

SINCLAIR

TOP User Guide

sinclair

QL

User Guide

IMPORTANT WARNING

THIS MANUAL IS FOR INFORMATION ONLY.

ITS CONTENTS ARE PROVISIONAL AND MAY BE SUBJECT TO CHANGE PRIOR TO ITS SUPPLY WITH THE QL COMPUTER.

AS A FULL UP TO DATE MANUAL WILL BE SUPPLIED WITH EVERY QL COMPUTER WE STRONGLY RECOMMEND THAT THIS DOCUMENT BE DESTROYED WHEN IT HAS SERVED ITS PURPOSE.

DISCLAIMER

In no circumstances will either Sinclair Research Limited or PSION Limited be liable for any direct, indirect, incidental or consequential damage or loss including but not limited to loss of use, stored data, profit or contracts which may arise from any error, defect or failure of the QL hardware or the software supplied with it.

Sinclair Research has a policy of constant development and improvement of their products. Therefore, the right is reserved to change manuals, hardware, software and firmware at any time and without notice.

WARNING

To minimise the possibility of a loss of data it is strongly recommended that periodic copies (or backups) are made of Microdrive cartridges containing data or programs.

© 1984 Sinclair Research Limited

© 1984 PSION Limited

sinclair

QL

User Guide

Introduction

Beginner's Guide

Reference Guide

Keywords

Concepts

Applications Software

QL Quill

QL Abacus

QL Archive

QL Easel

Information



OL User Guide

- Introduction
- Registrar's Guide
- Registrar's Guide
- Keywords
- Concepts
- Application Software
- OL User
- OL Admin
- OL Archive
- OL Email
- Information

sinclair

QL

Introduction

PROVISIONAL

Unpacking

When you unpack your QL you will find:-

- The QL User Guide
- The QL computer
- A power supply
- An aerial lead

this is about two metres long and has different sockets at either end.

- A network lead

this is also about two metres long but has the same type of connector at each end.

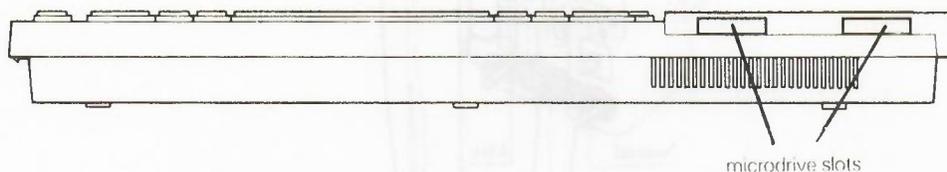
- A Microdrive Wallet
- A Microdrive Wallet
- Three Plastic Feet

containing the QL Application Software.
containing 4 blank Microdrive cartridges.

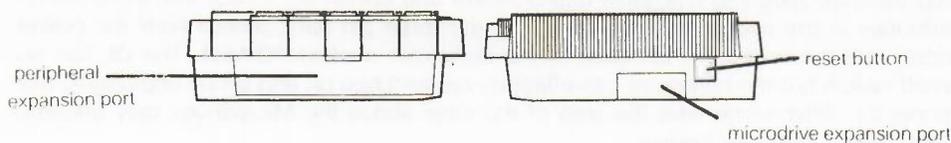
A Guided Tour

On the back and sides of the computer there are a series of connectors. Most of these are not required initially but are used for plugging other peripherals into the QL (a peripheral is a piece of equipment which can be plugged into computer to expand its capabilities).

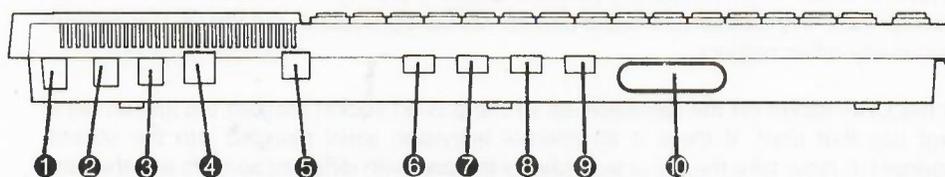
On the front right hand side you will find two slots, these are the two Microdrives. Always make sure these two slots do not have any microdrive cartidges in them when the computer is switched on. Microdrive cartridges are used for storing programs and data for the QL. Below each slot there is a small light. When the light is on the Microdrives are in use and the cartridges should not be removed. There is another light on the front left hand side, this indicates if the QL is on.



On the right hand edge is another connection which is covered by a plastic strip. This is used for connecting up to six more Microdrives to the computer if you need to expand the storage capacity of your QL in the future. **ZX Spectrum Microdrives are not suitable for use in the QL or with QL Microdrives.**



On the left hand edge of the computer there is a large slot. This is used to add expansion peripherals to the computer for example more memory, sound generators, hard discs, etc.



- | | | | |
|---------|---------|----------|---------|
| 1. Net | 2. Net | 3. Power | 4. RGB |
| 5. UHF | 6. SER1 | 7. SER2 | 8. CTL1 |
| 9. CTL2 | 10. ROM | | |

PROVISIONAL

On the back of the computer at the left hand side there is a large slot with a plastic cover, this is for QL. **ZX Spectrum ROM Cartridges are not compatible with the QL.**

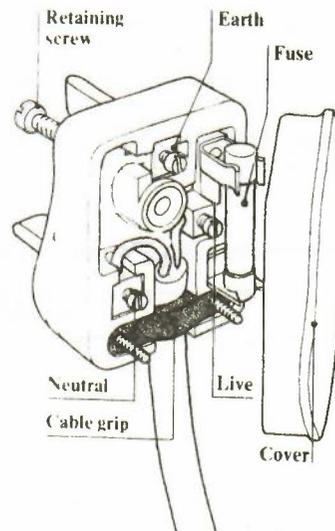
The back of the computer also contains two sockets for attaching two joysticks, two RS-232-C serial ports, power, monitor, TV connections and two connectors for the QLAN Local area network. The QLAN is compatible with the Spectrum Network and can be used to transfer data between the two machines.

To make the computer operational various connections have to be made:

The Power Supply

You will notice that the power supply has two leads. one is fitted with a flat three pin connector and which will later be connected to the QL (we will explain how), The other end must have a mains plug fitted.

Cut off a piece of the covering plastic, about 2cm long at the end of the lead. You will find two plastic covered wires, one blue and the other brown. Then cut off a piece of the covering plastic about 5mm long at the end of each of the wires. Connect the exposed end of the brown covered wire to the terminal marked L (live) and the exposed end of the blue wire to the terminal marked N (neutral). There is no wire to be connected to the terminal marked E (earth).



The Computer

Plug the main plug into a suitable mains socket and switch on. Check that there are no cartridges in the microdrive slots and plug the three pin flat connect from the power supply into the socket on the back of the computer marked POWER. The QL has no on/off switch but the computer can effectively be switched on and off by unplugging this connector. After some time the area of the case above the Microdrives may become warm, this is perfectly normal.

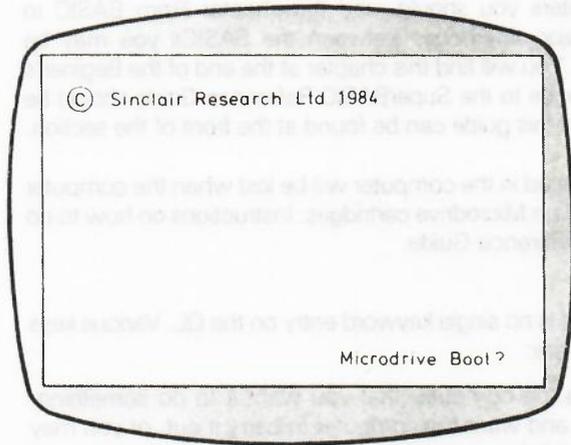
The Television

The QL is now working but before it can be used you must be able to communicate with it. A television is the simplest method of communicating with your QL. The television must be UHF. This means that the television must be able to receive BBC2. If the television is colour then the QL will produce a colour picture. If the television is in black and white then only a black and white picture will be reproduced, but the QL will perform fully in every other respect.

Find the UHF socket on the television set (if there is no socket marked but just an aerial socket use that one). If there is an internal television aerial plugged into the socket, disconnect it. Now take the aerial lead (this is the one with different sockets at either end of the lead). Push one end of the aerial lead into the television UHF socket. The right end can be found by comparing the lead with the original television aerial cable. Plug the other end of the lead into the socket marked on the back of the QL.

Plug in the television and switch on. Turn the volume down on the television. You now need to tune in the television. Choose an unused channel and adjust the tuning control until you see:

PROVISIONAL



If you fail to obtain a picture check first that your television is in working order and has been switched on. Try to obtain the normal broadcast stations. If you can receive the station, but cannot obtain the Copyright picture, try the computer with another television set.

If you have a monitor and wish to use it instead of a television set you will have to attach a suitably wired socket to the end of the monitor cable. A cable is available from Sinclair Research but will require wiring up to suit your particular type of monitor. A monochrome monitor can be connected using a 3-way or 8-way DIN plug but an RGB monitor requires an 8-way DIN plug; both fit into the socket on the back of the QL marked RGB. Information for wiring up the monitor lead can be found in the QL Concept Reference Section under the heading "monitor".

The question message "microdrive boot?" which appears on the screen is asking if you want it to continue to load more programmes from a Microdrive Cartridge to be included in the SuperBasic system - this can be ignored for now. Type "space" and the computer will continue to start up and will display its 'cursor' - a flashing coloured square.

When the cursor is visible, the computer is ready to accept new commands (or data). When the cursor is not visible, the computer is busy running a program.

If the machine ever fails to respond correctly or you want to force the program to stop, then the keys:



should be pressed together and the system will return the cursor to the screen. This sequence of keys is used to prevent accidental use. Should this fail to work, there is a reset button on the right-hand side of the computer which should be pressed.

In future when you start up your QL most of this procedure will not be necessary. You will notice when first switching on or after resetting a coloured pattern appears on the screen for a few moments. This is perfectly normal and is only the computer carrying out a self-test before going on to display the Copyright message.

The QL can be tilted to make typing easier. Three plastic feet are supplied; these can be inserted into the holes in the rubber feet on the bottom of the computer and fixed by moving from side to side until they are in position.

Getting Started

If you are new to computing you should now start working through the QL Beginner's Guide. If you are familiar with other computers you should read the chapter From BASIC to SuperBASIC which describes the major differences between the BASICs you may be familiar with and the BASIC on the QL. You will find this chapter at the end of the Beginner's Guide. If you are an expert then reference to the SuperBASIC Reference Guide should be sufficient. A note on the organisation of this guide can be found at the front of the section.

Please remember that any programs stored in the computer will be lost when the computer is switched off. Programs can be saved on Microdrive cartridges. Instructions on how to do this can be found on page of the Reference Guide.

Use of the Keyboard

Unlike previous Sinclair computers there is no single keyword entry on the QL. Various keys however have special meanings, they are:



The **ENTER** key is used to indicate to the computer that you want it to do something. Maybe you have typed in a command and want the computer to carry it out, or you may have typed in some data and want to tell the computer that you have finished and that it can carry on with its program.



The keyboard has two shift keys: Pressing shift and an alphabetic key will generate the upper case character. In the case of the other keys, the character engraved on the top part of the key will be generated, e.g. pressing SHIFT 5 will give %.



The CAPSLOCK key will force the keyboard to always generate upper case characters on alphabetic keys, but will continue to generate lowercase characters on all other keys unless the SHIFT key is pressed. CAPSLOCK is switched off by pressing the key again.



CTRL and ALT are used to control the computer, usually their use is combined with an alphabetic key to indicate a specific command.

Delete

Unlike most computer keyboards the QL does not have a DELETE key. The CTRL key together with the (back arrow) key are pressed together to delete the previous character.

The Cursor

As you type characters the cursor will move along the line showing where the next character will be displayed. When the cursor is visible and flashing the QL is ready to accept commands or data, if the computer is busy then the cursor is not visible.

PROVISIONAL

sinclair

QL

Beginner's Guide

The Beginner's Guide introduces the complete beginner to the concepts and techniques required to write complex computer programs. A section is included to illustrate the special features of SuperBASIC. The final section in the Beginner's Guide may be used by programmers familiar with other dialects of BASIC to convert to SuperBASIC.

The Beginner's Guide will be available shortly.

sinclair

QL

Keywords

The Keyword Reference Guide lists all SuperBASIC keywords in alphabetical order. A brief explanation of the keywords function is given followed by loose definition of the syntax and examples of usage. An explanation of the syntax definition is given in the Concept Reference Guide under "syntax definition". Each keyword entry indicates to which, if any, group of operations it relates, ie **DRAW** is a graphics operation and further information can be obtained from the graphics section of the Concept Reference Guide.

Sometimes it is necessary to deal with more than one keyword at a time, ie **IF, ELSE, THEN, END, IF**, these are all listed under **IF**.

An index is provided which attempts to cover all possible ways you might describe a SuperBASIC keyword. For example the clear screen command, **CLS**, is also listed under "clear screen" and "screen clear".

PROVISIONAL

ABS

math functions

Will return the absolute value of the argument. It will return the argument if the argument is positive and will return minus the argument if the argument is negative.

syntax: **ABS**(*numeric-expression*)

example: i. **ABS**(0.5)
 ii. **ABS**(-value)

AUTO

AUTO allows line numbers to be generated automatically when entering programs directly into the computer. Normally programs will be entered via the screen editor. Pressing escape will terminate automatic line numbering.

syntax: **AUTO**

example: i. **AUTO**

PROVISIONAL

ATAN
ACOT

ATAN and ACOT will compute the arctangent and arccotangent respectively. There is no limit to the size of the argument other than the maximum number the machine can store.

syntax: ATAN(*numeric__expression*)
 ACOT(*numeric__expression*)

example: i. ATAN(1)
 ii. ACOT(36574)

BAUD sets the baud rate for communication via both serial channels. The speed of the channels cannot be set independently.

syntax: BAUD *numeric__expression*

The value of the numeric expression must be equal to one of the standard baud rates supported by the QL.

- example: i. BAUD 9600
- ii. BAUD print__speed

Supported baud rates are:

comment

- 75
- 300
- 600
- 1200
- 2400
- 4800
- 9600
- 19200 (transmit only)

BEEP

sound

syntax: *duration*: = *numeric__expression* range 0 to 32768
pitch: = *numeric__expression* range 0 to 255
wrap: = *numeric__expression* range 0 to 15
fuzzy: = *numeric__expression* range 0 to 8
rand: = *numeric__expression* range 0 to 8
grad__X: = *numeric__expression* range 0 to 32768
grad__Y: = *numeric__expression* range 0 to 8

BEEP | *duration* ,*pitch__1*
| , *pitch__2* ,*grad__x*, *grad__y*
| , *wrap*
| , *fuzzy*
| , *rand*

where

duration specifies the duration of the sound in units of 20 mS. A duration of 0 will run the sound forever or until terminated.

pitch specifies the pitch of the sound in

pitch__2 specifies an upper pitch level between which the sound will 'bounce'

grad__x
grad__y specify the rate at which the sound will bounce between the two specified pitches

wraps will force the sound to wrap around the specified number of times. If *wrap* is less than 15, then it will wrap forever.

fuzzy will add a random number to the pitch on every cycle of the sound resulting in a fuzzy sound.

PROVISIONAL

window

BLOCK

Fill a block of the specified size and shape, at the specified position relative to the current window in the specified colour. **BLOCK** has no effect if the specified block falls outside the current window. **BLOCK** uses the pixel coordinate system and will draw the block in the window assigned by the last **USE** channel statement.

syntax:

- x__origin*: = *numeric__expression*
- y__origin*: = *numeric__expression*
- x__size*: = *numeric__expression*
- y__size*: = *numeric__expression*

BLOCK *channel* *x__origin*, *y__origin*, *x__size*, *y__size*

example:

- i. **BLOCK** 10, 10, 5, 5, 6, 2
- ii. 10 PRINT "Bar Chart"
20 LET bottom = 80
30 LET x = 20
40 LET width = 16
50 FOR height = 11, 23, 34, 56, 67, 29, 5, 3
60 LET colour = 3.6*height : REMark pick a colour
70 **BLOCK** x, bottom-height, width, height, colour
80 **BLOCK** x+1, bottom-height + 1, width-2, height-1, colour
90 LET x = x + width-1
100 END FOR height

BORDER

Adds a border to the window attached to the default channel or the channel assigned in the last USE statement.

syntax: *border__spec*: = *numeric__expression* range 0 to *x__size*/2

BORDER [*channel*] *border* [, *colour*]

Width specifies the thickness of the top and bottom edges of the border. The sides are twice this width.

example:

- i. **BORDER 10, 255** black and white stipple border
- ii. **10 FOR i = 6 TO 0 STEP -1 : BORDER i+2,i**
20 BORDER 8

sets consecutive borders and then a transparent border protect the result

comment: For all subsequent operations except **BORDER** the window size is reduced to allow space for the border. If another **BORDER** command is used then the full size of the window is used; thus multiple **BORDER** commands have the effect of changing the size and colour of a single border. Multiple borders are not created unless specific action is taken.

The colour of the border may be specified in the standard SuperBASIC manner, ie, it may be a single solid colour or it may be a stipple (see concept colour).

PROVISIONAL

devices

CAT

CAT will obtain and display the catalogue of cartridge in the specified Microdrive.

syntax: CAT *expression*

The expression must specify a valid Microdrive device

- example:
- i. CAT MDV1
 - ii. CAT "MDV2"

PROVISION

CHR\$

CHR\$ is a function which will return the character whose value is specified as a parameter.
CHR\$ is the inverse of CODE.

syntax: CHR\$ (*numeric__expression*)

- example:
- i. CHR\$(27) ASCII escape character
 - ii. PRINT CHR\$(65) print A

PROVISIONAL

graphics

CIRCLE

Draws a circle or an ellipse at a specified angle on the screen at a specified position and size. CIRCLE uses the graphics coordinates system. The circle will be drawn in the default window or the window attached to the channel assigned in the last USE statement.

syntax:

x: = *numeric__expression*
y: = *numeric__expression*
radius: = *numeric__expression*
eccentricity: = *numeric__expression* range 0 to 1
angle: = *numeric expression* range 0 to 2 (*pi symbol*)

CIRCLE *x__position*, *y__position*, *radius*, | *eccentricity*, *angle* |

x horizontal offset from the graphics origin
y vertical offset from the graphics origin
radius radius of the circle
eccentricity the ratio between the major and minor axis of an ellipse. eccentricity of 1 is a circle 0; is a straight line.
angle the orientation of the major axis of the ellipse relative to the screen vertical. The angle must be specified in radians

If eccentricity and angle are not specified then CIRCLE will default to drawing a circle. **comment**

CLEAR

CLEAR will clear out the variable area.

syntax: CLEAR

example: i. CLEAR

comment: RUN will do an automatic CLEAR before starting the program

PROVISION

CLOSE

CLOSE will flush all buffers related to the specified channel and will then close it. Any window associated with the channel will be deactivated.

syntax: *channel*: = # *numeric__expression* range 0 to 16

CLOSE *channel*

No action will be taken if an attempt is made to close an unopened channel

example: i. CLOSE 4
 ii. CLOSE *input__channel*

CLS*screen*

Will clear the current window to the current **PAPER** colour, excluding the border if one has been specified. **CLS** will accept an optional parameter which will specify if only part of the current window must be cleared.

syntax: *part* := *numeric__expression*

where: 0 - whole screen (default if no parameter)
 1 - top excluding the cursor line
 2 - bottom excluding the cursor line
 3 - whole of the cursor line
 4 - right end of cursor line including the cursor position

CLS *part*

example: i. **CLS**
 ii. **CLS 3**

PROVISIONS

SuperBASIC

CODE

CODE is a function which returns the internal code used to represent the character which is supplied as a parameter. **CODE** is the inverse of **CHR\$**

syntax: **CODE** (*string__expression*)

example: i. **PRINT CODE(A)** prints 65

1895



PROVISIONAL

SuperBASIC

CONTINUE

CONTINUE allows a program which has been broken into to be continued.

syntax: CONTINUE

example: i. CONTINUE

COPY

COPY will copy data from a channel or device to another channel or device until and end of file marker is detected or a timeout occurs. If the destination specification is omitted (ie TO device) then the console device is assumed.

syntax: **COPY** *device* | **TO** *device* |

It must be possible to input from the source device and it must be possible to output to the destination device.

example:

- i. **COPY** MDV1__data__file copy to default window
- ii **COPY** NET3 **TO** MDV1__data copy data from network station to MDV__data.

