ESC, "R
ESC, "U
ESC, "D
ESC, "D
ESC, "U
"£, 35

OWNER'S MANUAL

Dip Switch Settings

Daisystep 2000
Printer

Manufacturer's
Interface

A1	On	B1	-	SW1	Left
A2	On	B2	On	SW2	Left
A3	On	B3	On	SW3	Right
A4	On	B4	On		
A5	On	B5	On		
A6	On	B6	On		
A7	On	B7	On		
A8	On	B8	On		

Juki 6100

Printer Manufacturer's
Interface

A1	Off	SW1	On	A7	Off
A2	Off	SW2	On	A8	On
A3	On	SW3	Off	A9	Off
A4	Off	SW4	Off	A10	Off
A5	On	SW5	Off		
A6	On	SW6	On		

Riteman II
Printer

Left Right Upper Right

L-1	On	R-1	On	UR-1	On
L-2	On	R-2	On	UR-2	On
L-3	On	R-3	Off	UR-3	On
L-4	On	R-4	Off	UR-4	Off
L-5	Off	R-5	On	UR-5	Off
L-6	Off	R-6	On	UR-6	Off
L-7	Off	R-7	Off	UR-7	Off
L-8	Off	R-8	Off	UR-8	Off
				UR-9	Off
				UR-10	On

Epson LX-80
Printer

1-8	Off	2-4	Off
1-7	Off	2-3	Off

1-6	On	2-2	On
1-5	Off	2-1	Off
1-4	Off		
1-3	Off		
1-2	Off		
1-1	Off		

Brother HR-5
Printer

1-1	Off	2-1	On
1-2	On	2-2	On
1-3	Off	2-3	Off
1-4	On	2-4	Off
1-5	Off	2-5	Off
1-6	On	2-6	Off
1-7	Off		
1-8	Off		

Brother M1009
Printer

Version K	Later Versions	Version K	Later Versions
1-1	Off	1-1	On
1-2	On	1-2	On
1-3	On	1-3	On
1-4	On	1-4	On
1-5	On	1-5	On
1-6	On	1-6	Off
1-7	Off	1-7	Off
1-8	Off	1-8	Off

Kaga Taxan & Canon PW1080
Printer Manufacturer's
Interface

1-1	On	2-1	Off	3-1	Off
1-2	On	2-2	Off	3-2	Off
1-3	Off	2-3	On	3-3	Off
1-4	Off	2-4	Off	3-4	Off
1-5	Off	2-5	Off		
1-6	On	2-6	Off		
1-7	On	2-7	Off		
1-8	Off	2-8	Off		

QfLash presents RAM disk for the QL

- * The ultimate in speed, reliability, features, and compatibility
- * Seven times faster than other RAM disks
- * QL screen image (32768 bytes) loads in 0.15 seconds!
- * I/O-intensive programs run up to 100 times faster
- * No bugs reported since initial release in August 1985
- * Full dynamic memory allocation - no FORMAT command necessary. No memory reserved for file data unless really used for your files
- * Up to 8 RAM devices can be used with up to 255 files per device
- * Works on standard (128K) QL and with all QL software, and takes full advantage of expansion memory up to the 640K maximum
- * Just as easy to use as microdrives of floppies, but much faster
- * Incredible low price £27.00 or DM 97.00 (all included)

Send Eurocheque to QfLash, Post box 102121, D-2 Hamburg 1,
W-Germany or call (Hamburg) 6512742 or 7650461

A QUIZZICAL AND PUZZLING CHRISTMAS? FUN FOR ALL THE FAMILY WITH

QL Quiz. A single user and competitive 2-4 player quiz, over 1000 questions, with powerful editor option to alter rules, add questions, make your own quiz **£20.00**

Gumshoe Logic. Logic puzzle program set in a detective agency, 5 levels of difficulty, randomised cases **£12.00**

Cheques, POs to:

Megacycal Software
PO Box 6, Birkenhead, Merseyside, L43 6XH

INVESTORS!

Track your shares with
"STOCKMARKET MANAGER"
from Portfolio Software

**PORTFOLIO
SOFTWARE**

New, Improved Version! Facilities include:

- Simple entry of purchases, sales, prices etc.
- Portfolio valuation - analysed by investment type.
- comparison of performance against market.
- Calculation of % return on each investment.
- Assessment of CGT liability.
- Handles all investment types, including Traded Options.

Order now at special price of **£34.95 incl. p&p**

ARE YOU SURE YOU'RE INSURED?

Get the best value in house insurance with this new product from Portfolio Software "INSURANCE MANAGER". Facilities include:

- Quantify total contents value by risk category
- Database of key insured items, incl. details of purchase, serial number, insured value etc.
- Evaluate best quotation from competing companies.

Special introductory price of **£24.95 incl. p&p**

**PORTFOLIO SOFTWARE -
THINKING SOFTWARE FOR PRACTICAL USERS**

Orders/cheques to PORTFOLIO SOFTWARE, PO BOX 15, LONDON SW11 5RP

Please send: "STOCKMARKET MANAGER" @ £34.95 incl. ☐ No. of copies
"STOCKMARKET MANAGER" Information Sheet ☐
"INSURANCE MANAGER" @ £24.95 incl. ☐
"INSURANCE MANAGER" Information Sheet ☐ (Enclose large SAE)

Overseas mail add £2.50 Europe; £5.00 elsewhere

Name:

Address:

QLU 12/85



Introduces Q-Disc WINCHESTERS

10, 20 or 40 Megabyte
Winchester disc systems
with optional
3.5in. floppy disc drives



PRICE LIST (excluding VAT)

10Mb system only . . .	£1100
10Mb with one Floppy .	£1250
10Mb with two Floppies	£1400
20Mb system only . . .	£1500
20Mb with one Floppy .	£1650
20Mb with two Floppies	£1800
40Mb system only . . .	£2700
40Mb with one Floppy .	£2850
40Mb with two Floppies	£3000

CST are pleased to announce yet another first for the Sinclair QL.

A range of 10, 20 and 40Mb Winchester systems with optionally one or two 3.5 inch 720K Floppy disc drives and high speed tape streaming output, in one enclosure.

These systems are designed for the professional office environment and care has been taken to avoid unsightly cable connections.

With the exception of the 10Mb unit without floppy, the QL power supply is built in and there are 3 mains outlets at the back for monitor, printer, graph plotter etc...

Winchesters can store huge amounts of valuable information; for example a 10Mb unit can store as much as approximately 100 microdrives. Good back up is essential when relying on one storage facility. Hence CST has developed a simple yet sophisticated system to provide easy backup of data on floppy disc, high speed streaming tape or microdrive.

The CST Winchester systems were developed with close co-operation from Sinclair Research Ltd and are fully compatible with the QL, QDOS, Psion application programs and all planned Sinclair peripherals and software.

For applications requiring more than one interface, the CST Q+4 Expansion system can be used (as illustrated above), to connect up to four interfaces to the QL.

If you have any questions as to which system would be most suitable for your application, please do not hesitate to contact us.

Orders To

24 Green Street
Stevenage

Hertfordshire SG1 3DS

Telephone (0438) 352150



Cambridge Systems Technology

PART 3 The Commands

Last month we developed a number of general library routines for use within our programs. In so doing we looked at how to enter information on the computer (*gen_in*), how to display it (*prompt*) and how to store and retrieve it using DATA statements or files on microdrive cartridges. Here using similar techniques we examine how the QL understands things, makes decisions about them and manipulates them.

NUMBER CRUNCHER

Everything on a computer is dealt with numerically. Where we are dealing with numbers as opposed to characters this poses few problems. Simply regard the QL as a glorified calculator with virtually unlimited memory. Arithmetic operations need not be entered step by step but may appear all together on a single line:

PRINT 3+2*5+7

When entering complex formulae on the QL there are two points to note. First, the order of evaluation is determined by the priority of the arithmetic operator (see Table 1).