PROVISIONAL

maths functions

COS

COS will compute the cosine of the specified argument. The argument must be in the range -60000 to +60000 and must be specified in radians.

syntax: `COS(numeric_expression)` range -60000 to 60000

example:

- i.
- ii. `COS(3.141592654/2)`

PROVISIONAL

COT

math functions

COT will compute the cotangent of the specified argument. The argument must be in the range -30000 to 30000 and must be specified in radians.

syntax: **COT**(*numeric__expression*) range -30000 to 30000

example: i. **COT**(3)

ii. **COT**(3.1415 92654/2)

PROVISIONAL

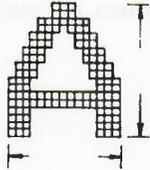
screen

CSIZE

Sets the character size. The standard size is 0,0 in 512 mode and 2,0 in 256 mode.

syntax: width: = numeric__expression range 0 to 3
 height: = numeric__expression range 0 to 1

CSIZE width, height



Width defines the horizontal size of the character space. Height defines the vertical size of the character space. The character size is adjusted to fill the space available.

width	size	height	size
0	6 pixels	0	10 pixels
1	8 pixels	1	20 pixels
2	12 pixels		
3	16 pixels		

example:

- i. CSIZE 3,0
- ii. CSIZE 3
- iii. 10 FOR height = 0 TO 1
 20 FOR width = 0 TO 3
 30 CSIZE width, height
 40 PRINT "Testing..."
 50 END FOR width
 60 END FOR height

CURSOR

CURSOR allows the screen cursor to be positioned anywhere in the current window. The positioning uses the pixel coordinate system. The origin is the relative to the current window origin.

syntax: **CURSOR** *channel* *x__position, y__position*

example: i. **CURSOR** 0,0
 ii. **CURSOR** 20,30

comment: Specifying a channel for **CURSOR** will affect the cursor position in window linked to the specified channel.

PROVISIONAL

SuperBASIC

DATA

DATA allows data to be defined within a program. The data can be read by a subsequent READ statement is ignored by SuperBASIC when it is encountered during normal processing.

syntax: DATA * | *expression*, | *

example: i. DATA "Monday", "Tuesday", "Wednesday"
ii. DATA 1, "JAN", 2, "FEB", 3, "MAR"

READ reads data contained in DATA statements and assigns it to variables. Data is first read from the first item in the first DATA statement encountered in the program. Subsequent READS read from subsequent items in the DATA statement and then from subsequent DATA statements. An error is reported if a READ is attempted for which there is no data. The RESTORE command may be used to set the line from which data will be read to any line in the program. READ

syntax: READ * | *identifier*, | *

example: i. 10 DIMension days\$(7,4)
20 FOR count = 1 TO 10
30 READ days\$(count)
40 PRINT DAYS\$(count)
50 END FOR count
60 DATA "MON", "TUE", "WED", "THUR", "FRI"
70 DATA "SAT", "SUN"

ii. 10 FOR count = 1 TO 12 : READ month\$(count)
20 DATA "January", "February", "March"
30 DATA "April", "May", "June"
40 DATA "July", "August", "September"
50 DATA "October", "November", "December"

PROVISIONAL

DATE\$

clock

DATE will return the date and time contained in the computers real time clock. The clock has battery back-up and normally will not require setting.

The format of the string returned by **DATE\$** is:

"dd-mm-yy hh:mm:ss"

where	dd is the day	1 to 28, 29, 30, 31
	mm is the month	1 to 12
	yy is the year	84, 85, etc
	hh is the hour	0 to 23
	mm are the minutes	0 to 59
	ss are the seconds	0 to 59

syntax: **DATE\$**

example: i. **PRINT DATE\$**
 ii. **PRINT DATE\$(10 TO)**

PROVISIONAL

functions and procedures

DEFine FuNction
END DEFine

Defines a SuperBASIC function. The function must be associated with a standard identifier and the formal parameters specified. Both formal parameters as well as those defined in **LOCAL** statement, have no effect on similarly named identifiers outside the function.

When a function is defined the type of the formal parameters need not be specified. SuperBASIC will determine a type when the function is activated.

An answer is returned from a function by appending an expression to the **RETURN** statement or by assigning a value to the name of the function.

The type of data returned by the function is indicated by the terminating character of the function identifier.

define: *parameters: = (expression * | , expression | *)*

long: This form allows multi-line functions to be defined

syntax: DEF FUNCTION | FN | *identifier* | *parameters* |
LOCAL *identifier* * | , *identifier* | *
statements
RETURN *expression* | |
END DEF

LOCAL and **RETURN** can be at any position within the procedure body.

example: 10 DEFine FuNction mean(a, b, c)
20 LOCAL answer
30 LET answer = (a + b + c)/3
40 RETURN answer
50 END DEFine

60 PRINT mean(1,2,3)

short: The short form of the **DEFine FuNction** command is included to allow compatibility between SuperBASIC and other forms of **BASIC**.

syntax: DEF FN *identifier* | *parameters* = *expression*

The expression is expressed in terms of the supplied parameters

example: DEFine FuNction round__up number = INT(number + 0.5)
PRINT round__up 3.45

DEFine PROCEDURE
END DEFine

Defines a SuperBASIC procedure. The sequence of statements between the **DEFine PROCEDURE** statement and the **END DEFine** statement constitutes the procedure. The procedure definition must also include a list of formal parameters which the procedure is to use. The formal parameters must be enclosed in brackets for the definition, but brackets are not necessary when the procedure is called. Variables may be defined to be **LOCAL** to the procedure. Formal parameters and locally defined variables have no effect on similarly defined variables outside the procedure.

The procedure is called by entering its name as the first item in a SuperBASIC statement. It is possible to regard a procedure definition as a command definition in SuperBASIC; many of the system commands are themselves defined as procedures.

syntax: *parameters* = *expression* * | , *expression* | *

```
DEFine PROCEDURE identifier(parameters)
  LOCAL identifier * | , identifier | *
  statements
  RETURN
END DEFine
```

LOCAL and **RETURN** can appear at any position within the procedure body.

example:

- i. 10 DEFine PROCEDURE start__screen
 20 WINDOW 0,0,100,100
 30 BORDER 4,255
 40 PRINT "Hello Everybody"
 50 END DEFine

- ii. 10 DEFine PROCEDURE slow__scroll(scroll__limit)
 20 LOCAL count
 30 FOR count = 1 TO scroll__limit
 40 SCROLL 1
 50 END FOR count
 60 END DEFine

comment: The parameters for a procedure must be enclosed in brackets for the definition. If the procedure has no parameters then it is not necessary to specify an empty set of brackets.

PROVISIONAL

devices

DELETE

DELETE will remove a file from the catalogue of the cartridges in the specified Microdrive.

syntax: **DELETE** *device*

The device specification must be a Microdrive device

- example:**
- i. **DELETE** MDV1__old__data
 - ii. **DELETE** MDV1__letter__file

DIMension

... arrays

Defines an array in to SuperBASIC. String, integer and floating point arrays can be defined. String arrays handle fixed length strings and the final dimension is taken to be the string length.

Array indices run from 0 up to the maximum index specified in the **DIMension** statement; thus **DIMension** will generate an array with one more element in each dimension than is actually specified. When an array is specified it is initialised to zero for a numeric array and zero length strings for a string array.

syntax: *dimension* := *numeric__expression*
 array__spec := *identifier*(*dimension* *| , *dimension* |*)

DIMension *array__ *| , array__ |**

- example:**
- i. DIMension string__array\$(10,10,50)
 - ii. DIMension matrix(100,100)

PROVISIONS

graphics

DRAW

DRAW will draw a line from the current graphics position to the specified position turning it through an optional angle. The point specification may be repeated to allow multiple lines to be drawn with a single call to **DRAW**.

DRAW uses the graphics coordinate system.

syntax: `point = x__position, y__position , angle`

`DRAW point * TO point *`

- example:
- i. `DRAW 10,50`
 - ii. `PLOT 0,0 : DRAW 0,0.5 TO 0.5,0.5`

EDIT

The **EDIT** command enters the QL screen editor and allows screen editing and line syntax checking to be performed.

The editor is entered by typing:

EDIT *line__number*

the optional line number will specify an initial line for the edit line, if it is omitted then the initial edit line will be the first line.

When a change is made to a line the line will be highlighted to indicate that it is not necessarily correct. When an attempt is made to move off the line, the complete line is checked for syntax errors. If no errors are detected then the edit cursor is moved. If syntax errors are detected then the edit cursor is not allowed to move off the line.

The edit cursor is moved to the line which requires editing. The screen will be scrolled up or down to keep the edit line visible. Changes can be made to a line either by inserting text or by changing text. In insert mode space is made for any new characters by scrolling the line sideways. The line will wrap around if necessary. In change mode characters in the edit line are replaced by characters typed in from the keyboard and the edit cursor is moved one character position to the right. Change mode and insert mode are switched between by typing **ALT**.

The edit cursor can be repositioned within the edit line with the cursor keys. Characters to the left and right of the edit cursor can be deleted by pressing the left and right cursor key in conjunction with the **CTRL** key.

Pressing **ENTER** will 'enter' a blank line into the edit window and allows new lines to be added.

Pressing **ESCAPE** during a line edit at any time will restore the original unedited line. Pressing **ESCAPE** at any other line will exit the editor.

PROVISIONAL

multitasking

EXEC

EXEC will load a sequence of programs and execute them in parallel. Communication 'pipes' will be automatically set up between each program so allowing each program to communicate with the others. For each program a series of devices can be specified which will be opened for the program before execution starts. The command interpreter will be restarted after the programs have started execution.

syntax:

- io__device*: = *device* use to specify I/O mapping for the program
- program*: = *device* used to specify a Microdrive file containing the program
- seperator*: = | !
| ,
- process*: = *program* *| , *io__device* |*
- EXEC *process* *| *seperator* *process* |*

The ! is used to separate individual processes in the command line and indicates that a communication pipe is to be set up between the processes.

example: i. EXEC MDV1__communications ! MDV1__current__job

multitasking

EXEC__N is the same as EXEC except that the system will wait for the last program to terminate before the command interpreter is restarted. EXEC__N

example: i. EXEC MDV1__accounts, data ! MDV1__printer__process

EXIT

EXIT will continue processing at the **END** of the named repeat structure.

syntax: EXIT *identifier*

example: i. 10 REMark start looping
 20 LET count = 0
 30 REPEAT loop
 40 LET count = count + 1
 50 PRINT count
 60 IF count = 20 THEN EXIT loop
 70 END REPEAT loop

 ii. 10 REPEAT outer__loop
 20 FOR n = 1 TO 1000
 30 REM program statements
 40 REM program statements
 50 IF RND = .5 THEN EXIT outer__loop
 60 END FOR n
 70 END REPEAT outer__loop

PROVISIONAL

math functions

FLASH

EXP

EXP will return the value of e raised to the power of the argument. The argument must be in the range -500 to 500 otherwise overflow will occur.

syntax: EXP(*numeric__expression*) range -500 to 500

- example:
- i. EXP(3)
 - ii. EXP(3.141592654)

FLASH

This turns the flash state on and off.

syntax: FLASH *numeric__expression* range 0 to 1

where: 0 will turn the flash off
1 will turn the flash on

example: i. 10 PRINT "A";
20 FLASH 1
30 PRINT "flashing";
40 FLASH 0
50 PRINT "word"

comment: FLASH operates in 256 mode only (low resolution)

PROVISIONAL

repetition

FOR END FOR

The FOR statement allows a group of SuperBASIC statements to be repeated a controlled number of times. The FOR statement can be used in both a long and a short form.

NEXT and **END FOR** can be used together within the same **FOR** loop to provide a loop epilogue. A loop epilogue is a group of SuperBASIC statements which will **NOT** be executed if a loop is exited via an **EXIT** statement. **EXIT** statement.

define: *for__item* = $\left\{ \begin{array}{l} \text{numeric_exp} \\ \text{numeric_exp TO numeric_exp} \\ \text{numeric_exp TO numeric_exp STEP numeric_exp} \end{array} \right.$
for__list = *for__item* * | , *for__item* | *

short: The **FOR** statement is followed on the same logical line by a sequence of SuperBASIC statements. The sequence of statements is then repeatedly executed under the control of the **FOR** statement. When the **FOR** statement is exhausted, processing continues on the next line. The **FOR** statement does not require its terminating **NEXT** or **END FOR**.

syntax: **FOR** *identifier* = *for__list* : *statement* * | : *statement* *

example:
i. **FOR** i = 1, 2, 3, 4, 2 TO 7 STEP 2 : **PRINT** i
ii. **FOR** element = first TO last : **LET** buffer(element) = 0

long: The **FOR** statement is the last statement on the line. Subsequent lines then contain a series of SuperBASIC statements terminated by an **END FOR** statement. The statements enclosed between the **FOR** statement and the **END FOR** are processed under the control of the **FOR** statement.

syntax: **FOR** *identifier* = *for__list*
statements
END FOR *identifier*

example:
i. 10 **FOR** value = data TO 1 STEP -1
20 **LET** factorial = factorial * value
30 **PRINT** value, factorial
40 **END FOR** value
ii. 10 **FOR** element = 1 TO length
20 **IF** data (element) = 0 **THEN** **EXIT** element
30 **LET** data (element) = root(data(element))
40 **NEXT** element
50 **PRINT** "Operation completed"
60 **END FOR** element

comment: For a simple **FOR** statement the **END FOR** and **NEXT** may be used interchangeably.

warning: Currently the for loop identifier must define a floating point control variable this restriction will be lifted.

FORMAT

FORMAT will format and make ready for use the cartridge contained in the specified Microdrive.

syntax: **FORMAT** *__device*

Device specifies the Microdrive to be used for formatting and the identifier part of the specification is used as the median or volume name for that cartridge

example: i. **FORMAT** MDV1__data__cartridge
 ii. **FORMAT** MDV2__wp__letters

warning: **FORMAT** can be used to reinitialise a used cartridge, however, all data contained on that cartridge will be lost.

PROVISIONAL

compatibility

GOTO

For compatibility with other **BASICs**, SuperBASIC supports the **GOTO** statement. **GOTO** will unconditionally transfer processing to the statement number specified. The statement number specification can be an expression.

syntax: **GOTO** *expression*

example: i. **GOTO** program__start
 ii. **GOTO** 9999

The control structures available in SuperBASIC make the **GOTO** statement redundant. **comment**

GOSUB

compatibility

For compatibility with other **BASICs**, SuperBASIC supports the **GOSUB** statement. **GOSUB** transfers processing to the specified line number; a **RETURN** statement will transfer processing back to the statement following **GOSUB**.

The line number specification can be an expression.

syntax: **GOSUB** *expression*

example i. **GOSUB** 100
 ii. **GOSUB** 4*select__variable

comment: The control structures available in SuperBASIC make the **GOSUB** statement redundant.

PROVISIONAL

The **IF** statement allows conditions to be tested and the outcome of that test to control subsequent program flow.

IF
THEN
ELSE
END IF

The **IF** statement can be used in three forms.

The **THEN** keyword is followed on the same logical line by a sequence of SuperBASIC statements. These statements are executed if the expression contained in the **IF** statement evaluates to be non zero.

short

syntax: `IF expression THEN statement *|: statement |*`

example:

- i. `IF a = 32 THEN PRINT "Limit reached"`
- ii. `IF test_data > maximum THEN LET maximum = test_data`
- iii. `IF a THEN PRINT "a is not zero"`

The **THEN** keyword is the last item on the logical line. A sequence of SuperBASIC statements is written following the **IF** statements. The sequence is terminated by the **END IF** statement. The sequence of SuperBASIC statements is executed if the expression contained in the **IF** statement evaluates to 1.

long 1

syntax: `IF expression THEN`
`statements`
`END IF`

example:

- i. `10 IF number > limit THEN`
`20 LET error_count = error_count + 1`
`30 PRINT "Number out of range"`
`40 END IF`

The **THEN** keyword is the last entry on the logical line. A sequence of SuperBASIC statements follows on subsequent lines, terminated by the **ELSE** keyword. If the expression contained in the **IF** statement evaluates to be non zero then this first sequence of SuperBASIC statements is processed. After the **ELSE** keyword a second sequence of SuperBASIC statements is entered, terminated by the **END IF** keyword. If the expression evaluated by the **IF** statement is not equal to 1 then this second sequence of SuperBASIC statements is processed.

long 2

syntax: `IF expression THEN`
`statements`
`ELSE`
`statements`
`END IF`

example:

- i. `10 IF number > limit THEN`
`20 PRINT "Range error"`
`30 ELSE`
`40 PRINT "Inside limit"`
`50 ENDIF`

In all three forms of the **IF** statement the **THEN** is optional. In the short form it may be replaced by a colon to distinguish the end of the **IF** and the start of the next statement. In the two long forms it can be removed completely.

comment

IF statements may be nested as deep as the user requires (subject to available memory). However, confusion may arise as to which **ELSE**, **END IF** etc matches which **IF**. SuperBASIC will match nested **ELSE** statements etc to the closest **IF** statement, for example:

nesting

```
10 IF a = b THEN
20   IF c = d THEN
30     PRINT "error"
40   ELSE
50     PRINT "no error"
60   END IF
70 END IF
```

The **ELSE** is matched to the second **IF**

INK

This sets the current ink colour, ie, the colour which output is written in.

syntax: **INK** *colour*

example: i. **INK** 5
 ii. **INK** 6,2

comment: The **INK** colour can be a stipple (see colour)

PROVISIONAL

repetition

REPeat
END REPeat

REPEAT allows general repeat loops to be constructed. REPEAT must be used with EXIT for maximum effect. REPEAT can be used in both long and short forms.

The REPEAT keyword and loop identifier are followed on same logical line by a colon and a sequence of SuperBASIC statements. EXIT will resume normal processing at the next logical line.

short

syntax: REPeat *identifier* : *statements*

example: REPeat wait : IF inkey\$ "" THEN EXIT wait

The REPEAT keyword and the loop identifier are the only statements on the logical line. Subsequent lines contain a series of SuperBASIC statements terminated by an END REPEAT statement.

long

The statements between the REPEAT and the END REPEAT are repeatedly processed by SuperBASIC.

syntax: REPeat *identifier*
statements
END REPeat *identifier*

example: i. 50 REPeat guess
60 INPUT "What is your guess?", guess
70 IF guess = number THEN
80 PRINT "You have guessed correctly"
90 EXIT guess
100 ELSE
120 PRINT "You have guessed incorrectly"
130 END IF
140 END REPeat guess

Normally at least one statement in REPeat loop will be an EXIT statement

comment

RND

RND generates a random number. Up to two parameters may be specified for RND. If no parameters are specified then RND returns a pseudo random floating point number in the range 0 to 1. If a single parameter is specified then RND returns an integer in the range 0 to the specified parameter. If two parameters are specified then RND returns an integer in the range specified by the two parameters.

syntax: RND | *numeric_expression* | | , *numeric_expression* |

- example:**
- i. PRINT RND
 - ii. PRINT RND 10,20
 - iii. PRINT RND 1,6
 - iv. PRINT RND 10

PROVISIONAL

documentation

REMark 239

Allows explanatory text to be inserted into a program. The remainder of the line is ignored by SuperBASIC.

syntax: REMark *text*

example: i. REM This is a comment in a program

RESTORE

RESTORES allows the data pointer, ie. the position from which subsequent **READS** will read their data. If **RESTORE** is followed by a parameter then the data pointer is set to that value. If no parameter is specified then the data pointer is reset to the start of the program.

syntax: *line* = integer__expression range 1 to 32768

RESTORE *line*

- example:**
- i. **RESTORE**
 - ii. **RESTORE 999**

functions and procedures
RETURN

RETURN is used to force a function or a procedure to terminate and resume processing at the statement after the procedure or function call. When used in a function the **RETURN** statement can also be used to return the functions value.

syntax: **RETURN** *expression*

- example:**
- i.


```

10 DEFine FuNction sinh x
20 IF ABS x > accuracy__limit THEN
30 RETURN EXP(x-LN 2)
40 ELSE
50 RETURN (exp(x) - exp(x))/2
60 END IF
70 END DEFine
      
```
 - ii.


```

10 DEFine PROCedure warning n
20 REM print a warning message
30 IF warning__flag THEN
40 PRINT "WARNING:";
50 SElect ON n
60 ON n = 1
76 PRINT "Microdrive almost full"
86 ON n = 2
98 PRINT "Data space almost full"
100 ON n = REMAINDER
110 PRINT "Program error"
120 END SElect
130 ELSE
140 RETURN
150 END IF
160 END DEFine
      
```

comment: It is not compulsory to have a **RETURN** in a procedure. If processing reaches the **END DEFine** of a procedure then the procedure will return automatically.