PRIORITY	SYMBOL	DESCRIPTION
1.	^	Raise to the power
2.	*	Multiplication
	/	Division
	DIV	Integer Division
	MOD	Integer Remainder (ie modulus)

3. + Addition
- Subtraction

Table 1: Arithmetic Operators

Second, whilst the QL is capable of handling numbers over a range from -10^{616} to $+10^{616}$ it is accurate to only SEVEN SIGNIFICANT FIGURES. Anything in excess will be rounded up and expressed as a seven figure number followed by its exponent. For example:

PRINT 1234567
gives 1.234567E6

CHARACTER CODE

When it comes to how the QL treats characters, things are a little harder to understand. What may appear to us as a letter, punctuation mark and even mathematical symbol is to the QL a uniquely coded number with a value between 32 and 127. This value is set according to a standard international code known as ASCII (see *Concepts Section* in User Guide). To ascertain that this is the case enter:

PRINT CODE("A")
this will display the value 65.
PRINT CHR\$(65)
this will display "A".

CHR\$ converts the code back into the character.

One interesting feature of the ASCII code is that a difference of 32 separates each upper case letter from its lower case equivalent. This means that if you want to convert any character from upper to lower or vice-versa you need only apply the following

formulae respectively:

lower\$ = CHR\$(CODE
(any_capital\$)+32)
upper\$ = CHR\$(CODE
(any_lower\$)-32)

often this is written as:

lower\$ = CHR\$(CODE
(any_capital\$)+CODE
("A")-CODE("a"))
upper\$ = CHR\$(CODE
(any_lower\$)+CODE
("A")-CODE("a"))

Using this formula we can add a further procedure to our library (*Listing 1*) which when passed a string will change the case of each character within it, so that, a lower case character becomes an upper case one and vice versa. As usual the amended string will be returned in the variable *temp\$*.

STRING STORAGE

To recap briefly then, when we assign a collection of characters to a string variable, each is coded and then stored in sequence. The amount of memory that a string occupies depends on its size and is not fixed as it is with a numeric or integer variable. In fact, the only limitation upon a string's length is the amount of free memory available (where each character occupies one byte of memory).

Once characters have been grouped together in a string they may be manipulated in all sorts of ways. For example, where:

a\$ = "blackredgreenwhite"
b\$ = "bluemagentacyan yellow"

we can join strings together (concatenate) using the & operator.

all_colour\$ = a\$ & b\$
(ie **all_colour\$ = "blackredgreen whitebluemagentacyanyellow"**) or else we can extract a portion of a string. If a group of characters are required we must specify the positions of the first, and last characters within the string.

col1\$ = a\$(1 TO 5)
(**col1\$ = "black"**)
col2\$ = a\$(6 TO 9)
(**col2\$ = "red"**)

col3\$ = a\$(1 TO 5)&" and
"&b\$(1 TO 4)

(**col3\$ = "black and blue"**)

If we wish to extract a single character we need only give its position in the string.

char1\$ = a\$(17)
(**char1\$ = "t"**)
char2\$ = b\$(20)
(**char2\$ = "w"**)

Just as we can extract portions from a string we can also replace portions within it

a\$(9 TO 13) = "ochre"
(will replace "green" with "ochre" **a\$ = "blackredochre white"**)

However, when altering an existing string using this method it is important to bear in mind that you cannot increase the amount of memory allocated to it. In other words you cannot change its size. For example an attempt to replace "white" in string *a\$* with "tangerine" (ie, **a\$(14 TO 23) = "tangerine"**) would result in only the first five letters "tange" being moved in. The reason for this is that a string's size is set only when characters are assigned to the WHOLE of it. Replacement of some of its parts will not increase the memory allocation. So, if we want to add "tangerine" we would use the following method:

a\$ = a\$(1 TO 13)&"tangerine"

The length of a string can be found using the function LEN(string).

PRINT LEN("Sinclair QL")
will display a value of 10
str_len% = LEN(b\$)
will restore length of the string *b\$* in *str_len%*

For most complex string manipulations LEN plays a vital role. For example, SuperBasic unlike many other dialects does not have a RIGHT\$ function. This permits you to slice strings up from right to left. However, using LEN such a function may be constructed with:
slice\$ = a\$(LEN(a\$)+1- characters_to_slice TO LEN(a\$))

The command INSTR, short for "IN STRing", allows us to search through a string to see whether another string is contained within it. It returns a value of zero if no match is found otherwise the position of the first character in the match will be returned.