It is not compulsory to have a **RETURN** in a function. A function can be terminated by assigning a value to the name of the function, see **FuNction**

RANDOM

RANDOM allows the random number generator to be reseeded. If a parameter is specified the parameter is taken to be the new seed. If no parameter is specified then the generator is reseeded from internal information.

syntax: **RANDOM** *numeric__expression*

- example:**
- i. **RANDOM**
 - ii. **RANDOM 3.2235**

PROVISIONAL

graphics

PLOT

PLOT a point at the specified position relative to the graphics origin for the current window. Points are positioned by **PLOT** relative to the graphics origin.

syntax: $x_coordinate = numeric_expression$
 $y_coordinate = numeric_expression$
PLOT $x_coordinate, y_coordinate$

example: i. **PLOT** 256,128
 ii. **PLOT** x, x*x

POKE

POKE allows a memory location to be changed. An optional parameter can be specified to indicate if a byte or a word access is required. No optional parameter indicates that a byte access is required.

syntax: *address* = *numeric__expression*
 data = *numeric__expression*
 word = *numeric__expression* range 0 to 1

where: 0 indicates that **POKE** is to perform a byte access (8 bit)
 1 indicates that **POKE** is to perform a word access (16 bit)

POKE *address, data , word*

example: i. **POKE** 12235, 0
 ii. **POKE** 12345, 32768, 1

PROVISIONAL

multitasking

PAUSE

PAUSE will cause a program to wait a specified period of time. Delays are specified in units of 20ms.

syntax: PAUSE *numeric_expression*

- example:
- i. PAUSE 20
 - ii. PAUSE 100

PEEK

PEEK is a function which returns the contents of the specified memory location. AN optional parameter can be specified to indicate if a byte or a word access is required.

syntax: *address* := numeric__expression
 word := numeric__expression range 0 to 1

where: 0 indicates that **PEEK** is to perform a byte access (8 bit)
 1 indicates that **PEEK** is to perform a word access (16 bit)

PEEK (*address* [, *word*])

example: i. **PEEK 12245**
 ii. **PEEK 12,1**

PROVISIONAL

PAPER

PAPER set a new paper colour (the paper colour will be used by CLS, PAN, SCROLL, etc). The selected paper colour remains in effect until the next use of PAPER. *screen*

syntax: PAPER colour

example:

- i. PAPER 7
- ii. PAPER 7, 2
- iii. 10 REMark Show all colours and stipples
20 FOR i = 0 TO 7
30 FOR j = 0 TO 7
40 FOR k = 0 TO 3
50 PAPER i,j,k
60 SCROLL 6
70 END FOR k
80 END FOR j
90 END FOR i

PAPER will also set the STRIP colour

warning

PROVISIONAL

PAN

screen

PAN the entire current window the specified number of pixels to the left or the right. Paper is scrolled in to fill the clear area. An optional second parameter can be specified which will allow a part of the screen to be panned.

syntax: *part*: - *numeric expression*

where:

- 0 - whole screen (default in no parameter)
- 3 - whole of the cursor line
- 4 - right end of cursor line including the cursor position

PAN *numeric__expression* , *part*

If the expression evaluates to a **POSITIVE** value then the screen will be panned to the **LEFT**, otherwise it will be panned to the right. ((link positive and **LEFT** with highlight)

- example:**
- | | | |
|------|----------|---|
| i. | PAN 50 | pan left 50 pixels |
| ii. | PAN -100 | pan right 100 pixels |
| iii. | PAN 50,3 | pan the whole of the current cursor line
50 pixels to the left |

PROVISIONAL

files

OPEN (provisional)

Allows the user to open a channel for I/O.

syntax: *channel*: = # *numeric__expression* range 0 to 16

device: = see concept device

OPEN *channel, device*

example: i. # OPEN 5, f__name\$
 ii. # OPEN 15, "file__name"
 iii. # OPEN 7, MDV1__data__file
 iv. # OPEN 6, CON__10X20A20X20__32

Open channel 6 to the **CONsole** device creating a window size 10 x 20 pixels at position 20,20 with a 32 byte keyboard type ahead buffer.

Although the SuperBASIC syntax requires that a file name be supplied as a parameter to the **OPEN** statement SuperBASIC will automatically convert any unsuitable data to the correct form for the **OPEN** statement. This implies that if required the file name can be entered without quotes:

comment

OPEN # 5, "data__file"

OPEN # 5, data__file are equivalent

OVER

screen

OVER selects the type of over printing required. The selected type remains in effect until the next use of **OVER**.

syntax: **OVER** *numeric__expression* range -1 to 1

where:

0 implies print **INK** on **STRIP**

1 implies print in **INK** on transparent **STRIP**

-1 implies print in **INK** over previous contents of screen

example:

```
i.  OVER 1
ii. 10 REMark Shadow Writing
    20 PAPER 7 : INK 0 : OVER 1
    30 FOR i = 0 TO 10
    40  CURSOR i,i
    50  IF i= 10 THEN INK 2
    60  PRINT "Shadow"
    70 END FOR i
```

PROVISIONAL

repetition

NEXT

NEXT is used to control REPEAT and FOR constructions.

syntax: NEXT identifier

The identifier must match that of the loop which the NEXT is to control

example:

- i. 10 REMark this loop must repeat forever
10 REPEAT infinite_loop
30 PRINT "still looping"
40 NEXT infinite_loop
- ii. 10 FOR index = 1 TO limit
20 INPUT "data? ", array(index)
30 NEXT index
- iii. 10 REPEAT odd
20 LET number = RND(1,100)
30 IF number/2 = INT(number DIV 2) THEN NEXT odd
40 PRINT number; " is odd"
50 END REPEAT odd

If NEXT is used inside a REPEAT-END REPEAT construct it will force processing to continue at the statement following the matching REPEAT statement. in REPEAT

The NEXT statement can be used to repeat the FOR loop with the control variable set at its next value. If the list of values is used or if the range of the control variable has been exceeded then processing will continue at the statement following the NEXT; otherwise processing will continue at the statement after the FOR. in FOR

PROVISIONAL

compatibility

ON GOTO ON GOSUB

For compatibility with other **BASICs**, SuperBASIC supports the **ON GOTO** and **ON GOSUB** statements. These statements allow a variable to select from a list of possible line numbers a line to process in a **GOTO** or **GOSUB** statement.

syntax: **ON** *variable* **GOTO** *expression* * | , *expression* | *
 ON *variable* **GOSUB** *expression* * | , *expression* | *

example: i. **ON** x **GOTO** 10, 20, 30, 40
 ii. **ON** select__variable 1000, 2000, 3000, 4000

comment: **SElect** can be used to replace these two **BASIC** commands

PROVISIONAL

screen

MODE

MODE sets resolution of the screen definition and also the number of solid colours available. **MODE** will clear the entire screen and will reset all all windows and will close any channels which are using those windows for output.

syntax: **MODE** *numeric__expression*

where:

0, 256, 8 will select low resolution (8 colour mode)
1, 512, 4 will select high resolution (4 colour mode)

256 and 512 ar the number of pixels across the screen in mode

example: i. **MODE 256**
 ii. **MODE 4**
 iii. **MODE 0**

If the QL is being used on a television set then it will not be possible to see the full 512 pixels available in high resolution mode. **warning**

PROVISIONAL

SuperBASIC

NEW

NEW will clear out the old program and old variables.

syntax: NEW

example: NEW

PROVISIONAL

SuperBASIC

LRUN

LRUN will load a specified Microdrive file which must contain a SuperBASIC program. When loaded the program will start execution.

syntax: LRUN *device*

device must be a Microdrive device

example: i. LRUN MDV1__QUILL
 ii. LRUN MDV1__game

MERGE

Will load a file from the specified device. If the new file contains a line number which doesn't appear in the program then the line will be added. If the new file contains a replacement line for one that already exists then the line will be replaced. All other old program lines are left undisturbed.

syntax: **MERGE** *device*
device must be a Microdrive device

example: i. **MERGE MDV__1__overlay__program**
 ii. **MERGE MDV__1__new__data**

PROVISIONAL

functions and procedures

LOCAL

Allows a series of variables to be defined to be **LOCAL** to a procedure or a function.
LOCAL data is lost when the procedure or function terminates.

syntax: LOCAL *identifier* * |, *identifier* | *

example:

- i. LOCAL a, b, c
- ii. LOCAL temp_data

LN
LOG

LN will return the natural logarithm of the specified argument. **LOG** will compute the logarithm to base 10. There is no upper limit other than the maximum number the computer can store

syntax: **LOG**(*numeric__expression*) range greater than zero
 LOG(*numeric__expression*) range greater than zero

example: i. **LOG**(20)
 ii. **LN**(3.141592654)

LOAD

LOAD will load a SuperBASIC program from any QL device. The default device for **LOAD** is

MDV1__

syntax: **LOAD** *identifier*

example:

- i. **LOAD** "MDV1__EASEL"
- ii. **LOAD** ARCHIVE
- iii. **LOAD** NET3
- iv. **LOAD** SER1__E

comment The standard rules for SuperBASIC identifiers and SuperBASIC coercion still apply. It is not necessary to use the quote symbol in the **LOAD** statement. Since **MDV1__** is the default device it is not necessary to specify it. If the specified Microdrive file is not found on the specified Microdrive no attempt is made to search other drives on the system. To load from any other device the complete device specification must be given.

PROVISIONAL

strings

LEN

LEN will return the length of a string specified as a parameter.

syntax: LEN(*string_expression*)

example: i. LEN("LEN will find the length of this string")
 ii. LEN("output_string\$)

PROVISIONAL

LET

Starts a SuperBASIC assignment statement. The use of the **LET** keyword is optional. The assignment may be used for both string and numeric assignments. SuperBASIC will automatically convert unsuitable data types to a suitable for wherever possible.

syntax: **LET** *variable* = *expression*

example:

- i. **LET** a = 1 + 2
- ii. **LET** a\$ = "12345"
- iii. **LET** a\$ = 6789
- iv. **b\$** = test__data

PROVISIONAL

math functions

INT

INT will return the integer part of the argument. The argument must be in the range -32767 to 32767.

syntax: INT(*numeric_expression*) range -32767 to +32767

example: i. INT(34.657938)

PROVISIONAL

screen

INVERSE

INVERSE will cause all subsequent characters to be output in inverse, ie. PAPER on INK rather than INK on PAPER. INVERSE will remain in effect until the next use of INVERSE.

syntax: INVERSE *numeric...expression* range 0 to 1

where: 0 - turn inverse off
1 - turn inverse on

example: i. INVERSE 0
ii. INVERSE 1

PROVISIONAL

conditions

INKEY\$

INKEY\$ is a function which returns a single character input from a channel. If no channel is specified then the default channel or the channel specified in the last USE command is used.

syntax: INKEY\$ (channel)

- example:
- i. PRINT INKEY\$
 - ii. PRINT INKEY\$(#4)

INPUT

INPUT allows data to be entered into a SuperBASIC program directly from the QL keyboard by the user. SuperBASIC will wait until the specified amount of data has been input before continuing with the program. Each item of data must be terminated by the enter key.

INPUT will assume the default channel for input and output if no channel specification is specifically given. If a new default channel (channels) has been assigned with the **USE** command then this (these) channels will be used instead.

If input is required into a particular channel the cursor for the window connected to that channel will appear and start to flash.

syntax:

```
separator: = | !
            | ;
            | ;

channel: = numeric__expression

prompt: = | channel | expression__separator

INPUT | prompt | | channel | | variable | * | , variable |
```

example:

- i. `INPUT "Last guess" ! guess+0, "New guess? ! guess`
- ii. `INPUT "What is your guess?"; guess`
- iii. `10 INPUT "array size?"; length`
`20 DIMension array(length)`
`10 FOR element = 0 to limit-1`
`20 INPUT "data? ";array(element)`
`30 END FOR element`
`40 FOR element = 0 TO limit-1`
`50 PRINT array(element)`
`60 END FOR element`

comment: **INPUT** will output any expression as part of the prompt and will assume that any variable that doesn't form part of an expression requires data to be input, see example i.

PROVISIONAL

RUN

RUN allows a SuperBASIC program to be started. If a line number is specified in the RUN command then the program will be started at that point, otherwise the program will start at the lowest line number. RUN will reset the values of any defined variables. GOTO 1 may be used to start a program without clearing any variables.

syntax: RUN *numeric__expression*

example: i. RUN
 ii. RUN 10
 iii. RUN 2*20

Although RUN can be used within a program its normal use is to start program execution by typing it in as a direct command. comment

PROVISIONAL

SAVE

devices

SAVE will save a SuperBASIC program onto any QL device. The default device for SAVE is

MDV1__

syntax: SAVE *device*

example: i. SAVE MDV1__program__1
 ii. SAVE test__program
 iii. SAVE NET3
 iv. SAVE SER1

comment: The standard rules for SuperBASIC identifiers and SuperBASIC coercion still apply. It is not necessary to use the quote symbol in the **SAVE** statement. Also since MDV1__ is the default device it is not necessary to specify it, To **SAVE** on any other device the complete device specification must be given.

PROVISIONAL

devices

SBYTES

SBYTES allows areas of the QL memory to be saved on a QL device. The default device for SBYTES is

MDV1__

syntax: SBYTES *devices*

example:

- i. SAVE MDV1__screen__data
- ii. SAVE test__program
- iii. SAVE NET3
- iv. SAVE SER1

The standard rules for SuperBASIC identifiers and SuperBASIC coercion still apply. It is not necessary to use the quote symbol in the **SAVE** statement. Also since MDV1__ is the default device it is not necessary to specify it, To **SAVE** on any other device the complete device specification must be given.

comment

SIN

SIN will compute the sin of the specified argument. The argument must be in the range -60000 to 60000 and must be specified in radians.

syntax: **SIN**(*numeric_expression*) range -60000 to +60000

example: i. **SIN**(3)
 ii. **SIN**(3.141592654/2)

PROVISIONAL

graphics

SCALE

SCALE allows the scale factor used by the **GRAPHICS** procedures to be altered. A **SCALE** of 'x' implies that a vertical line of length 'x' will fill the vertical axis of the window in which the line is drawn. A scale of 100 is the default.

syntax: **SCALE** *numeric__expression*

- example:**
- i. **SCALE 0.5**
 - ii. **SCALE 10**
 - iii. **SCALE 100**

SCROLL

Scrolls the current window up or down. Paper is scrolled in at the top or the bottom to fill the clear space.

An optional second parameter can be specified to obtain a part screen scroll

syntax: *part* = *numeric__expression*

where: 0 - whole screen (default in no parameter)
 1 - top excluding the cursor line
 2 - bottom excluding the cursor line

SCROLL *numeric__expression* , *part*

If the expression evaluates to a positive value then the screen will be scrolled upwards.

- example:**
- i. **SCROLL 10** scroll up 10 pixels
 - ii. **SCROLL -70** scroll down 70 pixels
 - iii. **SCROLL 10, 2** scroll the bottom of the 10 pixels

PROVISIONAL

conditions

SElect allows various courses of action to be taken depending on the value of an variable.

select__variable: = *numeric__variable*

select__item: = | *expression*
| *expression TO expression*

select__list: = | *select__item* * | , *select__item* |*

Allows multiple actions to be selected depending on the value of a *select__variable*. The *select* variable is the last item on the logical line. A series of SuperBASIC statements follows, which is terminated by the next **ON** statement or by the **END SElect** statement. The **ON REMAINDER** statement allows a catch all which will respond if no other *select* conditions are satisfied.

syntax:

```
SElect ON select__variable
* | |ON select__variable |= select__list
  statements |*
| ON select__variable |= REMAINDER
  statements
END SElect
```

example:

```
i. 10 SElect ON error__number
    20   ON error__number = 1
    30   PRINT "Divide by zero"
    40   LET error__number = 0
    40   ON error__number = 2
    50   PRINT "File not found"
    60   LET error__number = 0
    70   ON error__number = 3 TO 5
    70   PRINT "Microdrive file not found"
    80   LET error__number = 0
    90   ON error__number = REMAINDER
    90   PRINT "Unkown error"
    110  error__recovery
    120 END SElect
```

If the *select* variable is used in the body of the **SElect** statement then it must match the *select* variable given in the *select* header.

The short form of the **SElect** statement allows simple single line selections to be made. A sequence of SuperBASIC statements follows on the same logical line as the **SElect** statement. If the condition defined in the *select* statement is satisfied then the sequence of SuperBASIC statements is processed.

syntax:

```
SElect select__variable = select__list : statement
* | : statement |*
```

example:

```
i.  SElect test__data = 1 TO 10 : PRINT "Answer within range"
ii. SElect answer = 0.00001 TO 0.00005 : PRINT "Accuracy OK"
iii. SElect a = 1 TO 10 : PRINT a!"in" : = REMAINDER : PRINT a!"out"
```

The short form of the **SElect** statement allows ranges to be tested more easily than with an **IF** statement. Compare example ii. above with the corresponding **IF** statement.

SElect
END SElect

define

long

short

comment

SQRT

SQRT will compute the square root of the specified argument. The argument must be greater than zero.

syntax: **SQRT**(*numeric__expression*) range greater than zero

example: i. **SQRT**(3)
 ii. **SQRT**(a+2 + b+2)

PROVISIONAL

STOP

STOP will terminate execution of a program and will return SuperBASIC to the command interpreter.

syntax: STOP

example: i. STOP

STRIP

This will set the current strip colour. The strip colour is the background colour which is used when **OVER 1** is selected. Setting **PAPER** will automatically set the strip colour to the new **PAPER** colour.

syntax: **STRIP** *colour*

- example:**
- i. **STRIP 7**
 - ii. **STRIP 0,4,2**

PROVISIONAL

TAB

TAB is a function which will return sufficient spaces to move the print position to the required column. If the print position is greater than the specified column then no action is taken. **TAB** must be used from within a **PRINT** statement. **TAB** takes account of the current character size.

syntax: *position* = numeric__expression

TAB *position*

example: i. **PRINT** **TAB**(30); "This starts at column 30"
 ii. **PRINT** "column 0 "; **TAB**(20); "column 20"

TAB is normally followed by the ; print separator, any other separator would attempt to space out the output and would nullify the effect of the **TAB**. **comment**

TAN

TAN will compute the tangent of the specified argument. The argument must be in the range -30000 to 30000 and must be specified in radians.

syntax: TAN(*numeric__expression*) range -30000 to 30000

- example:
- i. TAN(3)
 - ii. TAN(3.141592654/2)

PROVISIONAL

debugging

TRACE

The **TRACE** command turns on and off the SuperBASIC trace option. When trace is active a list of the line numbers and an indication of the statement within the line is output on the default screen.

Trace will accept an extra parameter which will output the trace information to a channel (which must have been previously opened) this feature allows program tracing to continue without interfering with the standard program window.

syntax: **TRACE** | *channel* | *numeric__expression* range 0 to 1

example:

i.	TRACE 0	trace off
ii.	TRACE 1	trace on
iii.	TRACE	trace__switch
iv.	TRACE #4 1	trace on and output trace information to channel 4

UNDER

Turns underline either on or off for subsequent output lines. Uses the current **INK** colour.

syntax: **UNDER** *numeric__expression* range 0 to 1

- example:**
- i. **UNDER** 1
 - ii. **UNDER** switch__value

PROVISIONAL

default channels

USE

USE allows the default channels for PRINT, to be defined to the system. Various groupings of function are assumed by the USE command:

group 1: PRINT output
 graphics output
 window functions (BORDER, WINDOW, CLS, PAN, SCROLL, etc)

group 2: INPUT prompt

group 3: INPUT input
 LIST output

syntax: channel = # numeric__expression range 0 to 16

USE channel |,channel ||,channel |

If one parameter is specified then groups 1, 2 and 3 are set to the specified channel. If two parameters are specified then group 1 is set to the first parameter and groups 2 and 3 are set to the second. If three parameters are specified then each group is set to the respective channel.

example: i. USE 0, 5
 ii. USE print__channel, prompt, Microdrive

PROVISIONAL

USR

USR allows a machine code program to be accessed.

syntax: *address* := *numeric__expression*

USR *address*

example: *i. USR 0*

PROVISIONAL

exceptions

WHEN

WINDOW

Information available shortly

... ..
... ..
... ..

WINDOW

... ..
... ..

... ..

WINDOW

WINDOW

Allows the user to create a window on the QL display screen. The window is created without any border.

syntax: *x__origin: = numeric__expression*
 y__origin: = numeric__expression
 x__size: = numeric__expression
 y__size: = numeric__expression

WINDOW *x__origin, y__origin, x__size, y__size*

x__origin and *y__origin* are the X and Y coordinates of the top left hand corner of the window. *x__size* and *y__size* are the width and depth of the window respectively.

Coordinates are specified using the pixel coordinate system.

example: **WINDOW 30, 40, 10, 10**

PROVISIONAL

sinclair

QL
Concepts

The Concept Reference Guide attempts to describe concepts relating to SuperBASIC and the QL hardware. Concepts are listed in alphabetical order of the most common term for that concept. An index is provided which attempts to anticipate any other terms which may be used.

The concept section of the Reference Guide places each concept in alphabetical order for the most common wording for that concept, i.e. string comparisons will be found under S. At the end of the section there is an index that tries to anticipate any other names that a particular concept may have, e.g. comparisons (string).

Array Literals

Array literals provide a short hand form of initialising an array. The contents of an array can be specified simply and the array initialised with a single assignment.

Array literals are enclosed in curly brackets the level of nesting indicates the number of dimension of the final array. Array literals can be used to define string arrays.

example: {0,1,2,3} a one dimensional array
 {{0,1}{2,3}} a 2 × 2 array
 {{{0,1}{2,3}}{4,5}{6,7}} a 2 × 2 × 2 array

 {"one", "two"} a 2 × n array

i. 10 DIMension data (2,2), answer\$(1,3)
20 LET data = {{1,10}{4,2}}
30 let answer\$ = {"yes", "no"}

PROVISIONAL

Array

Arrays must be DIMensioned before they are used when an array is dimensioned the value of each of its elements is set to zero (a zero length string for a string array). An array dimension runs from zero up to the specified value. There is no limit on the number of dimensions which may be defined other than the total memory capacity of the computer. Data in an array is stored such that the last index defined cycles round most rapidly:

example: consider array(4, 2)

stored as:

```
array(0, 0)
array(0, 1)
array(0, 2)
array(1, 0)
array(1, 1)
array(1, 2)
array(2, 0)
array(2, 1)
etc.
```

The element referred to by array(a, b, c) is equivalent to the element referred to by array(a)(b)(c)

PROVISIONAL

character set and keys

To be announced.

Coercion in SuperBASIC is the act of forcing a value to a type which will allow the requested operation to be performed, i.e. if SuperBASIC is requested to perform a numeric addition then for the operation to succeed the two operands must themselves be numerical. SuperBASIC will attempt to convert non numerical operands to floating point operands and then will continue.

Coercion between data types will be performed when necessary and when SuperBASIC can deduce the necessary types and can perform the conversion.

- example i. LET answer = "1" + "2" + "3" is valid SuperBASIC
- ii. LET answer = 3 + "2" is valid SuperBASIC
- iii. LET answer = "3.141592654" is valid SuperBASIC
- iv. LET answer = "PI" is NOT VALID
- v. LET answer\$ = 32 + "156" is valid

a = b + c no conversion is necessary before performing the addition, conversion is not necessary before assigning the result to a.

a% = b + c no conversion is necessary before performing the addition but the result is rounded to an integer value before assigning.

a\$ = b\$ + c\$ b\$ and c\$ are converted to floating point, if possible, before being added together. The result is converted to string before assigning.

Colour

In general on the QL colours may be specified at three levels. In its most general form a "colour specification" consists of a background or main colour (which will normally be referred to as 'colour'), a 'contrast' colour and a 'stipple' pattern. A colour specification can therefore have up to THREE arguments although the procedure call mechanism allows various parameters to be assumed.

Single *colour := composite colour*

The single argument specifies the three parts of the colour specification. The background colour is contained in the bottom three bits of the colour byte. The next three bits contain the exclusive or (XOR) of the main colour and the contrast colour. The top two bits indicate the stipple pattern.

This will be the general case, by specifying only the bottom three bits, (i.e. the required colour) no stipple will be requested and a single solid colour will be used for display.

Double *colour := background, contrast*

The "colour" is a stipple of the two specified colours. The default checkerboard stipple is assumed (stipple 4).

Triple *colour := background, contrast, stipple*

Background and contrast colours and stipple are each defined separately.

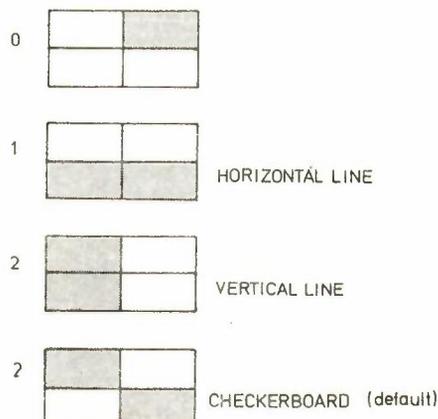
Colours The codes for colour selection depend on the screen mode in use:

Code	256 pixel mode	512 pixel mode
0	black	black
1	blue	black
2	red	red
3	magenta	red
4	green	green
5	cyan	green
6	yellow	white
7	white	white

Stipples "!!!!" implies that a four pixel square is filled with contrast colour.

- example
- i. PAPER 255 : CLS
 - ii. PAPER 2,4 : CLS

Warning Stipples should not be used on a television set fed by a UHF signal.



PROVISIONAL

Communications - RS-232-C

The QL has two serial ports (labelled SER1 and SER2) for connecting it to equipment which uses serial communications obeying EIA standard RS-232-C or a compatible standard.

Unfortunately the RS-232-C "standard" shows itself in a large number of different forms on different equipment, and it can be a tedious job, even for an expert, to connect together for the first time two pieces of supposedly standard RS-232-C equipment. This section attempts to cover most of the basic problems you will encounter.

The RS-232-C standard refers to two types of equipment:

- Data Terminal Equipment (DTE)**
- Data Communication Equipment (DCE)**

The main difference between these two types is that the Transmit data (TxD) and Receive Data (RxD) are switched round between them, i.e.

- The TxD line is output for Data Terminal Equipment**
- The RxD line is input for DTE and output for DCE**

Serial port 1 (SER1) on the QL is configured as DCE while serial port 2 (SER2) is configured as DTE. This means that it should be possible to connect at least one of the serial ports to a given device simply by using whichever port is wired the correct way. The pin-out for the serial ports is given below. A cable for connecting the QL to a standard 25-way "D" type connector is available from Sinclair (see the QL software and peripherals catalogue).

SER1		SER2	
pin	function	pin	function
1	GND	1	GND
2	TxD	2	TxD
3	RxD	3	RxD
4	DTR	4	DTR
5	CTS	5	CTS
6	+12V	6	+12V
TxD	Transmit Data		
RxD	Receive Data		
DTR	Data Terminal Ready		
CTS	Clear to send		

Once the equipment has been connected to the correct port the Baud Rate (transmission speed) must be set so that they are the same for both the QL and the connected equipment. The QL can be set to operate at

75	
300	
600	
1200	
2400	
4800	
9600	
19200 (transmit only)	baud

The QL baud rate is set by the BAUD command (see reference guide - BAUD).

The parity must be set to match the parity expected by the connected equipment. This can be set up when the serial channel is opened (see reference guide - OPEN).

It is not necessary to set the number of stop bits. The QL will always receive data with any number of stop bits and will always transmit at least two stop bits.

Communications on the QL is "full duplex", that is both transmit and receive can operate concurrently.

PROVISIONAL

It may be necessary to connect correctly the "handshake" signals. These signals allow the two QL and the connected equipment to monitor and control each others communication. The full RS-232 standard allows for nineteen signals, most of which are ignored by most equipment. The QL uses two control signals:

CTS - Clear To Send
DTR - Data Terminal Ready

CTS is a signal from DCE to DTE which indicates if data can be output on the TxD line.
DTR is a signal from DTE to DCE which indicates if data may be output on the RxD line.

Some pieces of equipment will function correctly without any use of handshake signals. The QL can ignore handshaking on transmission or not, depending on the parameters in the "OPEN" command (see reference guide - OPEN). However, the QL will not receive correctly without the use of CTS (on port 1) and DTR (on port 2).

If additional control signals are required by the equipment being connected to the QL, they must be wired up, etc.

PROVISIONAL

Data Types

These are whole numbers in the range -32767 to +32767. Variables are assumed to be integer if the variable name is suffixed with a percent (%).

Integer

Floating point numbers in the range -10^{615} to 10^{615} . The number of significant decimal digits is $\pm 10^{615}$ to 10^{615} . This is the default type in SuperBASIC.

Floating Point

A sequence of characters up to 32768 characters long (see character set).

String Literals

Devices

All I/O on the QL is to or from a logical file.

When a channel is opened certain basic information must be communicated to the system. This extra information is appended to the logical device name (cf. with appending the name of a disc file to the disc drive device name).

A file name has the same form as a standard identifier, however, certain characters have special reserved meanings when used in this context.

The general form of a file (device) name is:

device name *[_information]*

The underscore is interpreted as a separator between the logical device and any additional definitions which may have been made and between individual extra definitions.

Each logical device on the system requires its own special "extra information" although default parameters will be assumed in each case.

define *device* := *identifier*

where the form of the identifier is outlined below.

example **for CONsole device**

CON__wXhaxXy__k Console I/O
 w window width
 h window height
 x window X co-ordinate
 y window Y co-ordinate
 k keyboard type ahead buffer length (bytes)

example **CON__20x50a0,0__32e**

SCR__wXhaxXy Screen Output
 w window width
 h window height
 x window X co-ordinate
 y window Y co-ordinate
 k keyboard type ahead buffer length (bytes)

example **SCR__10x10a20x50**

SERnp Serial (RS-232-C)
 n port number
 p indicates parity
 e - even
 o - odd
 m - mark
 s - space

 h indicates handshaking
 i - ignore
 h - handshake

default 8 bit no parity with handshake

example **SER1__E**

NETnn Serial Network I/O
 nn node (station) number

example **NET32**

PROVISIONAL

Microdrive File Access

n Microdrive number

name Microdrive file name

MDVn__name

example MDV1__data__file

Eventually Microdrive file access will be possible by simply using the name of the file (or cartridge) and letting the system search for the required data. However, specifying the Microdrive number will speed up the operation.

Comment

Direct Command

SuperBASIC makes a distinction between a statement typed-in preceded by a line number and a statement typed-in without a line number. Without a line number the statement is a **DIRECT COMMAND** and is processed immediately by the command interpreter. For example : **RUN** is typed-in on the command line and is processed, the effect being that the program starts to run. If a statement is typed-in with a line number then the syntax of the line is checked and any detectable syntax errors marked. A correct line is entered into the SuperBASIC program and stored. These statements constitute a SuperBASIC **PROGRAM** and will only be executed when the program is started with the **RUN** or **GOTO** command.

PROVISIONAL

error handling

To be announced.

Expressions

SuperBASIC expressions can be string, numeric, logical or a mixture, unsuitable data types are automatically converted to a suitable form by the system wherever this is possible.

Define	<i>string__expression</i> :=	any expression that will return and answer its type string
	<i>numeric__expression</i> :=	any expression which will return an answer which is numeric (floating point or integer)
	<i>integer__expression</i> :=	any expression which will return an answer which is integer
	<i>expression</i> =	<i>expression operator expression</i> (<i>expression</i>) <i>atom</i>
	<i>atom</i> =	<i>identifier</i> <i>constant</i> <i>function__call</i>

PROVISIONAL

Functions and Procedures

SuperBASIC functions and procedures are defined with the **DEFine FUNction** and **DEFine PROCedure** statements. Functions and procedures are activated by typing the name of the function or procedure at the appropriate point in the program.

It is not necessary to always specify the complete set of parameters when a procedure or function is called. For example the SuperBASIC **PRINT** statement is implemented as a SuperBASIC procedure. This can accept a variable number of parameters of varying types.

Formal parameters (those specified in the function or procedure definition) are considered by SuperBASIC to be typeless, no assumptions about the usage of the data is made by SuperBASIC when the function or procedure is defined. The type of the formal parameters is defined when the procedure or function is activated. Various facilities have been built in to SuperBASIC to allow a procedure or function to determine the type and number of parameters it must process.



Graphics

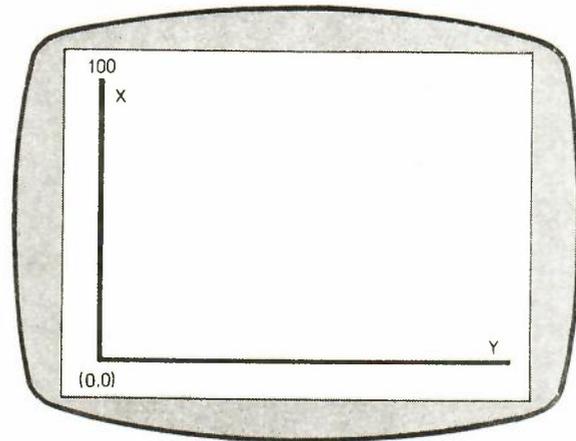
The graphics procedures ensure that whatever screen mode is in use, consistent figures are produced. It is therefore not possible to use a simple pixel count to indicate sizes of figures. Instead the graphics procedures use an arbitrary scale to specify sizes, this scale incorporates a user selectable scale factor which will enable the relative size of graphics output to be varied.

The scale factor is such that the full distance in the vertical direction has length 100 by default. The scale in the x direction is maintained so that x and y distances specifications draw a line with the same physical length. If the scale factor is increased then this will increase the maximum length of line that can be specified before the window size is exceeded.

The graphics procedures draw relative to the graphics origin which is in the bottom left hand corner of the current window. This is NOT the same as the pixel origin used to define the position of windows and blocks, etc. The graphics origin allows a standard cartesian co-ordinate system to be used.

The graphics procedures draw in the current ink colour where the term colour has its usual QL meaning.

It is important to realise that the QL screen will in both screen modes will have non square pixels and changing mode will have changed the shape of the pixels. Thus if the graphics procedures were simply pixel based they would draw different shapes in the two modes. For example, in one mode we would have a circle while the same figure in the other mode would be an ellipse.



PROVISIONAL

Identifier

A SuperBASIC identifier is a sequence of characters, numbers and underscores.

letter: = | a..Z
 | A..Z

Define

number: = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |

type: = | \$ (string)
 | % (integer)
 | letter, number (floating point)

example

- i. a
- ii. limit_1
- iii. current_guess
- iv. counter
- v. 13_November is not valid

An identifier must begin with a letter or an ampersand, followed by a sequence of letters, numbers and underscores. The final character indicates the type of the identifier. If an identifier starts with an **&** then it is assumed to be a system identifier and has a special case.

An identifier can be up to 255 characters long.

Identifiers are used in the SuperBASIC system to identify variables, procedures, functions, loops, etc.

The concept of an identifier is fundamental to SuperBASIC. It is used to identify variables, functions, procedures, loops, etc. The concept of a type being indicated by the form of the identifier is not always appropriate and should not be used. For example, PROCedures devices, etc.

Comment

NO meaning can be attributed to an identifier other than its ability to "identify" constructs to SuperBASIC. SuperBASIC cannot infer the intended use of an identifier from the identifier's name!

Warning

Joystick

The joystick marked CTL1 and CTL2 ports allow two joysticks with D-type connectors to be attached to the QL. It is necessary to use a special adapter to enable the joysticks to be inserted. The joysticks are arranged to generate specific key depressions when moved in a specific way.

	joystick 1 (CTL1)	joystick 2 (CTL2)
mode	key	key
up	cursor up	F4
down	cursor down	F2
left	cursor left	F1
right	cursor right	F3
fire	space	F5

Note Any program running on the QL which uses joysticks must be able to adopt the conventions listed below.

PROVISIONAL

Keyword

SuperBASIC keywords are the keywords which are defined in the Keyword Reference Guide. Keywords conform to the rules for standard identifiers. The case of the keyboard is insignificant. Keywords may be echoed in a mixture of upper and lower case, the upper case characters indicate the minimum which must be typed in for the computer to recognise the keyword. The keyword is always reproduced in full when the program is listed.

PROVISIONAL

Machine Code Programming

Full details of machine code programming facilities will be made available in the near future.

Memory Map

The QL contains a Motorola 68008 microprocessor, which can address 1 Megabyte, i.e. from 00000 to hex FFFFF. The use of addresses within this range are defined by Sinclair Research to be as follows:

Warning

Use of reserved areas in the memory map may cause incompatibility with future Sinclair products. Spurious output to addresses defined to be Peripheral I/O can cause unpredictable behaviour. It is recommended that these areas are NOT written to.

All I/O can be performed using either the relevant SuperBASIC commands or the QDOS operating system tapes.

The screen **RAM** is organised as a series of sixteen bit words starting at address Hex 20000 and progressing in the order of the raster scan, i.e. from right to left with each display line and then from the top to bottom of the picture. The bits are within each word are organised so that a pixel to the left is always more significant than a pixel to the right, (i.e. the pixel pattern on the screen looks the same as the binary pattern). However, the organisation of the colour information in the two screen modes is different:

where:

- G - green
- B - blue
- R - red
- F - flash

Setting the Flash bit toggles the flash state and freezes the background colour for the flash to the value given by R, G and B for that pixel. Flashing is always reset at the beginning of each display line.

Microdrives

Microdrives provide the main source of non volatile storage on the QL. Each Microdrive cartridge has a capacity of at least 100 Kb. Each cartridge can be write protected by removing the small plastic lug on the right hand side. Each cartridge must be formatted before use and can hold up to 255 sectors of 512 bytes per sector. QDOS keeps a catalogue of files stored on the cartridge. The number of files is limited to 50. QDOS also utilises spare RAM to provide buffers for as many sectors as possible to reduce wear on the tape and to improve performance.

Physically each Microdrive cartridge contains a 200'' loop of high quality video tape moving at 30 inches per second. The tape completes one circuit every 7 1/2 seconds. It is important that the exposed parts of the tape are protected by placing the cartridge in its protective cover whenever it is not in use. Also never switch the QL on or off with a Microdrive cartridge in place.

Up to six extra Microdrives can be added to the QL system.

PROVISIONAL

Multitasking

Full details of the multitasking facilities on the QL will be made available in the near future.

PROVISIONAL

monitor

A monochrome or colour monitor can be connected to the QL via the RGB socket on the back of the computer. Connection is via an 8 way DIN plug connected to the QL and a suitable plug at the other end.

pin function

1	RED	0-5V – (TTL compatible) active high
2	GREEN	0-15V – (TTL compatible) active high
3	BLUE	0-5V – (TTL compatible) active high
4	VSYNC	0-5V – (TTL compatible) active high – vertical sync
5	CSYNC	0-5V – (TTL compatible) active low – composite sync
6	VIDEO	0-5V – 1V pk-pk into 75 ohms – composite monochrome video
7		
8	GND	0-5V – ground

A monochrome monitor can be connected by using a screened lead with 3-pin or 8-pin DIN plug at the QL end. The connection at the monitor end will vary according to the monitor but is usually a phono plug. The monitor must have a 75Ω 1V pk-pk composite video non-inverting input (which is the industry standard, so most do). Both 3-pin DIN plugs and phono plugs are commonly available from audio shops.