LISTING 1

```
30000 REMark *** Case convertor ***
30010 DEFine PROCedure case_conv(str$)
30020 LOCAl f
30030 temp$=str$
30040 FOR test = 1 TO LEN (str$)
30050 f = CODE(str$(test))
30060 IF f>64 AND f<91 THEN temp$(test)=CHR$(f+32)
30070 IF f>96 AND f<123 THEN temp$(test)=CHR$(f-32)
30080 END FOR test
30090 END DEFine
```

M1 = "cyan" INSTR b\$...

(returns 12 ie "

"bluemagentacyanyellow")

M2 = col2\$ INSTR all_colour\$
(returns 19)

M3 = "blue" INSTR a\$

(no match - returns 0)

Often it is necessary to set up strings containing a large number of identical characters. Assigning each character in turn is both repetitive and prone to error, particularly where spaces are concerned. However, provided no more than two characters are to be repeated the function **FILL\$** may be used. You simply specify the character(s) to fill with and the length of the string to be filled.

blank\$ = FILL\$(" ",50)
(creates a string with 50 spaces)

asterisk\$ = FILL\$(" ",5)
(asterisk\$ = "*****")

mix\$ = FILL\$(" - ")

(mix\$ = " - - ")

A possible application using **FILL\$** would be to centre a string on paper. Assuming you are using an 80 column printer this would be:

temp\$ = FILL\$(" ",(80/2-LEN(a\$)/2))&a\$:PRINT temp\$

of course, a more efficient alternative would be to use **AT**

AT (80/2-LEN(a\$)/2):PRINT a\$

Coercion (mentioned briefly in Part I) provides a bridge between different data types. It permits numbers to be converted into strings and strings to be converted into numeric values provided, in the latter case, that there is no type mismatch. Additionally, you can convert floating point numbers into integers and vice versa. The following examples illustrate what may be achieved.

1. Integer to floating point

10 **x% = 237: y% = 5**

20 **result = x% / y%**

2. Floating point to integer

10 **fl_point = 678.54**

20 **integer% = fl_point**

It is worth noting that numbers are rounded and not truncated as with the **INT** function.

3. Numeric to string

10 **number = -113.675**

20 **string\$ = number**

30 **PRINT string\$(5 TO 8)**

4. String to floating point or integer

10 **sale\$ = "23.50":**

cost\$ = "12.45"

20 **profit = sale\$ - cost\$**

30 **NP% = (100*profit)/sale\$**

If a string holds non-numeric characters then conversion will not be possible. For instance **value = "£123.45"** will generate an error as the pound sign is non-numeric and consequently unacceptable.

Aside from **INPUT** routines, this type of coercion is most commonly encountered where time or calendar date are incorporated into programs.

CLOCK AND CALENDAR

The QL's internal clock (and calendar) works only when the machine is on. As a consequence the date and time must be set at the beginning of every session. This is done using the function **SDATE**. It takes the form:

SDATE year,month,day,hour,minute,second

All values are numeric. However for reasons mentioned in the last section, if these values are to be **INPUT** they need to be entered as strings, checked and then coerced into their numeric equivalents. The library routine given in *Listing 2* shows how this may be done (note that it makes use of the procedures given in Part 2 of this manual).

Once the current date and time have been entered, the QL will keep track of them until it is switched off. To get time and date to display we use the function **DATE\$**. This returns a string laid out as follows "yyyy mmm dd hh:mm:ss". In this case coercion would be required to process each bit of information. For example:

10 **birthyear% = 1957**

20 **now\$ = DATE\$**

30 **years_old% = now\$**
(1 TO 4) - birthyear%

An alternative to using coercion is to use the function **DATE**. This measures time in seconds and provides an economical way of storing information on file. **DATE** may be converted into a readable format using **DATE\$**. The following program illustrates how this works:

10 **SDATE 1986,1,1,0,0,0**

20 **PRINT "The date and Time is"!DATE\$: now = DATE**

30 **PRINT DATE!"seconds have elapsed since"!DATE\$(0)!**

As shown above **DATE** is measured in the number of seconds that have elapsed since "1961 Jan 1 00:00:00".

Incidentally, the furthest into the future you can go is 2/2/2097.

The command **ADATE** allows the QL's clock to be fine tuned. Adjustments are entered in seconds.

ADATE 60

(forwards 60 secs)

ADATE -43200

(back 12 hours)

Another useful clock function is **DAY\$**. This will return the day

from the internal clock (Mon,Tue,etc).

PRINT DAYS

(will display current day)

PRINT DAYS(0)

(displays 'Sun' being 1/1/1961)

This ties up our examination of SuperBasic. In the three installments of this manual we have tried to cover the areas which have hardly been touched upon by Sinclair's User Guide. The object has been to shine some light on the unknown rather than restate the obvious. We end with a word of encouragement to those new to programming. Remember over 90% of all programs ever written have bugs in them!

LISTING 2

```

30000 REMark **** Date Entry ****
30010 DEFine PROCedure date_entry(ch,x,y)
30020 LOCal loop,dte,yy$,mm$,dd$
30030 mm$="":yy$="":dd$=""
30040 REPEAT d_inner
30050   dte=0
30060   prompt "Enter Day",1,5,40
30070   gen_in dd$,2,ch,x,y:dd$=temp$
30080   prompt "Month",1,5,40
30090   gen_in mm$,2,ch,x+3,y:mm$=temp$
30100   prompt "Enter year in full",1,5,40
30110   gen_in yy$,4,ch,x+6,y:yy$=temp$
30120   temp$=yy$&mm$&dd$
30130   FOR loop=1 TO LEN(temp$)
30140     dte = temp$(loop) INSTR "1234567890"
30150     IF NOT dte THEN EXIT loop
30160   END FOR loop
30170   IF dte
30180     IF dd$>0 AND dd$<32
30190       IF mm$>0 AND mm$<13
30200         IF yy$>1984 AND yy$<1999
30210           EXIT d_inner
30220         END IF
30230       . END IF
30240     END IF
30250   END IF
30260 END REPEAT d_inner
30270 SDATE yy$,mm$,dd$,0,0,0
30280 AT #ch,y,x:CLS #ch,4:temp$=DATE$
30290 prompt temp$(1 TO 12)&DAY$,1,5,40
30300 END DEFine date_entry

```


PRINTERLAND



MAIL ORDER ONLY

Free...

WITH ALL PRINTERS: QL USERS PRINTERS GUIDE

We supply a free booklet with all printers which explains how to obtain all the features available on your printer.

■ DOT MATRIX	EX VAT	INC VAT
SHINWA CP A80	£165-00	£189-75
EPSON FX80 F/T+	£314-00	£361-10
EPSON FX100 F/T+	£425-00	£488-75
■ DOT MATRIX PLUS NEAR LETTER QUALITY		
EPSON LX80 SPECIAL OFFER	£195-00	£224-25
CANON 1080A SPECIAL OFFER	£235-00	£270-25
JUKI 5510 (colour option)	£249-00	£286-35
CANON 1156 17" CARRIAGE	£335-00	£385-25
■ DAISY WHEEL		
QUENDATA 1120	£225-00	£258-75
EPSON DX100 SPECIAL OFFER	£356-00	£409-40
■ COLOUR PRINTERS		
EPSON JX-80 SPECIAL OFFER	£450-00	£517-50
■ PRINTER INTERFACES		
MIRACLE SYSTEMS	£26-05	£29.95
■ COMPUTERS		
SINCLAIR QL ONLY	£173-05	£199-00

■ DISC DRIVES	EX VAT	INC VAT
Prices include power supply and interface.		
MICRO PERIPHERALS		
3 1/2" SINGLE DRIVE SYSTEM (.75 MBYTE)	£241-00	£277.15
3 1/2" DUAL DRIVE SYSTEM (1.5 MBYTE)	£365-00	£419-75
PCML DISK INTERFACE + 256K RAM & TOOLKIT		
INTERFACE	£240-00	£276-00
3 1/2" DUAL DRIVES	£199-00	£228-85
■ TYPEWRITER PRINTER COMBINED		
JUKI 2200	£245-00	£281-75
■ MONITORS		
PHILIPS 7502 GREEN	£75-00	£86-25
MICROVITEC CUB 1451/653	£220-00	£253-00
■ MODEMS		
TANDATA (complete system including		
FREE TELECOM GOLD/PRESTEL)	£173-00	£198-95
■ SOFTWARE	RING FOR BEST PRICES	