An RGB (colour) monitor can be connected using a lead with an 8-pin DIN plug at the QL end. The plug at the monitor end will vary according to the monitor, as there is no industry standard, and will often be supplied with the monitor. A suitable cable with an 8-way DIN plug at one end is available from Sinclair Research.

PROVISIONAL

operators

=	equal	numerical string	- logical equal - type 2 comparison
==	equivalence	numerical string	- "almost equal" - type 3 comparison
+		numerical	- addition
-		numerical	- subtraction
/		numerical	- division
*		numerical	- multiplication
<	less than	numerical string	- less than - type 2 comparison
>	greater than	numerical string	- greater than - type 2 comparison
<=	less than or equal	numerical string	- less than or equal to - type 2 comparison
>=	greater than or equal	numerical string	- greater than or equal - type 2 comparison
<>	not equal	numerical string	- not equal to - not equal type 3 comparison
&	ampersand	string	- concatenation
&&		bitwise	- AND
		bitwise	- OR
^^		bitwise	- XOR
~~		bitwise	- NOT
OR		logical	- OR
AND		logical	- AND
XOR		logical	- XOR
NOT		logical	- NOT
MOD		numerical	- modulus
DIV		integer	- divide
^		floating	- raise to the power
(^)		integer	- raise to the power
-		unary minus	
+		unary plus (NOP)	

If the specified logical operation is true then a value of 1 will be returned, if the operation is false a value not equal to one will be returned.

precedence: highest unary plus and minus
string concatenation
exponentiation
multiply and divide (modulus and integer divide)
add and subtract
logical comparison
NOT
AND
lowest OR and XOR

PROVISIONAL

peripheral expansion

The expansion connector allows extra peripherals to be plugged into the QL. Further details to be announced shortly.

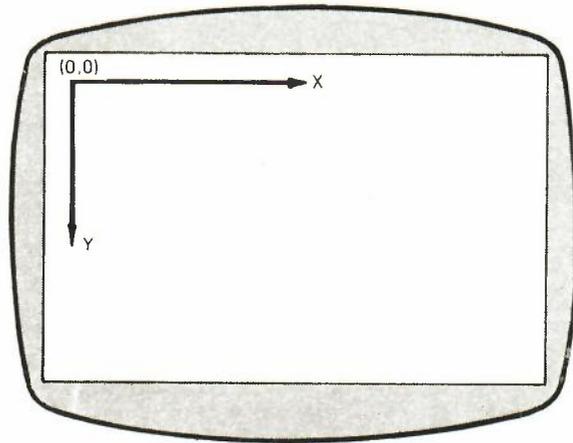


pixel coordinates

PROVISIONAL

The pixel coordinates system is used to define the positions and sizes of windows, blocks and cursor positions on the QL screen. The coordinate system has its origin in the top left hand corner of the default window (or screen) and always assumes that positions are specified as though the screen were in 512 mode (high resolution mode). The system will use the nearest pixel available for the particular mode set, this is so that positions are independent of the screen mode in use. Some commands are always relative to the default window origin, eg. **WINDOW**, while some are always relative to the current window origin, eg. **BLOCK**.

Characters are normally output to a window in a series of rows and columns the size of which is dependent on the character size in use. The **CURSOR** command allows the window cursor to be positioned down to the maximum resolution of the screen and therefore it becomes possible to specify a cursor position and hence a print position anywhere on the screen, this position need not be the same as the row-column grid initially assumed by the system.



PROVISIONAL

program

A SuperBASIC program consists of a sequence of SuperBASIC statements, where each statement is preceded by a line number. Line numbers are in the range of 1 to 32767.

syntax: *line... number statement *[:statement]**

example:

- i. 10 PRINT "This is a valid line number" : STOP
- ii. 10 REM a small program
20 FOR foreground = 0 TO 7
32 FOR contrast = 0 TO 7
43 FOR stipple = 0 TO 3
54 PAPER foreground, contrast, stipple
65 CURSOR 0,70
76 PRINT foreground | contrast | stipple
87 FOR step = 0 TO 2
98 SCROLL 2,1
100 SCROLL -2,1
110 END FOR step
120 END FOR stipple
130 END FOR contrast
140 END FOR foreground

PROVISIONAL

PROVISIONAL

qdos

QDOS is the QL Operating System. **QDOS** handles all process scheduling, all screen output (including the windowing capability) all Microdrive input and output, all keyboard input and all network and serial channel communication.

A full specification of **QDOS** will be available shortly.

PROVISIONAL

total repetition

Looping in SuperBASIC is controlled by two basic program constructs. Each construct must be identified to SuperBASIC:

REPEAT <i>identifier</i>	FOR <i>identifier = range</i>
<i>statements</i>	<i>statements</i>
END REPEAT <i>identifier</i>	END FOR <i>identifier</i>

These two constructs are used in conjunction with two other SuperBASIC statements:

NEXT <i>identifier</i>	EXIT <i>identifier</i>
-------------------------------	-------------------------------

NEXT and **EXIT** can be used in both constructs, in fact **EXIT** must be used in the **REPEAT** construct otherwise an infinite loop will result.

Processing a **NEXT** statement will either pass control to the statement following the appropriate **FOR** or **REPEAT** statement, or if a **FOR** range has been exhausted to the statement following the **NEXT**.

Processing an **EXIT** will pass control to the statement after the **END FOR** or **END REPEAT** selected by the identifier after the **EXIT** keyword. If an **EXIT** is used in a loop the loop must be terminated by **END FOR** or **END REPEAT**. **EXIT** can be used to exit through many levels of nested repeat structures.

Although a **REPEAT** loop can be terminated by a **NEXT** statement if it is the loop, the loop will be infinite since **EXIT** requires there to be an **END REPEAT** present.

warning

ROM cartridge slot

Allows a software to be loaded into the QL system via a Sinclair QL ROM cartridge. It is not possible to use **ZX Spectrum ROM Cartridges on the QL.**

warning: Never plug or unplug a ROM cartridge while the QL is powered up.

PROVISIONAL

screen

2 mode: The screen has 512 pixels across and is 256 pixels deep. Only the colours:

black
red
green
white

can be displayed

256 mode: The screen is 256 pixels across and 256 pixels deep. The full set of colours is available in this mode:

black/blue/red/magenta/green/cyan/yellow/white

comment: Surprisingly reducing the number of available colours while increasing the resolution of the screen allows more colours to be generated using the stipple patterns to mix various colours. Stipples should not be used with a domestic television fed by the UHF socket.

slicing

Under certain circumstances it is possible to reference more than one element in an array ie. SLICE THE ARRAY. The array slice can be thought of as defining a subarray or a series of subarrays to SuperBASIC. Each slice can define a contiguous sequence of elements belonging to a particular dimension of the original array. The term array in this context can include a numerical array a string array or a simple string.

consider: DIMension y(3,3,3)
 DIMension x(2,2,2)

then: LET x = y(0 TO 2, 0 TO 2, 0 TO 2)

 will set all corresponding elements in x equal to those in y.

In general for numeric array assignments the two arrays must be, if necessary after slicing) the same "shape".

It is not necessary to specify an index for the full number of dimensions of an array. If a dimension is omitted then slices are added which will select the full range of elements for that particular dimension, the slice (0 TO). SuperBASIC can only add slices to the end of the list of array indices

syntax: index: = | numeric__exp
 | numeric__exp TO numeric__exp
 | numeric__exp TO
 | TO numeric__expression
 array: = identifier {index *|, index|*|

An array slice can be used to specify a source or a destination sub array for an assignment statement.

example: i. PRINT data__array
 ii. PRINT letters\$(1 TO 15)

warning: Assigning data to a sliced string array or string variable may not have the desired effect. Assignments made in this way will not update the length of the string and so it is possible that the system will not see the assignment. The length of a string array or a string variable is only updated when an assignment is made to the whole string.

PROVISIONAL

sound

Sound on the QL is generated by the system IPC (8049) second processor. Sound is generated according to a series of preset sound types specified by parameters to the system. The sound capabilities of the QL can be expanded by adding expansion hardware to the expansion interface.

system guide

control

warning

string arrays String arrays and numeric arrays are essentially the same. However there are slight differences in treatment by SuperBASIC. The last dimension of the string defines the maximum length of the string. String lengths on either side of a string assignment need not be equal. If the sizes are not the same then either the right hand string is truncated to fit or the length of the left hand string is reduced to match. If an assignment is made to a sliced string then if necessary the 'hole' defined by the slice will be padded with spaces, (this is subject to the warning below).

It is not necessary to specify the final dimension of a string array. Not specifying the dimension selects the whole string while specifying a single element will pick out a single character and specifying a slice will define a sub string.

comment: Unlike most BASICS SuperBASIC does not treat string arrays as fixed length strings. If the data stored in a string array is less than the maximum size of the string array then the length of the string is reduced.

warning: Assigning data to a sliced string array or string variable may not have the desired effect. Assignments made in this way will not update the length of the string and so it is possible that the system will not recognise the assignment. The length of a string array or a string variable is only updated when an assignment is made to the whole string.

PROVISIONAL

start up

The QL starts up after switch on in a default state:

reset: At reset program execution is the first code to be executed by the QL after switch on after pressing the reset switch. It will perform a RAM test which will give a spurious pattern on the screen, which will later clear. The default window will then be drawn and the initial sign on message written.

If a suitable ROM cartridge is in the machine booting will continue under the control of the code in the ROM cartridge if no suitable ROM cartridge is found then the process continues.

The system asks if further booting from the Microdrive is required. If the answer NO (n or N) is typed then the system will enter SuperBASIC and display a flashing cursor.

If any other key is typed then each Microdrive in turn will be searched for a file.

PROCS

if this file is found then the procedures contained in it will be loaded and link into the SuperBASIC system. The system then searches for a file

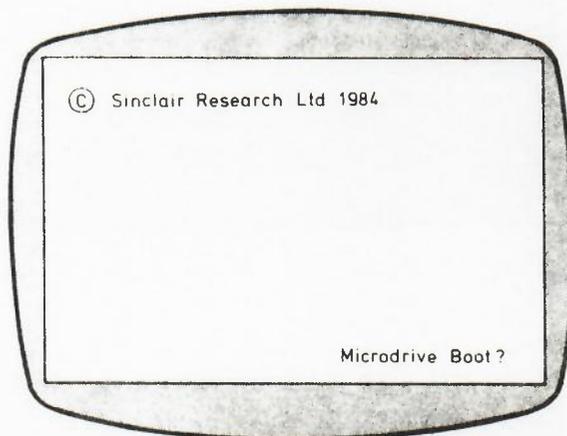
RUN

Which if found will be loaded and entered. If the file is not found then the extended SuperBASIC system entered. RUN must be an executable file.

screen: The default screen is in low resolution mode with a window.

keyboard: The default keyboard type ahead buffer is 32 bytes long. No caps lock.

warning: It is important NOT to power up the QL with a Microdrive cartridge in position. If booting off the Microdrive is required then the Microdrive must be inserted between switching on and answering the boot question. When a Microdrive cartridge is inserted into a drive the exposed tape should be on the left and side and at the far end.



statement

A SuperBASIC statement is an instruction to the QL to perform a specific operation, for example:

```
LET a = 2
```

will assign the value 2 to the variable identified by a

More than one statement can be written on a single line by separating the individual statements from each other by a colon (:), for example:

```
LET a = a + 2: PRINT a
```

will add 2 to the value identified by variable a and will store the result in back in a. The answer will then be printed out.

If a line is not preceded by a line number then the line is a direct command and SuperBASIC processes the statement immediately. If the statement is preceded by a line number then the statement becomes part of a SuperBASIC program.

warning: Certain SuperBASIC statements can have an effect on the other statements over the rest of the logical line in which they appear ie, IF, FOR, REPEAT, REM, etc. It is meaningless to use certain SuperBASIC statements as direct commands.



PROVISIONAL

strings

A string is a sequence of characters which may be up to 32000 bytes long. String data is identified to the system by enclosing the required string in quotes. String data may be stored in a string variable which is identified by suffixing a standard variable identifier with \$.

SuperBASIC will ensure that correct data types are used in each operation, if necessary SuperBASIC will coerce data to the correct form. This feature allows SuperBASIC to accept string data as input to numeric operators.

In general type coercion will be performed by SuperBASIC as long as the required type can be inferred and the conversion is possible. (see concept coercion)

- example:
- i. "this is an example of string data"
 - ii. LET alphabet\$ = "abcdefghijklmnopqrstuvwxy"
 - iii. LET answer = "1" + "2" + "3"
 - iv. LET a\$ = "1" + 12345

string comparison

order: (UK machines)

space

!'3457'()*+,-./:;=\$?@ ^_ ' : tilde

digits or numbers in numerical order

AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz

The relationship of one string to another may be:

equal: All characters or numbers are the same or equivalent

lesser: The first part of the string, which is different from the corresponding character in the second string, is before it in the defined order.

greater: The first part of the first string which is different from the corresponding character in the second string, is after it in the defined order.

types of comparison type 0 case dependent - character by character comparison

type 1 case independent - character by character

type 2 case independent - numbers are sorted in numerical order

type 3 case independent - numbers are sorted in numerical order

use: type 0 not normally used by the SuperBASIC system.

type 1 File and variable comparison use

type 2 SuperBASIC , =, =, = and use

type 3 SuperBASIC == (equivalence) uses

PROVISIONAL

system variables

System variables are used by SuperBASIC to "communicate" with a user program.

define: `system__variable = &identifier`

System variables cannot be type string.

A list of system variables will be released shortly.

list

PROVISIONAL

syntax definitions SuperBASIC syntax is defined using a non rigorous "meta language" type notation. Three types of symbols are used:

|| OR
Enclosed item(s) are optional
* * Enclosed items are repeated

ie.

| A | B | A or B
A A is optional
* A * A is repeated

Consider a superBASIC identifier:

A sequence of numbers, digits, underscores, starting with a letter and finishing with and optional % or \$

letter: = | A to Z
| a to z
a letter is
one of:
 ABCDEFGHIJKLMNPOQRSTUVWXYZ
 or abcdefghijklmnopqrstuvwxyz

digit: = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ||
a digit is 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9

underscore: = _
an underscore is _

identifier: = *letter* * *letter* | *digit* | *underscore* * | \$ | %

must start with a letter

a sequence of letters
digits and underscores
ie. repeat something
which is optional

An optional \$ or an optional
% but not both

PROVISIONAL

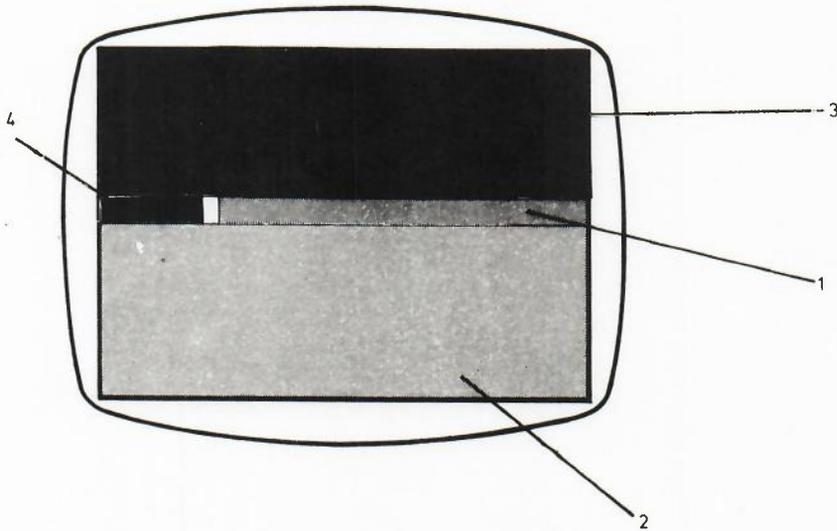
windows

Windows are areas of the screen which behave, in most respects, as though each individual window was a screen in it's own right, ie. the window will scroll when it has become filled by text, it can be cleared with the **CLS** command.

Windows can be specified and linked to channel when the channel is opened (see **OPEN**). The current window shape can be changed with the **WINDOW** command and a border can be added to a window with the **BORDER** command. Output can be directed to a window by printing to the relevant channel. Input can be directed to have come from a particular window by inputing from the relevant channel.

Certain commands (**CLS**, **PAN** etc) will accept an optional parameter to define part of the current window for their operation. This parameter can be defined as:

parts



sinclair

QL

QL Quill

Alabama

OL
OL Drill

C

C

C

C

PROVISIONAL

CHAPTER 1 ABOUT QUILL

QUILL is an extremely sophisticated word processor. It has been designed to give you the maximum in power and flexibility, yet is still easy to learn and to use. As you will see later, you will always be kept informed as to what you can do next and how to do it.

You can use a word processor in any circumstance where you would otherwise consider making use of a typewriter. The two machines are very similar in function, although the word processor has many advantages that are not matched by a conventional typewriter. Perhaps the most obvious difference is the ease with which mistakes can be corrected. Since the text is not printed immediately as you type it in, you can make as many corrections as you wish. You need only ask for the text to be printed when you are sure that it is exactly what you want and can therefore be assured of perfect results every time.

As you will see, by working through this manual, there are a number of other differences, all of which add to the convenience of use. For example, when using a typewriter, it is necessary to press the carriage return key at the end of every line. In QUILL, this function is performed automatically. Whenever the printed text reaches the end of a line, a new line is started; the only time that you need to press ENTER is when you want to start a new paragraph. Whenever a new line is started you will notice that the spacing of the text in the last line will be adjusted so that the left and right margins are aligned throughout the text. This process, which is known as justification, gives a highly professional appearance to the final result, without any effort on your part. Like most of the features of QUILL, this facility may be modified, depending on your requirements.

CHAPTER 2 GETTING STARTED

2.1 LOADING QUILL

When you switch on the computer it will only respond to commands in SuperBASIC. You will have to load QUILL from its Microdrive cartridge. You will normally do so by inserting the QUILL cartridge in drive 1 (the left hand drive) and then typing:

LGO MDV1_QUILL ENTER

After a few seconds the screen will show the message:

QUILL - Copyright Psion Ltd 1983
Press any key to start

You should then press any key on the keyboard to start QUILL.

2.2 GENERAL APPEARANCE

When you have loaded QUILL, you will see that the screen appears as shown in Figure 2.1. This is known as the *main* display. The screen is divided into three main sections, known as the *display area*, the *status area* and the *control area*.

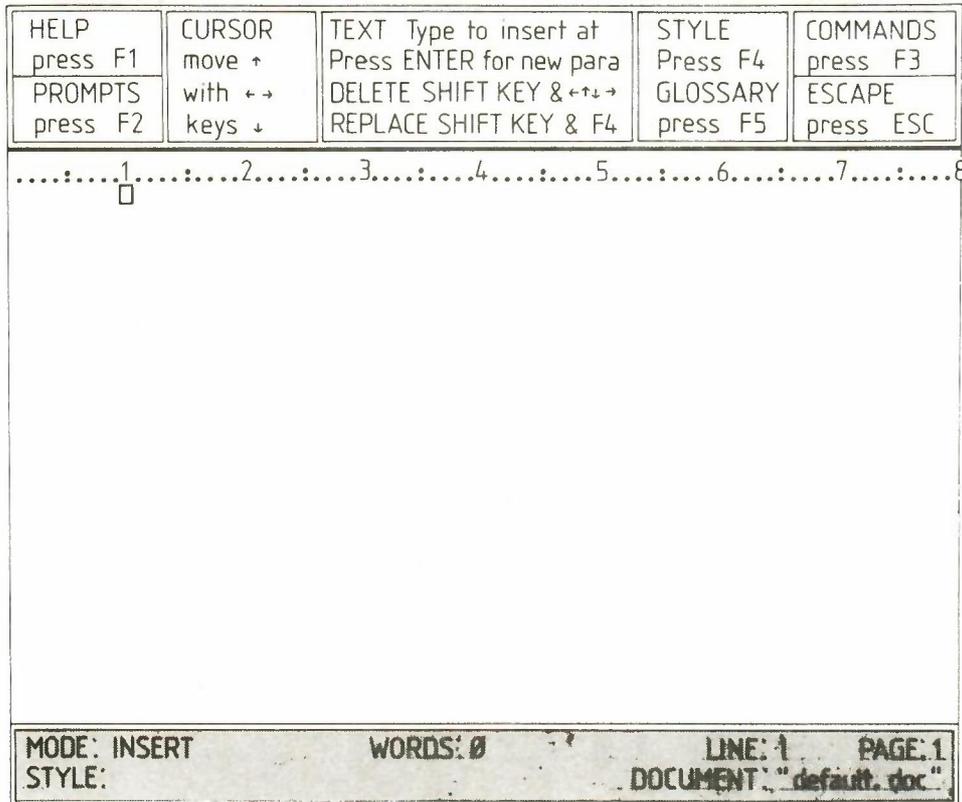


Figure 2.1: The Main Display.