**COMPLETE PACK
SPECIAL OFFER**

THE WORLDS BEST PRICES ON QL/MONITOR/PRINTER PACKS

Package includes: SINCLAIR QL, MICROVITEC CUB563, PRINTER & ALL LEADS AND INTERFACES
 WITH EPSON LX80 F/T £775-00 £891-25 WITH CANON 1080 £812-00 £933-80
 WITH CANON 1156 £900-00 £1035-00 OTHER PRINTERS ON REQUEST

**EDUCATIONAL · GOVERNMENT
OVERSEAS ORDERS WELCOME**

PRINTERLAND

NOW ONLY 3 MINUTES FROM JUNCTION 23 M62
 (MANCHESTER 25 minutes / LEEDS 20 minutes) NB. FROM EAST USE JUNCTION 24
 156 LONGWOOD GATE, LONGWOOD

Tel HUDDERSFIELD (0484) 646048/9

**FOR
THOSE 'GREAT'
MICRODEAL GAMES**

**JOYSTICKS
& INTERFACES**

QL



INTERFACES only £4.75

Allows any ATARI or COMMODORE Joystick to plug into the QL

QUICKSHOT II Joystick £9.95 For above interface

SPECIAL OFFER both the above £12.95

Order Form by POST or PHONE 0726 68020
 with your Credit Card No.



Name

Address

Postcode

MICROPOST 41 Truro Rd
 St. Austell, Cornwall
 PL25 5JE

MACHINE CODING

PART 2

The QL holds many secrets, which can only be unlocked in response to commands presented at the very lowest levels near the QDOS retreat...

The machine code programmer talks to QDOS generally in one of two ways. The first is by executing a particular 68000 TRAP instruction. The traps used by QDOS are #0, #1, #2, #3, and #4.

Trap #0 is very basic and simply causes the processor to enter supervisor mode. This is often useful when we are inside jobs, which normally run in user mode, as it allows us to alter the interrupt state and move the system stack.

Trap #1 provides a group of functions known as the 'manager' routines. These look after the major parts of the system such as jobs, memory allocation and system lists. Each trap #1 routine is entered with the function number in register D0, and possible parameters in registers D1 to D3 and A0 to A3. If the routine returns and results, these will be found in the same group of registers, and errors are reported generally by setting register D0 to a negative error code value.

JOBS AND TREES

If we take a look at the routine which scans a job tree, *MT.JINF*, for example, we find that it is entered by putting 2 in to D0 before the TRAP #1 instruction is executed. Other values it requires are the 'job IDs' of the job being scanned (in register D1) and the job at the top of the tree (in D2). When this routine returns, D1 holds the job ID of the next job in the tree, and registers D2, D3 and A0 have been set to hold values relevant to the job which was scanned.

The register usage, parameters required, results returned and possible error conditions are given in the QL Technical Guide, along with a very brief description of each routine's purpose. Trap #2 and trap #3 follow much the same lines as trap #1, except that each has a slightly different purpose.

Trap #2 is used for I/O Allocation, which means the opening and closing of channels. This trap also incorporates file deletion and medium formatting. The trap #2 routine to open a channel, *IO.OPEN*, is called by putting 1 in to D0 before executing the TRAP #2 instruction. It needs a fair amount of information before it can open the channel. D1 is used to hold the job ID of the job which will 'own' this channel, D3 holds a 'key' which determines how a channel should be opened (ie, read only, update and so on) and register A0 points to the name of the file or device to be opened. If the open is successful, then D0 holds 0 and register A0 holds the 'channel ID' of the newly-created channel.

Trap #3 deals with all the general-purpose I/O routines, allowing us to read and write single bytes or strings. Console I/O also has a special 'edit a line' routine which allows a previously-typed line to be edited by the user at the keyboard. Apart from these routines, trap #3 provides access to the enormous number of screen driver routines such as window panning and scrolling, ink, paper and strip colour changes, border alterations, circle and line drawing and recolouring. The final few trap #3 routines provide access to specialist file-handling operations which allow things like reading and setting the header on a file, moving the read/write pointer to any given position in a file (and, therefore, providing random access) and saving or loading complete files. Trap #3 is probably the most used QDOS routine.