2.2.1 The Display Area

The largest area, across the centre of the screen, is reserved for the text of your document. Almost everything that you type at the keyboard will appear in this area. Across the top of the display area is the *ruler*.

This is a row of dots, marking each character space across the display. Every fifth space is marked with a colon (:), and every tenth space is numbered. You will find this useful for finding exactly how far across the page you are at any time. The ruler marking directly above the cursor (see Section 2.5) is highlighted to make its position even clearer.

2.2.2 The Status Area

The *status area*, which uses the bottom three lines of the display, shows information about your current document. It normally shows the name of the current document. When you have just loaded QUILL you will not have given a name to a document and QUILL shows the name "default.doc". QUILL will use this document name for any text that you type, until you give the document another name.

The status area also shows that QUILL is currently in *insert mode* and that there is no special *style*, ie that you are using a normal typeface. In addition, the status area shows the number of words in the current document, and the line and page number of the

PROVISIONAL

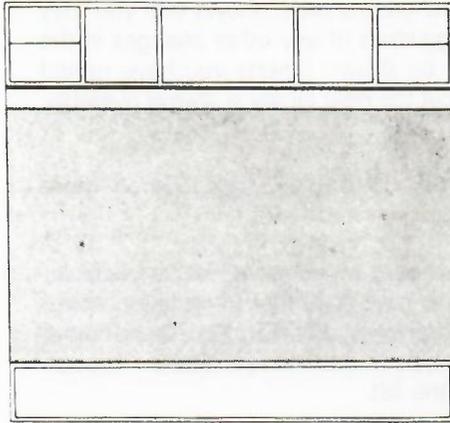


Figure 2.2: The Display Area.

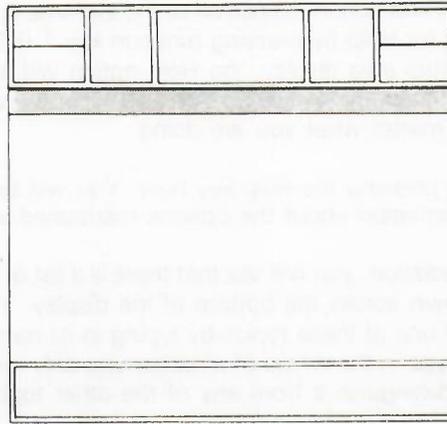


Figure 2.3: The Ruler.

position of the cursor. It initially shows that you are at line 1 of page 1 of a document which contains no words.

The status area is also used for showing any special text that is typed in during the use of commands. This is, for example, where the section of text being searched for will appear when you use the Search command (see Chapter 6).

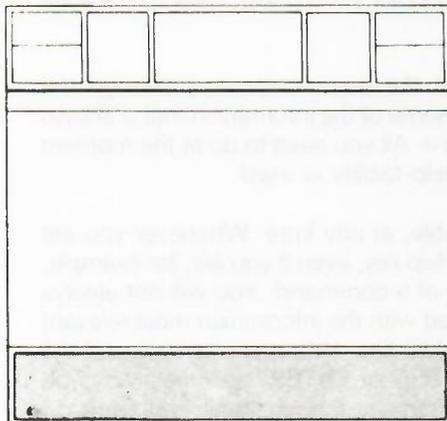


Figure 2.4: The Status Area.

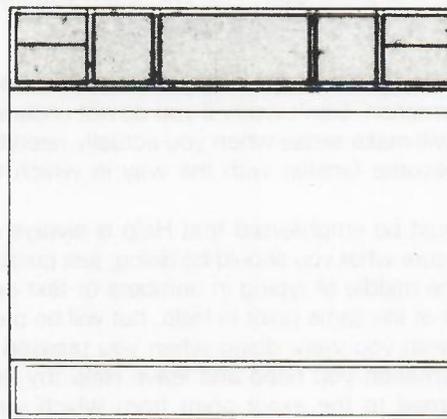


Figure 2.5: The Control Area.

The *control area*, at the top of the screen, is used to display the options available to you. The contents of this area will change each time that you select a different option, so that it is always relevant to what you are doing.

2.2.3 The Control Area

As we shall see later, you can choose not to display the control area so that you have more room for displaying the text. You will probably find this facility useful when you are more familiar with QUILL and know what options you have.

Immediately after loading QUILL the control area shows that you have eight main options which are, from left to right, to:

- obtain Help
- turn the prompts on or off
- move the cursor
- add or remove text
- change the style of text
- use or define a glossary entry
- use a command
- press ESC

2.3 HELP

The first option, displayed at the extreme left of the control area, shows that you may ask for Help by pressing function key 1 (F1). Regardless of any other changes in the control area display, the Help option will always be shown (unless you have turned off the prompts - see Section 2.4). This indicates that the Help facility is always available, no matter what you are doing.

Try pressing the Help key now. You will see that the display changes to show more information about the options mentioned above.

In addition, you will see that there is a list of topics about which more Help is available, shown across the bottom of the display. You may ask for further information about any one of these topics by typing in its name and pressing **ENTER**. You do not need to type in the whole of a name; you only need to type in the first few letters - enough to distinguish it from any of the other topics in the list.

When you press **ENTER** after typing in your choice of topic you will find that further, more detailed, information is shown about the topic you have selected and another list of sub-topics may then be shown. You can then select one of these sub-topics by typing in the first few letters of its name and pressing **ENTER**, as described before. You may continue this process until you are told that no further information is available.

At any stage you can go back to the previous screen by just pressing **ENTER** without typing any preceding text. Repeatedly pressing **ENTER** will therefore eventually take you back to the main display of your document, with the control and status areas. At this point you will have left the Help facility and will have been returned to exactly the same situation as before you pressed the Help key. A faster way to return from Help is to press **ESC**. This will return you from any point within Help, back to the state from which it was first called.

Try using the Help facility to examine some of the many pathways through the information. Don't worry if you do not understand some of the information that is shown - it will make sense when you actually need to use it. All you need to do at the moment is become familiar with the way in which the Help facility is used.

It must be emphasised that Help is always available, at any time. Whenever you are not sure what you should be doing, just press the Help key, even if you are, for example, in the middle of typing in numbers or text as part of a command. You will not always start at the same point in Help, but will be presented with the information most relevant to what you were doing when you pressed the Help key. When you have found the information you need and leave Help (by use of **ESC** or **ENTER**) you will always be returned to the exact point from which you started, as though there had been no interruption.

Use the Help key as often as you like - it is there to assist you and will usually be the quickest and simplest way of solving your problems.

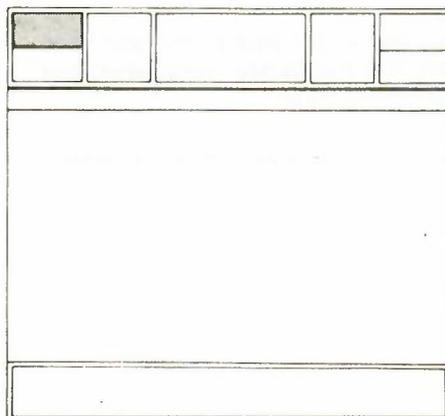


Figure 2.6: Help.

PROVISIONAL

In addition to showing your options, the control area highlights your choice and, when necessary, suggests what you should do. These aids to using QUILL are known as prompt messages or just *prompts*.

2.4 THE PROMPTS

You can switch off the display of the control area and the prompts it contains by pressing function key 2 (f2). When you do this the display will be redrawn without the control area, leaving more room for your document. The appearance of the screen is then as shown in Figure 2.8.

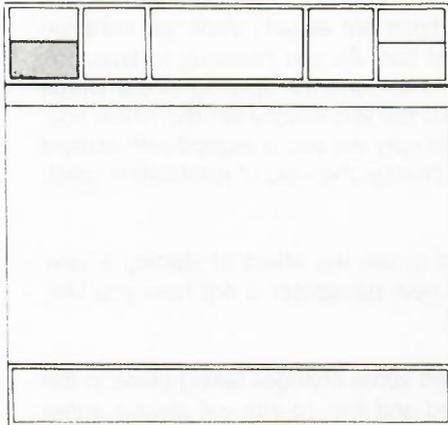


Figure 2.7: Prompts

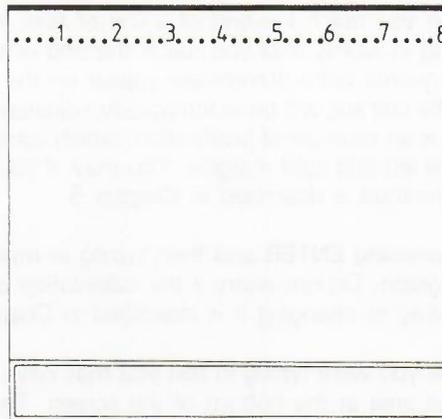


Figure 2.8: The Main Display Without Prompts.

You can bring back the control area and the prompts it includes by pressing F2 again. QUILL works in exactly the same way, whether the prompts in the control area are displayed or not - you are free to choose either option.

You will probably find it most useful to type your document with the prompts in the control area visible, and then turn them off to give you more room for looking at the finished result.

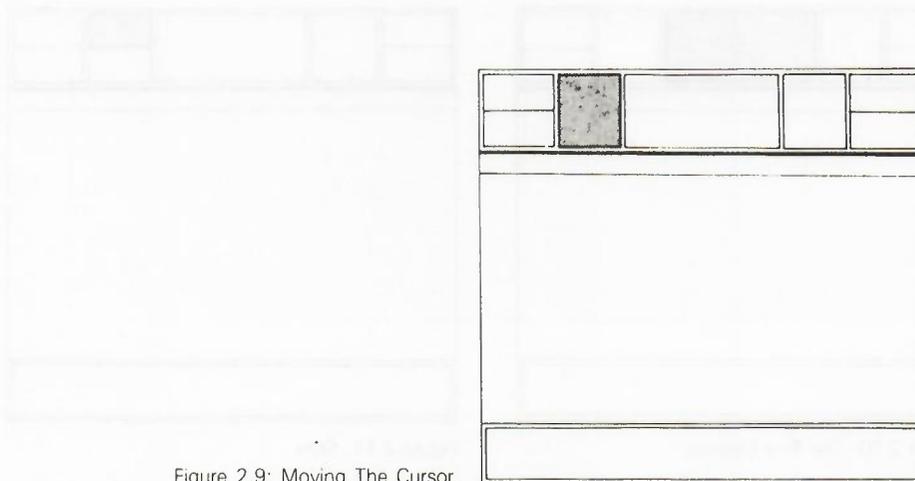


Figure 2.9: Moving The Cursor.

2.5 THE CURSOR

At the top left hand corner of the central text area you will see a small flashing rectangle. This is known as the *cursor* and marks the position where your typed input will be placed.

The second option in the control area shows that the cursor can be moved around the text area by use of the four cursor keys on the keyboard. When you are displaying text, each time that you press one of these keys, the cursor will move by one space in the direction indicated by the cursor. The cursor will not pass the end of the text so, with no text in your document, you will not be able to move the cursor from its original position.

2.6 TEXT So far we have managed to avoid putting anything into the text area of the screen. Since this is the main function of QUILL, we cannot delay it any longer.

The next option, shown at the centre of the control area, indicates the various ways in which you can change the text of the document. Simply typing at the keyboard will insert the text at the cursor position. Try typing at few words to see how this works.

The second line of this option shows that pressing **ENTER** is used to mark the start of a new paragraph. As mentioned in the introduction, you do not need to press **ENTER** when you reach the end of a line of text. If you have not already done so, continue typing in words until you reach the end of the first line. As you continue to type, the new words will automatically appear on the second line and the spacing of the words on the first line will be automatically adjusted so that the text exactly fills the whole line. This is an example of justification, which controls the way the text is aligned with respect to the left and right margins. You may, if you like, change the type of justification used; the method is described in Chapter 5.

Try pressing **ENTER** and then typing in more text to see the effect of starting a new paragraph. Do not worry if the indentation of the new paragraph is not how you like, the way of changing it is described in Chapter 7.

While you were typing in text you may have noticed some changes taking place in the status area at the bottom of the screen. The word and line counts will always agree with the contents of the document. The remainder of the status area will not have changed. In particular, the document will still be called "default__doc" since you have not yet given it any other name. A document will be named when it is saved on a Microdrive cartridge, as described in Chapter 4.

Now that you have some text in your document, you can try moving the cursor around the text area by use of all four cursor keys. When you have finished, move the cursor to the end of the text. See Chapter 3 for a description of how to use the other parts of this option.

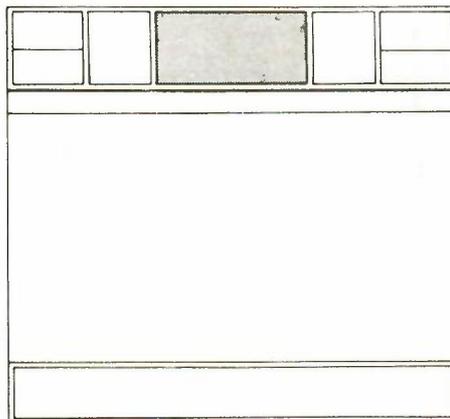


Figure 2.10: The Text Options.

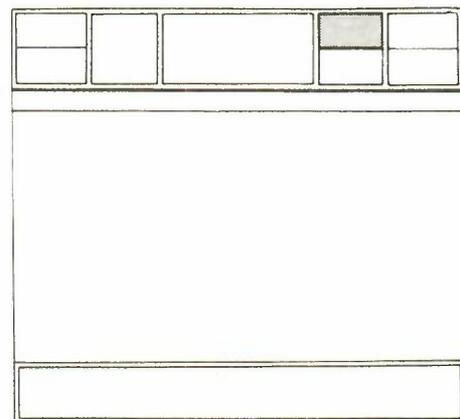


Figure 2.11: Style.

2.7 STYLE A further option in the command area is headed **STYLE** and is used to modify the appearance of the text in your document. You should press **F4** to use this option.

When you do so you are given four choices: to use bold (or heavy) type; to display high script (superscript); to display low script (subscript) or to produce underlined text. Any one of these is brought into effect by pressing **F4** and then a single key from the list shown in the control area. As an example let us use this option to produce underlined text. Press **F4**, then the **U** key, followed by **ENTER**.

After you have pressed **ENTER** the display returns to normal and nothing seems to have changed, except that the style is marked as "UNDERLINE" in the status area. If you now type in some more text you will see that it is underlined as it is displayed in the text area.

PROVISIONAL

In QUILL you always see exactly what will appear in the final printed version. This removes all the frustration that is normally associated with the production of a properly laid out document.

How do you turn off the underlining? Fortunately this is as simple as turning it on in the first place. All you have to do is press F4, the U key and **ENTER** again. If you now type in a few more words you will see that they are not underlined - the underlining option works like a simple on-off switch, or *toggle*.

You will find a fuller description of using underlining, and the other three style options, in Chapter 7.

A glossary is a useful way of storing a frequently-used sequence of keypresses for later recall. The sequence is stored under a single key. You can define a glossary entry by pressing **SHIFT** and F5 together, followed by the key under which you want the sequence to be stored. You then type the sequence of keypresses that you want to store, ending by pressing F5 again. The sequence of actions is carried out as it is stored.

You can recall the sequence at a later time by pressing F5 and then the key under which you stored the sequence.

Glossaries are described in more detail in Chapter 8.

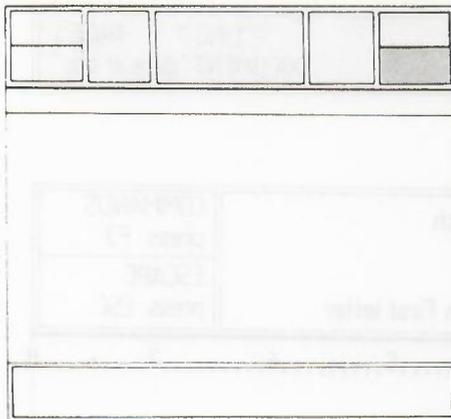


Figure 2.12: Glossaries.

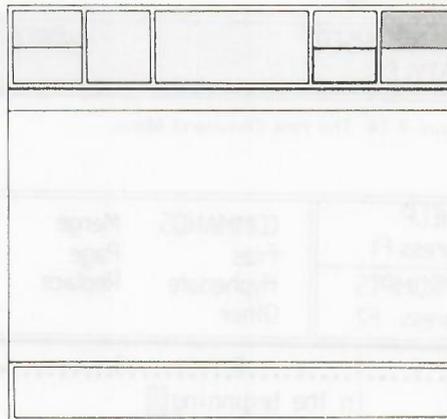


Figure 2.13: The Commands.

Another option available from the main display is to use a command.

You gain access to the commands by pressing F3, as indicated at the right hand side of the control area. When you press F3 the control area will again change, this time to show a list of the commands that are available. The screen will change to look like that shown in Figure 2.14.

You can select any of the commands shown in the menu (list) at the centre of the control area by pressing the key corresponding to its first letter.

QUILL has more commands than can be displayed in the command menu. You are therefore given two alternative lists and can switch between them with the Other command, which is shown in both lists.

If, when the command menu is shown (after you have pressed F3), you press the O key, the control area will change to show the second command menu, illustrated in Figure 2.15.

If you press the O key again the original list of commands will reappear. You can use the Other command to switch between the two lists at any time that the command menu is displayed.

Since some commands start with the same letter it is important to check that the command you want is displayed in the control area before you select it.

2.8 GLOSSARIES

2.9 COMMANDS

PROVISIONAL

HELP press F1	COMMANDS Copy Design Other	Erase Footer Goto	Header Justify Load	Margins Print Quit Press First letter	Save Tabs	COMMANDS press F3 ESCAPE press ESC
------------------	-------------------------------------	-------------------------	---------------------------	--	--------------	---

.....1.....2.....3.....4.....5.....6.....7.....8

In the beginning

command>

MODE: INSERT STYLE:	WORDS: 3	LINE: 1 PAGE: 1 DOCUMENT: "default.doc"
------------------------	----------	---

Figure 2.14: The First Command Menu.

HELP press F1	COMMANDS Files Hyphenate Other	Merge Page Replace	Search Zap Press First letter	COMMANDS press F3 ESCAPE press ESC
------------------	---	--------------------------	-------------------------------------	---

.....1.....2.....3.....4.....5.....6.....7.....8

In the beginning

command>

MODE: INSERT STYLE:	WORDS: 3	LINE: 1 PAGE: 1 DOCUMENT: "default.doc"
------------------------	----------	---

Figure 2.15: The Second Command Menu.

PROVISIONAL

While you have the list of commands displayed, press the Help key and you will find that the information given is concerned with how to select a command. When you leave Help (by using **ENTER** or **ESC**) you will, of course, be returned to the display of the list of commands, ready for making a selection.

Choose any one of the commands and press the key corresponding to its first letter. The control area will again change to confirm your selection and to inform you what options are available for that particular command. If you like, you may press the Help key again at this stage when you will find that the information given will be how to make use of the command that you have selected. When you return from Help you will be left with the selection of options available for your chosen command.