Trap #4 is like trap #0 in that

it has one purpose only in life: to tell the next trap #2 or trap #3 routine that the addresses being passed to it will be relative to register A6 rather than absolute. It does this by setting a byte in the job's header.

VARIED VECTORS

The other way of accessing various QDOS routines is via a 'vector'. A vector is a pointer to somewhere else, which may seem a bit strange, but its advantages are obvious. If the system is being updated often then it is very likely that the address of each routine in the system will alter from time to time. As information needs to be published about where these routines are, and as other routines need to call them, the fact that they alter often is rather inconvenient. But, if the addresses of each routine are placed in special pre-defined locations in memory which never change from version to version, it is extremely easy to access each routine. These special locations are the vectors, and the addresses inside the vectors are entirely irrelevant to us. We call the vectored routine by grabbing the address from the vector and then calling it as a sub-routine or jumping to it. The easiest way of doing this is as follows:

```
MOVE.W _the_vector,A2
JSR (A2)
```

There is no reason why A2 in particular should be used, but it is generally speaking the most obvious choice. We move only a word in to the address register from the vector because each vectored routine is in ROM, which always starts at address \$00000000 on a QL. Things are specially arranged to ensure that each of the routines starts at an address less than \$00008000, which allows word-long addresses to be used via the 'absolute short' addressing mode.

The vectored routines which exist provide many facilities. Some of them almost duplicate trap-invoked routines but have simpler parameters or can be

used in certain situations only; other provide entirely new facilities which are useful to have around. Many of the vectored routines are used mainly by the SuperBasic interpreter, as they refer to the SuperBasic variables such as the name table and name list. Other routines are useful in all sorts of applications: the routines which convert numbers into dates and days of the week, for example, or those which convert numbers into strings and vice versa. There are routines even to create and maintain linked lists, perform floating point arithmetic operations and compare two strings. It seems that a good deal of applications software could be written entirely using QDOS vectored utility routines!

CHANNELS AND IDs

In order to be able to refer to a given channel or job uniquely, the operating system needs to have a number for that channel or job. As both jobs and channels may be created and destroyed asynchronously, a number which increases on a per-job or per-channel basis is insufficient by itself. QDOS uses the same system for both jobs and channels, and the numbers used are referred to as the job or channel IDs. Each ID is held in a long word, which means four consecutive bytes (thirty-two bits), and comprises two parts. The lowest two bytes (sixteen bits) form an incremental number between zero and a given upper limit which refers to the channel or job's position in its requisite table. The channels opened in a row, for example, may have positions two, three and four in the channel table. If channel two is then closed and a further channel opened, the new channel will be given position two in the table as it is the first empty slot. This position is often referred to as the job or channel number, but has nothing to do with the '#' channel numbers used by the SuperBasic interpreter.

As this intimates, it is very easy for a channel to be closed

and another to be opened in its place before a job talking to that channel knows about the closure. If the job is reading from or writing to the channel it obviously will get very confused. To avoid such things, the high word (sixteen bits) of the channel or job ID holds a cyclic 'tag' which is allocated by the operating system. So, to take our case of three channels being opened in quick succession, they may still be given numbers two, three and four, and the tags might be four, five and six.

Now, when channel two (with tag 4) is closed and the new one opened, the new channel two will have a tag of seven. The operating system can see straightaway that the new channel is different from the first, so when a job attempts to com-

municate with the old channel two it will be told that this channel is not open. As these principles apply to jobs as much as to channels, the job ID mechanism is the same as the channel ID mechanism.

LABELS AND TABLES

To find the position in memory of a job, given its job ID, we need to know only a few things. If we assume that register A6 points to the base of the system variables, the code below will return in A0 the base address of the jobs whose ID is held in register D1. This base address is the address of the start of the code immediately after the job header.