Since the actions of the commands have not yet been explained, you should cancel the command by pressing **ESC**, when you will be returned to the main display.

We have seen how the Other command works. The descriptions of the various commands will take up much of the rest of this manual and we shall, for the moment, describe the use of just two more commands, Quit and Zap.

You use Quit to leave QUILL and will therefore need it every time you use the program. Press **F3** and then the **Q** key. You will then be asked whether you want to save your current document on a Microdrive cartridge (just press **ENTER**) or to abandon it (press **A**). Choosing to save the document leads you into a sequence similar to that of the Save command, described in Chapter 4. Provided you do not have anything you want to save you can press the **A** key. You will then be returned to SuperBASIC and will have to reload QUILL if you want to continue using it.

A third alternative is to press **ESC**. This will cancel the command and you will go back to your document.

2.10 ESCAPE

The Zap command is in the second command menu and so you will have to use the Other command before selecting Zap. (If you just press **F3** and then **Z**, you will not select the command.) You must press **F3**, **O** and then **Z**. **Zap clears from memory the text of the current document, but does not return to SuperBASIC.**

If you clear the text before you have saved it on a Microdrive cartridge you will not be able to recover it without typing it in again. When you select this command, QUILL will therefore ask you to confirm your choice by pressing ENTER. You have the alternative choice of pressing **ESC** which cancels the command and returns you to your document.

If you press **ENTER** the text of your document is cleared and QUILL is left in the state it had when you first loaded it.

You can generally use the **ESC** key to cancel the current action, or to go back to the main display. We have already seen how it is used to leave Help and to cancel the selection of the Quit and Zap commands.

In almost all cases pressing **ESC** during a partially-completed command will cancel the rest of the command, without undoing any of the completed sections (see, for example, the descriptions of the Copy and Replace commands in Chapter 6). There is only one exception to this rule - pressing **ESC** during the use of the Design command will cancel any Design options (eg the number of lines per page) that you have just set.

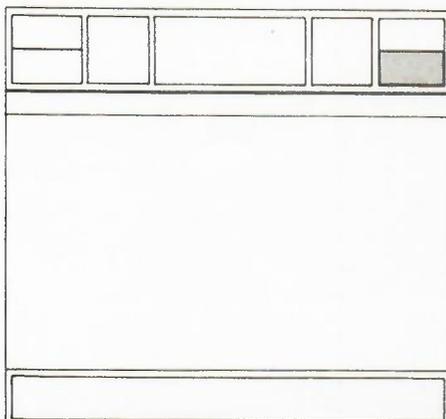


Figure 2.16: Escape.

2.11 THE MICRODRIVES

You would be wise to make a copy of QUILL on another cartridge, just in case of accidents. You can do this with the Backup option of the Files command in the following way:

```
F3 F B MDV1__QUILL ENTER MDV2__QUILL ENTER
```

These two commands will Save and Load applications on the cartridge in the *current drive*. This is the drive that was last used and, when you have just loaded QUILL as described in Section 2.1, this will be drive 1.

You can put a Microdrive cartridge in the second drive and load from or save to it, provided that you include the drive specifier in the name of the first file you access on that drive. (See Section 9.3 for a full description of Microdrive file names.)

You could, for example, load a file called "LETTER__DOC" from drive 2 by using the Load command as:

```
F3 L MDV2__LETTER__DOC ENTER
```

You do not need to include the extension. QUILL will assume that it is __DOC unless you type in something different. You could therefore just use:

```
F3 L MDV2__LETTER ENTER
```

Remember that this will make drive 2 the current drive so that you could now load another file, called "REPORT__DOC" from drive 2 by:

```
F3 L REPORT ENTER
```

Drive 2 will remain the current drive until you specify a different drive specifier in a file name.

3.1 INTRODUCTION

In this chapter you will learn how to use the simple editing facilities of QUILL. The changes to the text will always take place at the position of the cursor, which should therefore first be moved to the place where you want to make alterations. This form of editing is known, for fairly obvious reasons, as *cursor editing*. You may practise using these techniques on a piece of text that you type in yourself, or you may use the text provided with QUILL. If you type in your own text, as was described in Chapter 2, do not worry about any mistakes you make. In fact it may be a good idea to add deliberate mistakes - each mistake will give you extra practice in using the editing facilities.

To insert letters or words into the middle of the text, you should follow the following steps.

- 1) Move the cursor, by using the four cursor keys, to the point where you want to make the insertion.
- 2) Type at the keyboard the letters or words that you want to be inserted.

That's all there is to it! As you will see, the text is rejustified automatically as you make the insertion.

If you insert several words it would be painful to have to wait while the text adjusted each time you press a key. QUILL detects this situation and reacts by splitting the line at the point where you are inserting text. You can then type in as much text as you like. Figure 3.1 shows a split line during insertion of text.

HELP press F1	CURSOR move ↑ with ← → keys ↓	TEXT Type to insert at Press ENTER for new para DELETE SHIFT KEY & ← → REPLACE SHIFT & F4	STYLE Press F4 GLOSSARY Press F5	COMMANDS press F3 ESCAPE press ESC
------------------	--	--	---	---

.....1.....2.....3.....4.....5.....6.....7.....8

In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon face of the deep. And the spirit of God moved upon the face of the waters.

And God said, Let there be light! □

and there was light.

And God saw the light, that it was good; and God divided the light from the darkness. And God called the light Day, and the darkness he called Night. And the evening and morning were the first day.

And God said

MODE: INSERT	WORDS: 95	LINE 4 PAGE 1
STYLE:	DOCUMENT: "default.doc"	

Figure 3.1: Inserting Text - A Split Line.

QUILL will restore the text when you finish inserting text at that point (ie when you press a cursor key, a function key or ENTER).

The deletion of text at the cursor position is also very simple. You use the **SHIFT** key together with either the left or the right cursor key.

To see the action of the left cursor key, position the cursor immediately after the text that you want to delete. Now press **SHIFT** and, keeping it pressed, also press the left cursor key briefly. This will cause the letter immediately to the left of the cursor position to be deleted and the cursor will move one space to the left. Each time you press the

3.2 INSERTING TEXT

3.3 DELETING TEXT

left cursor key, with the shift key held down, one more letter will be deleted as the cursor moves to the left. If you wish to delete several letters you can also hold the left cursor key down, using the auto repeat facility. Always release the cursor key before releasing the shift key.

If you use **SHIFT** together with the right cursor key, text will be deleted, character by character, from beneath the cursor position, and the text to the right will close up to fill the gap. You are deleting text to the right of the cursor.

You can delete whole words at a time, to the left or to the right of the cursor, by pressing **SHIFT** and **CTRL** together and, while holding them down, pressing either the left or the right cursor key.

If you press the **CTRL** key and, while holding it down, press either the up or the down cursor key you will delete the whole of the line to the left or to the right of the cursor.

In all cases the text will be rejustified automatically.

3.4 REPLACING TEXT

In the context of cursor editing, replacement is understood to mean the overwriting of existing text by the new text typed at the keyboard. In order to replace text, QUILL must be changed from insert to overwrite mode.

As indicated in the text option shown in the control area, you can do this by pressing **SHIFT** together with **F4**. When you do this, two things will happen. The mode indicator at the left hand side of the status area will change from "INSERT" to "OVERWRITE". It now shows that text typed at the keyboard will replace existing text. Pressing **SHIFT** and **F4** again will change back to insert mode.

With QUILL set to overwrite mode, position the cursor at the start of the text to be replaced and type in the replacement over the old text. Once you have finished making replacements, it is a good idea always to return to insert mode.

Figure 3.2 shows a typical situation where you would want to use the overwrite mode. With the display as shown, with the cursor on the "n" of "sentence", you can overwrite with "e" and "n" to correct the word. (You can also practice deleting a character by removing one of the "m"'s from "ammended"!)

HELP press F1	Cursor move ↑ with ← → keys ↓	TEXT Type to insert at Press ENTER for new para DELETE SHIFT KEY & ←↑↓→ REPLACE SHIFT KEY & F4	STYLE Press F4 GLOSSARY Press F5	COMMANDS press F3 ESCAPE press ESC
------------------	--	---	---	---

.....1.....2.....3.....4.....5.....6.....7.....8

 this is a sentnece to be ammended

MODE: OVERWRITE	WORDS 7	LINE 1 PAGE 1
STYLE:		DOCUMENT Default.doc

Figure 3.2: Overwriting.

When you have produced a document you will probably want to save a copy of it on a Microdrive cartridge. At some later date you may want to make some changes and keep a copy of the new version. If you have a printer you will certainly want to produce printed copies.

This chapter describes the commands provided to perform these operations, and also other functions relating to documents stored on the Microdrives. Remember that all commands are called in the way described in Chapter 2, that is, by pressing F3 and then pressing the appropriate letter.

You use this command to save a copy of the text of a QUILL document on the current drive. When you call the command you are asked to type in a name for the document. The simplest way to use the command is, therefore to type in something like the following sequence.

F3 S MYLETTER ENTER

You will usually want to save the document with a name that is different from that of any other document saved on that cartridge. (If you use the same name you will normally want to replace the old version with a new one.) This assumes that you can remember the names of all the documents that you have saved on that cartridge - what if you have forgotten some of the names?

If you are not sure about the names you have used before you can ask for a list of the names of all documents saved on the cartridge, before choosing a name for your new document. Instead of typing in the name immediately, you can type in a question mark, eg

F3 S ?

You will then be shown a list of the names of all the existing documents on the current Microdrive cartridge and are again asked to type in a new name for the document to be saved. You should then choose a name and type it in, ending with **ENTER**. If this name is the same as that of a document which is already saved on the current drive, you will be asked if you want the new document to replace the old one. In response you should either type in Y (yes) or press **ESC**. If you type a Y the old version of the document will be replaced with the new one, otherwise you will be asked to type in another, different, name. When the document has been saved QUILL returns to the main display.

When you name a document to save it, or if you load a previously saved document, you will see that QUILL displays the document name in the status area. If you want to resave such a named document QUILL offers you a further option - to replace the old version on the cartridge, without having to type in the document name. In such a case you can just type in:

F3 S ENTER

when the new version will be saved on the Microdrive cartridge, replacing the old one.

Before you have named a document QUILL supplies the name "DEFAULT__DOC". This is the name that will be used to save the document if you do not type in a name of your own.

You should use the Load command when you want to copy a document from the current Microdrive cartridge into the computer's memory so that it may, for example, be re-edited.

You are first asked to type in the name of the document you want to load. You can either enter the name or type in a question mark, when QUILL will display a list of all the documents on the current Microdrive cartridge and again ask you to type in the name.

If the name you type in does not correspond to the name of an existing document an appropriate error message will be given.

4.2 SAVE

4.3 LOAD

4.4 PRINT This command is used to produce a printed copy of all or part of a QUILL document. It is, of course, necessary that you have a printer and that it is correctly connected to the computer, otherwise nothing much will happen!

When you use this command you must first specify the document you wish to be printed. This may be either a named document, which has previously been saved on the current Microdrive or the current document in memory (which may or may not have a name).

As with the Save and Load commands you may, before entering the name, type in a question mark for a list of all the available files.

When you have entered the name of an existing document you will be asked if you want the whole document to be printed and should type either Y (yes) or N (no). If you type Y, the printing will start immediately, otherwise you will be asked for the number of the page of the document at which you want printing to start and also the page number of the last page to be printed.

Please note that QUILL will work with most makes of printer, although you may have to make some modifications in the case of a printer which requires unusual control codes for such operations as turning on and off the underlining. Details of these modifications are given in the general section of the manual, where the printer driver program is described.

You may also wish to change things such as the line spacing, the number of characters per line and the number of lines per page of the printed document. These are all included in the Design command, which is described in Chapter 7.

4.5 FILES The Files command is used to make modifications to entire documents, which have been saved on the current Microdrive.

Your options under this command are

Rename - to change the name of an existing document. You will first be asked to type in the old name of the existing document and then to type in the new name that you wish it to have. For example, to change the name of a document called "MYLETTER" to "NEWLETTER", you would type:

F3 F R MYLETTER ENTER NEWLETTER ENTER

Delete - to erase an existing file. You are asked to type in the name of the file which you want to delete. For example, to delete a document called "MYLETTER" you would type:

F3 F D MYLETTER ENTER

Please note that this command is NOT REVERSIBLE and should therefore be used with care.

Import - to insert another file, exported from one of the other Psion 4D packages, into your document. It assumes a file name extension of __EXP. The document from the Microdrive cartridge is inserted into the current document at the cursor position.

To insert a document called "TABLE__DOC" into the current document you should first move the cursor to the point where you want it to be inserted and then type:

F3 F I TABLE ENTER

You must not use this option to insert another QUILL document - the Merge command is provided for this purpose.

Backup - This will make a copy of the file whose name you type in. It is sensible to make copies (on a different Microdrive cartridge) of all important documents, in case the original is lost or damaged. There is an example of its use, to make a copy of the QUILL cartridge, in Chapter 2.

This chapter is concerned with the *format* of the text; that is, the layout and appearance, as opposed to the actual content. You will learn how to move the position of the left, right and indent margins, and how to change the justification which, as you may remember from Chapter 2, affects the way the text is aligned with respect to the margins.

The positions of the margins are changed with the Margin command. The change in the margin positions takes effect from the current paragraph and continues until the end of the document, or until the next change of margin positions.

Press the command key (**F3**) and then the M key to start this command. In addition to other changes in the control area you will see that three choices: LEFT, INDENT and RIGHT will appear and that the LEFT option is highlighted. These options represent the three margins, and the one that is highlighted is the one that you can move. You can step from margin to margin by pressing the space bar. When a margin is highlighted you can move that margin by means of the left or right cursor keys.

Suppose you wish to move the left margin to the right by three characters, starting with the second paragraph of your document.

You should first move the cursor to any point in the second paragraph and then type:

F3 M

As indicated by the highlighting, the left margin is the one you can move, so you just have to press the right cursor key three times. The change in the margins takes place immediately, so that you can see the effect before you leave the command.

You can leave the command straight away by pressing **ESC**, or you can continue to make further margin changes. Press the space bar until the correct margin is selected and move it with the left and right cursor keys. You can use the up and down cursor keys to move the cursor to another paragraph and make further changes to the margins. After you have made all the changes you want you can leave the command and go back to the command menu by pressing **ESC**.

The indent margin marks the character position which is used for the start of a new paragraph. It is initially set at the tenth character position.

There is no restriction on the relative positions of the indent and left margins. If you do not want to use indented paragraphs you may move them so that they are both in the same place. You may even place the indent line to the left of the left margin. This is useful for producing numbered paragraphs as shown in the following example.

Indent Margin

Left Margin

- 1) This is the first of a series of paragraphs to show how you can use indent margins.
- 2) The indent margin is three characters to the left of the left margin.

In this case, starting a new paragraph (by pressing **ENTER**) will allow text to be typed at the 'indent' position. Provided the text you type there does not pass the position of the left margin, the first time you press the space-bar will cause the cursor to move to the left margin. All following text will be displayed between the left and right margin positions until you press **ENTER** again.

5.1 INTRODUCTION

5.2 CHANGING THE MARGINS

5.3 JUSTIFICATION

The Justify command allows you to alter the type of justification used in your document. As for the Margins command, all changes take effect from the current paragraph (that containing the cursor) and remain in force until the end of the document, or to the next change of justification. When you select this command you will see that you are offered the choice of left, right (and left), or centred justification.

When QUILL is first loaded it assumes right justification. The text is aligned on both the left and right margins, producing text with an appearance like that of this paragraph. If there are not sufficient characters on a line to make the margins match, extra spaces will be added between the words until they do. The final effect is very professional, but if you use an unusually large quantity of extra-long or hyphenated words in a document, unpleasantly excessive inter-word spaces can be produced.

Choosing left justification, by pressing the L key after calling the Justify command, will produce text which looks like the text in this manual. The left hand margin is aligned, but the spacing of the text within a line is not adjusted, so that the right hand margin is left uneven.

Centre justification, selected by pressing the C key, causes the text of each line to be centred between the left and right margins. The text could then appear as shown in this paragraph. Centre justification is useful, for example, in centering headings and titles, or for adding labels to diagrams.

This chapter will extend your knowledge of the editing facilities to include block copies, moves and erasures. In addition, the extremely powerful technique of search and replace editing will be introduced.

6.1 INTRODUCTION

In addition to copying a block of text from one place in the document to another, the Copy command also allows you to move blocks of text.

6.2 COPY

The only difference between copying and moving text is that, in the case of a copy, the original text is left in position so that you end up with two 'copies'. If you **move** some text the new copy is inserted and the old copy is deleted, so that you are left with only one version. The Copy command gives you the option of leaving or of deleting the old copy and therefore gives you both facilities in a single command.

When you invoke the Copy command (by pressing **F3** and then the C key) you are first asked to move the cursor to the beginning of the text you want to copy. You should move the cursor with the cursor keys and then press **ENTER**. You are next asked to move the cursor to the end of the text to be copied. When you move the cursor the text that will be affected by the command is highlighted so that it is easy to see how much text will be copied. After you have selected the text you should again press **ENTER**.

In response to the next prompt you should move the cursor to the point where you want the selected text to be inserted and press **ENTER** once more. The copy will be made and inserted immediately.

You are then asked if you want to delete the old copy. You should press the D key to delete the old version (to produce the effect of a move) or the K key to keep it.

You can then terminate the command by pressing **ESC**, when you will go back to the command menu. If you press **ESC** before this point you will abort the command without any copies being made.

However, you also have an option of making further copies of the text at other places in your document. All you have to do is to move the cursor to the point where you want another copy and press **ENTER**. You can repeat this as many times as you want. When you have finished making copies you should press **ESC** to leave the command.

As is normal in QUILL, pressing **ESC** will cancel any partially-completed action, but will not undo anything that has been completed. All copies that you have made will be left in the text when you press **ESC**.

You should use this command if you want to remove any large blocks of text from your document.

6.3 ERASE

As with the Copy command, you are asked to move the cursor to the start of the text to be erased and then to press **ENTER**. You then have to move the cursor to the end of the text - again the text which will be affected is highlighted so that you can see exactly how much text will be removed. When you are satisfied that you have marked the correct amount of text you should press **ENTER** and the marked text will be erased immediately.

Remember that you can delete small pieces of text with the delete character, delete word and delete line operations described in Chapter 3.

6.4 SEARCH The Search command allows you to search for a particular sequence of characters, through all or part of your document. The first search will start at the beginning of the text, but can be continued from the last found position.

When you use the command you are asked to type in the text which you want to find, finishing with **ENTER**. Quill will immediately start searching your document from the top until it finds the first occurrence of the text. The cursor is left positioned at the start of the found text. If this is the occurrence you want you can leave the command by pressing **ESC**.

However, once you have given the command some text you can use it again to find the next occurrence of that text. Instead of pressing **ESC**, just press **ENTER**. If you do this QUILL continues the search from the current cursor position until it finds the next occurrence of the given text. You can repeat this as many times as you like, finding successive occurrences. Press **ESC** to leave the command when you have found the occurrence you want.

Suppose that you want to search your document for occurrences of the word 'river'. You will therefore press **F3** and then the O (the command is in the second command menu) and the S key. Then type in the word 'river' (do not type the quotation marks unless they are part of the text you are looking for) followed by **ENTER**. The search will start immediately and stop with the cursor positioned on the first character of the first occurrence of 'river'.

If you want to find the next occurrence you should then just press **ENTER**. This reactivates the command and finds the next occurrence of 'river'. These steps may be repeated as many times as you want, so that all occurrences of 'river' can be located. Press **ESC** when you want to leave the command.

6.5 REPLACE The Replace command is similar to the Search command, but also gives you the ability to replace some or all of the occurrences that are found.

You are asked to type in the text to be found, and are then asked to type in the replacement text (each piece of text should be ended by pressing **ENTER**).

Quill searches from the top of your document until the first occurrence is found, as for Search, and then asks if you want to replace the found text. If you press the Y key the text is replaced; if you press any other key (except **ESC**) the old text will be left. In either case Quill then continues the search for the next occurrence and offers you the same choice of leaving or replacing the found text. This continues until no further occurrences are found, or until you press **ESC**.