JB_TAG	EQU	\$10
SV_JBBAS	EQU	\$68
JB_END	EQU	\$68
SV_JBTOP	EQU	\$6C

```

MOVE.W D1,D4
EXT.L D4
LSL.L #2,D4
ADD.L SV_JBBAS(A6),D4
CMP.L SV_JBTOP(A6),D4
BCC.S BAD_JOB
MOVEA.L D4,A0
MOVE.L (A0),D4
BMI.S BAD_JOB
MOVEA.L D4,A0
SWAP D1
CMP.W JB_TAG(A0),D1
BNE.S BAD_JOB
LEA.L JB_END(A0),A0
    
```

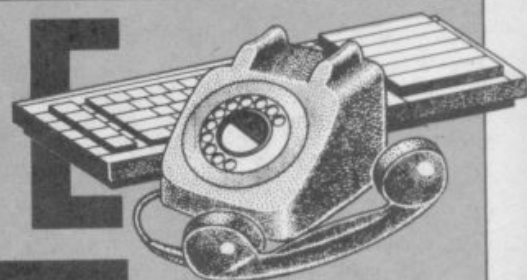
*A0 holds job's base address

The label BAD_JOB is jumped to if the job ID passed in register D1 is invalid in any way. Apart from finding a job in memory, this routine also provides a convenient way of telling if a given job exists or not. As much the same techniques apply to channels, similar code may be used to discover if

given channels are open, or where their channel definition blocks are. Obviously, the system variables used will need to be changed.

Immediately before the LEA.L JB_END(A0),A0 instruction, register A0 holds the address of the job's job header. This header contains a great deal of useful information which may be extracted and in certain cases directly altered. The information held here includes such things as the job's length, its owner, its priority and its suspension status, a pointer to its trap re-direction vectors and a copy of all the 68000 registers except the supervisor stack pointer. These register entries are particularly useful as they allow us to put things on a job's stack before it is activated, or to set up the registers.

HELPLINE



With the recent price drop the QL has now become a hacker's machine *par excellence*. Its uses are no longer limited to running so-called 'serious' applications such as Psion's four packages but extend to cover games playing, advanced programming, enhancing QDOS, dabbling in AI and linking up to a host of peripherals.

Not surprisingly, a host of new applications have generated new problems for QL owners. What scant technical documentation exists for the machine deals almost exclusively with QDOS and is geared for the expert. Furthermore outside information services from Psion (01 723 9408) and Sinclair (QLUB), often loathe to admit to faults in their respective products, have singularly failed to keep pace with an ever more demanding user base.

All this adds up to the fact that unless you have an enlightened philanthropic dealer as your

constant companion, you have nobody to turn to when the going gets rough on your QL! Well, for those who have subscribed or who registered with *QL User* magazine before May this simply is not true! There is the Technical Helpline.

The service is available after working hours and promises a 7 day turnaround on individual queries. However, before you pick up the phone and pour out your troubles to our resident expert bear in mind that the service is intended as the last recourse when all else has failed.

So, before you ring in ensure that you have taken the following steps:

Re-read the *Beginner's* section of the User Guide and check that you haven't committed something glaringly wrong like trying to FORMAT a microdrive with its 'write/protect' tab broken off. Many problems derive from the obvious overlooked.

Look-up the problem area in the *Concepts* section of the guide and, if necessary, make sure that you understand precisely what is meant by say, a 'file', and what the difference is between a 'function' and 'procedure' before you try solving a problem at a lower and more involved level.

Finally, if possible, see if you can reproduce your error upon a friend's or local dealer's machine! It may well be that the version of the QDOS or the peripherals attached to your machine interfere with its correct running. If possible try and compile a check-off list showing at what stage software works and at what stage it ceases to do so.

If these steps fail to solve your problems, you will at least be in a position to articulate your query clearly and with that half the battle will be won. After all we can only provide answers if we understand the question.

The procedure after dialling the Helpline (01 250 0350) is:

Give your name and address. State whether you are a subscriber or registered reader and in the former case quote your subscription number. After this state your problem as succinctly as possible and give information as to the circumstances in which the error occurs as well as details of your system — for example, the version of the ROM and what peripherals are attached.

One final point, because the helpline is run by an outside agency and not *QL User* staff, it is not the best way to get messages through to this magazine. The telephone number on page three of every issue is far better.

Did you miss Volumes I & II of the QL Owner's Manual? Then ring 01-251 6222 and ask for QL User Backnumbers