You can use the command to make multiple insertions by making the replacement text include the old text. Multiple deletions are also simple; all you have to do is to press **ENTER** immediately when you are asked to type in the replacement text.

Replacing, inserting and deleting are illustrated in the following three examples:

- 1 - to replace occurrences of 'river' by 'stream', give 'river' as the text to be found and 'stream' as the replacement text.
- 2 - to insert 'or stream', give 'river' as the text to be found and 'river or stream' as the replacement text.
- 3 - to delete 'river', give 'river' as the text to be found and give no replacement text (just press **ENTER**).

In this chapter we shall cover the remaining options for modifying the appearance of the text. It includes the setting of tab stops and page breaks, and using bold characters, underlining, subscripts or superscripts. In addition there is a section on the Design command, which you can use to change the default options used to control the overall appearance of your documents.

7.1 INTRODUCTION

A very common way of controlling the layout of a document is by use of tab stops. These are marked positions, at particular columns of the text of your document. When you press the **TABULATE** key, the cursor will move to the right, from its present position, to the next tab stop in the line. If you have passed the last tab stop, then pressing the **TABULATE** key will move you to the start of the following line.

7.2 TABS

7.2.1 Using Tab Stops

QUILL allows you to use tab stops of several different types, and to position them in any column. You can have up to sixteen tab stops in a line of your document.

7.2.2 Tab Stop Types

Each tab stop may be set, independently of each other, to be any one of four types.

The most common type is known as a *Left* tab stop and this works in exactly the same way as the tab positions on a normal typewriter. When you press the **TABULATE** key the cursor will move to the next tab position and any text you type in will start at the tab column. It is called a *Left* tab since the text at such a tab stop on successive lines is aligned at its left hand edge.

A second type is a *Right* tab stop. When you move to such a tab stop and start typing, the cursor will remain at the tab position and the text will appear to the left, so that it ends at the tab position. This will continue until the text to the left of the tab position has filled the space available, or until you press the **TABULATE** key again to move to the next tab position. The text at such a tab stop on successive lines is aligned at its right hand edge.

There is also a *Centred* tab stop. Text typed at such a tab position will be adjusted so that its central character is positioned on the tab stop. Again the aligning of the text will continue until the available space is filled, or you press the **TABULATE** key again.

The fourth type of tab stop is a *Decimal* tab, and is used for typing in numerical values. When a number is typed at such a tab stop it is positioned so that its decimal point is at the tab column. If you do not type a decimal point in the text, it will behave like a *Right* tab.

Figure 7.1 shows the appearance of text typed at each of the four different types of tab stops.

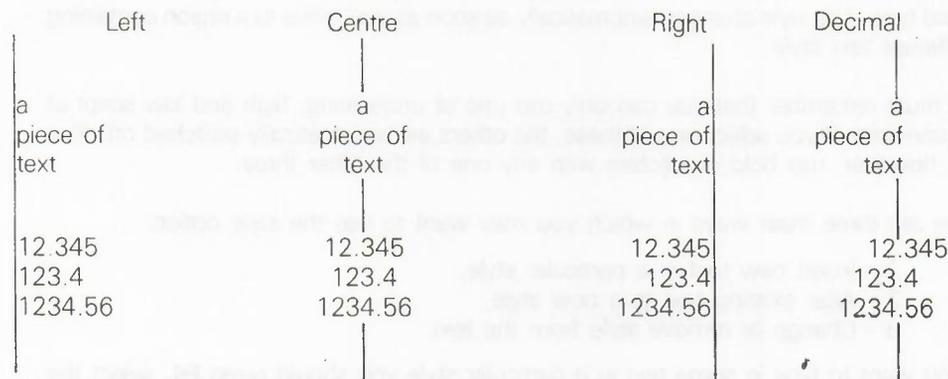


Figure 7.1: The Four Types of Tab Stop.

PROVISIONAL

7.2.3 The Tabs Command

When you have just loaded QUILL the tab stops are set at every tenth character position and are all Left tabs. You can change the number, position and type of tab stops with the Tabs command.

When you call the command the tab positions are drawn in the display, immediately beneath the ruler, as shown in Figure 7.2.

Each tab stop is marked by a letter (L C R or D) to indicate its type. The cursor is positioned at the beginning of the line. You can move the cursor to the left or right by using the left and right cursor keys.

You can place tab stops at any point in the line and mix the different types in any way you like. The only limit is that you may not have more than sixteen tab stops in the line. The new tab stops take effect from the current paragraph (that containing the cursor when you called the Tabs command) to the end of the document, or to the next change of tab positions.

When you have made all the changes that you want you can leave the command by pressing the **ESC** key, when you go back to the command menu.

Deleting a Tab Stop You can remove a tab stop by moving the cursor until it is over the tab marker and pressing the X key.

Inserting a Tab Stop The type of tab stop that will be inserted is that shown in the control area. You will see that the control area contains the words (L)eft, (R)ight, (C)entre and (D)ecimal and that the word (L)eft is highlighted. This shows that the next tab stop to be inserted will be a Left tab. You can change its type either by pressing the space bar (each time you press it the highlight moves from one type to the next) or by pressing the key corresponding to the first letter. If you want to change to a Right tab, for example, you can either press the R key, or keep pressing the space bar until the word (R)ight is highlighted.

All you have to do to insert a tab stop is to select the type you want, move the cursor to where you want the tab to appear, and press **ENTER**.

7.3 STYLE The underlining facility has already been used as an example of the use of the style option (in Chapter 2). In this section we shall examine its use more fully, together with the options to use bold characters, high script (superscript) and low script (subscript).

In general you can select any of these options by pressing **F4** and then the appropriate letter - Bold, Underline, High script or Low script. If one of these options is currently switched on, you can turn it off again by the same method as you used to turn it on; that is by pressing **F4** and then the appropriate letter.

Note that any text that you type will always appear in the style shown in the status area. If you move the cursor into a region which is in bold type, for example, the status area will show BOLD style, and any further text that you type within this region will also be in bold type. The style changes automatically as soon as you move to a region containing a different text style.

You must remember that you can only use one of underlining, high and low script at any one time. If you select one of these, the others are automatically switched off. You can, however, use bold characters with any one of the other three.

There are three main ways in which you may want to use the style option:

- 1 - Insert new text in a particular style,
- 2 - Alter existing text to a new style,
- 3 - Change or remove style from the text.

If you want to type in some text in a particular style you should press **F4**, select the style you want and then press **ESC**. Any text that you then type in, whether in insert or overwrite mode, will appear in the style you have selected. When you want to return to normal text you should switch off the style by pressing **F4**, deselect the style and press **ESC** again. Figure 7.4 shows underlined text being typed in.

PROVISIONAL

It is easy to add a particular style to existing text. The method is known as *painting* since you use the cursor like a paint brush, changing the style of any text over which it moves.

First you must move the cursor to the start of the text to be changed, press **F4** and select the style you want. Do not press **ESC**, but use the cursor keys to move the cursor through the text to be changed. When you reach the end of the text you want to alter, leave the option by pressing **ESC**. You do not need to switch off the style selection; it will revert to the correct style as soon as you move away from the area painted in the new style. Figure 7.5 shows the appearance of the screen while painting text with underlining.

HELP press F1	CURSOR move ↑ with ← → keys ↓	TEXT Type to insert at Press ENTER for new para DELETE SHIFT KEY & ← → REPLACE SHIFT & F4	STYLE Press F4 GLOSSARY Press F5	COMMANDS press F3 ESCAPE press ESC
------------------	--	--	---	---

.....1.....2.....3.....4.....5.....6.....7.....8
In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon face of the deep. And the spirit of God moved upon the face of the waters.
And God said, Let there be light; and there was light.
And God saw the light, that it was good; and God divided the light from the darkness. And God called the light Day, and the darkness he called Night. And the evening and morning were the first day.
And God said, <u>Let there be</u> █

MODE: STYLE UNDERLINE	WORDS: 98	LINE: 8 PAGE: 1 DOCUMENT: "default.doc"
--------------------------	-----------	--

Figure 7.4 Inserting Underlined Text

You can change, or remove, an existing style in the same way as you use to add a new style to existing text. Again you should move the cursor to the start of the text before pressing **F4** and selecting (or deselecting) the style you want. Move the cursor to the end of the text and press **ESC**.

When you change text from an existing style to a new one, QUILL does not remember the original style. Suppose, for example, you change text which was originally underlined to being in bold characters. If you later remove the bold style, the final text will be in plain characters, and will not revert to being underlined.

7.4 PAGE BREAKS

A page break marks the point in your document where a new page will start, irrespective of the length of the page that you have set in the Design command. It is very useful for making sure that a section of text, such as a list or a table, is started at the top of a new page and is not, therefore, shown in two parts on different pages.

You can set a page break at any time by using the Page command - press **F3** and then the P key. You should then position the cursor in the line at which you want the page to end and press **ENTER**. This will cause a new page to be started at the end of the line in which the page break was set.

You can clear a page break by moving the cursor until it lies on the page break and then pressing **SHIFT** and, while holding it down, pressing the right cursor key.

PROVISIONAL

7.5 DESIGN

You should use the Design command to change features that affect the entire document, both in the main display and on the printer. It controls such things as the number of characters on each line of the document, the spacing between the lines and the number of lines per page of the document. The appearance of the screen for this command is shown in Figure 7.6.

When you call the Design command the display of your document is replaced by a list of the properties you can affect. These include whether you display 40, 64 or 80 characters on a line of the display, the number of lines per page of your document, and so on. A full description of each option appears in Chapter 9.

Suppose you want to change the number of characters per line to 40, so that the display is suitable for a domestic television. All you have to do is to call the Design command and, when the list of topics is displayed, press the D key to select the 'Display width' option. This option is automatically highlighted and QUILL waits for you to press 4, 6 or 8 to select 40, 64 or 80 characters per line. It will not accept any other key presses. In this case you would press 4 for a 40 character display.

HELP press F1	CURSOR move ↑ with ← → keys ↓	STYLE—Turn style on/off Press key B,H or U To paint style use ←↑↓→ Press ENTER to end	STYLE Press F4 GLOSSARY Press F5	COMMANDS press F3 ESCAPE press ESC
------------------	--	--	---	---

.....1.....2.....3.....4.....5.....6.....7.....8

In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon face of the deep. And the spirit of God moved upon the face of the waters.

And God said, Let there be light ; and there was light.

And God saw the light, that it was good; and God divided the light from the darkness. And God called the light Day, and the darkness he called Night. And the evening and morning were the first day.

And God said

STYLE—Key B=Bold H=High L=Low U=Under

MODE	WORDS: 95	LINE: 4	PAGE: 1
STYLE: UNDERLINE		DOCUMENT: "default.doc"	

Figure 7.5 Painting Underline Style

You then have the option of selecting a further item to change - you can change any or all of the items listed in the display. When you have made all the changes you want you should leave the command by pressing the X key, as indicated in the control area.

This is the one case where you do not normally leave by pressing **ESC**. Normally when you press **ESC**, any changes that you have completed are kept; only half-completed actions are aborted. The Defaults command is an exception. If you leave it by pressing **ESC** any design options that you have made during that use of the command are cancelled.

HELP press F1	DESIGN the FORMAT the printed page Press the first letter of option	COMMANDS press F3
PROMPTS press F2	When finished press KEY X	ESCAPE press ESC

.....1.....2.....3.....4.....5.....6.....7.....8

Bottom margin (type No. & RET) ----- 6

Characters/inch (type No. & RET) ----- 10

Display width 80, 64, 40, (8,6,4) ----- 8

Gaps between lines (0 1,2) ----- 0

Lines per page (type No & RET) ----- 16

Start page no (type No. & ENT) ----- 0

Type density (single or double) ----- S

Upper margin (type No. & RET) ----- 0

View a wide document layout ----- OFF

MODE: INSERT	WORDS: 371	LINE: 16	PAGE: 1
STYLE:		DOCUMENT: "default.doc"	

Figure 7.6: The Design Command.

7.6 WIDE DOCUMENTS

The right margin is initially set at column 80 so that you can have up to 80 characters per line in your document. You may move the right margin further to the right - up to a maximum of column 160.

It is then impossible for QUILL to show the full width of your document on the screen. In this situation the display area acts like a window, through which you see only part of the full width of your document. As you move the cursor along a line, the window will slide across the width of your document, so that it always shows the region containing the cursor. This behaviour, which is known as *sideways scrolling*, is an extension of the normal up and down movement to show part of a document which has too many lines for them all to be seen at once.

One of the options of the Design command is to view a wide document. This changes the display so that you can see the general layout of a document that is more than 80 (or 64 or 40, depending on the setting you have selected for the display width) characters wide.

In this option each character is represented by a small block. You will be able to distinguish the different punctuation marks - commas, full stops, hyphens and so on - and see the structure of a whole page. You will not be able to read the characters and will have to use the Design command again to switch off this option before you can continue editing the document.

The provision of glossaries is a powerful feature of QUILL, in that it allows you to perform commonly-used operations by pressing only two keys.

One of the most common uses is to enable you to insert into your document a common word or phrase, without having to type it in full every time. The following example shows how to define a glossary entry by storing the text 'Yours faithfully,' under the Y key.

You may only define a glossary entry from QUILL's main display (eg you may not define a glossary entry from within a command).

To start a glossary definition, you press **SHIFT** and, while holding it down, press the glossary key (F5). You then release both keys and then press the key under which you want the entry to be stored. In this example, therefore, you will first press **SHIFT** and **F5** and then the Y key. You then type in the text, exactly as you wish it to appear:

Yours faithfully,

and then end the definition by pressing **F5** once more.

To use the glossary entry you simply press *f5* once and then the key under which the entry is stored. Thus if you now press *f5* and then the Y key, the text

Yours faithfully,

will appear at the current cursor position in the document.

If you redefine the glossary entry the new definition will replace the old one. For example, you could press **SHIFT** and **F5**, the Y key, then type:

Yours sincerely,

and end by pressing **F5** once more. The previous entry is lost. Pressing **F5** and the Y key will, in future, add the new phrase into your document. You can clear a glossary entry by redefining it to contain no text. If you press **SHIFT** and **F5**, then the Y key and follow this immediately by pressing **F5** again, you will have deleted that glossary entry.

When you leave QUILL, by means of the Quit command, all existing glossary definitions are saved onto the system Microdrive cartridge automatically. They are then loaded back into memory on any subsequent occasion that you load QUILL from that cartridge, so that there will be no need to keep redefining them.

8.1 INTRODUCTION

8.2 DEFINING A GLOSSARY ENTRY

8.3 USING A GLOSSARY ENTRY

8.4 REDEFINING A GLOSSARY ENTRY

8.5 KEEPING A GLOSSARY

The five function keys are used in QUILL for the following purposes:

Function Key	Use
F1	Help
F2	turn the prompts on and off
F3	call the command menu
F4	text style, and insert/overwrite
F5	use, and define a glossary entry

Notes:

- 1) Switching between insert and overwrite modes is selected by pressing **SHIFT** and, while holding it down, pressing **F4** (shift-F4).
- 2) Defining a glossary is selected by pressing **SHIFT** and, while holding it down, pressing **F5** (shift-F5).

9.2 FILES

9.2.1 File Names

A full file name consists of three sections, separated by underscores. The three components are:

an optional drive specifier	eg MDV1
a file name of up to x characters	eg LETTER
an optional three-letter extension	eg DOC

A full file name for an QUILL file could therefore be:

MDV2__LETTER__DOC

If you do not include a drive specifier in a file name then QUILL assumes that you are referring to the current drive, that is, the drive that was last used. The one exception is when you are loading QUILL itself from SuperBASIC, as described in Section 2.1. In this case you must include the drive specifier in the file name.

You do not normally need to specify an extension since QUILL supplies a default extension for every file access. The Load and Save commands supply a default extension of `__DOC`. The default extension for Import files is `__EXP`.

If you include an extension in any file name you type in then it will be used in preference to the default extension normally provided by QUILL.

Every time that an QUILL command asks you to type in a file name you have the option of pressing the `?` key to obtain a list of the names of files on the current drive. The file name `""*__*"` (file name and extension) will appear in the input line and, if you accept this by pressing **ENTER**, you will be given a list of all files on the current drive.

In this context the `""*"` character is a *wild card* which stands for any sequence of characters. You may also use the character `""?"` to represent any single character in a file name.

You have the option of using the line editor, described in Section 9.5, to modify the suggested file name, in order to obtain a list of the names of a particular group of files.

If, for example, you edit the file name to read `""*__TST"` and then press **ENTER** you will be given a list of the names of all files with an extension of `__TST`. Changing the file name to `""X*__*"` would result in a listing of all files, with any extension, whose names begin with X.

You could use the single character wild card as, for example,

MYFILE?__*

which would result in a listing of all files with names such as:

9.2.2 Wild Cards

and so on, with any extension.

Note: that this facility is only available when you are requesting a list of file names before typing in a file name for any of the file-based commands (Files, Load, Save and Print).

9.3 HELP Pressing **F1** displays a Help screen, containing information relevant to your current action and your possible options. You can ask for further information on any of the topics listed at the bottom of the display by typing in its name. Just pressing **ENTER** goes back one level in Help, until you reach the level at which you entered Help, when you will be returned to your document at the exact point where you left it. You can return to your document immediately from any level of Help by pressing **ESC**.

9.4 THE PROMPTS You can turn off the display of the control area and the prompts that it contains by pressing **F2**. This allows you to see more of your document on the display. You can restore the display of the control area by pressing **F2** again - each press changes the state of the display between on and off.

9.5 THE COMMANDS You gain access to the commands by pressing **F3**. This switches QUILT to display a command menu. In addition to being able to select a command you can, at this stage, move the cursor with the cursor keys. You are not allowed to insert or delete text ,or to define a glossary entry.

The control area display changes to show a list of the commands available. You select a command by typing its first letter. A further set of commands is available and you can select the alternative set by pressing the **O** key (Other). While you are in the command mode, the command menu switches between the alternative command sets each time you press the **O** key.

In general, you can leave any partially completed command by pressing **ESC**.

At the end of most commands QUILT returns to the main display.

In any command that requires text input (eg Save, Load, Files, Replace) you may edit the text with the aid of a line editor, similar to the line editor available in the other programs in the Psion 4D package. This line editor uses the following keys:

Key(s)	Action
Left cursor	Move one character to the left
Right cursor	Move one character to the right
Up cursor	Move one word to the left
Down cursor	Move one word to the right
CTRL + Left cursor	Delete one character to the left
CTRL + Right cursor	Delete one character to the right
CTRL + Up cursor	Delete all text to the left
CTRL + Down cursor	Delete all text to the right
SHIFT + Left cursor	Move left by one word
SHIFT + Right cursor	Move right by one word

The following commands are available.

COPY You should use this command for either moving or copying text from one place in the document to another.

You are first asked to move the cursor (with the cursor keys) to the start of the text to be moved and then to press **ENTER**. Next you should move the cursor to the end of the text you want to move and press **ENTER**. The text that will be affected is highlighted, for clarity. Following this you should move the cursor to the position where you want the marked text to appear, and press **ENTER** once more.

At this point a copy of the marked text is inserted at the cursor position and you are finally asked to press either the **K** or the **D** key, depending on whether you want to Keep or to Delete the original copy of the text.

PROVISIONAL

You are then given the opportunity of making further copies of the text at any other point in your document. You should position the cursor where you want another copy to appear and press **ENTER** to insert the copy. You may make as many copies as you like. When you have finished inserting copies you should press **ESC** to end the command and return to the command menu.

This command allows you to set or change a number of the default options which control the overall appearance of your document. Within the command you are asked to choose, by pressing the appropriate key, from the following options:

DESIGN

Bottom margin - type in the number of lines space to be left blank at the bottom of each printed page of your document. Press **ENTER** when you have typed in the number.

Characters/inch - type in the number of characters to be printed per inch. Press **ENTER** when you have typed in the number. (Normal values are 10 or 12, but this will depend on your printer.)

Display width - type in 4, 6 or 8 to select a display of 40, 64 or 80 characters per line in the display of your document. Quill will not accept any other characters.

Gaps between lines - type in 0, 1 or 2 to select how many blank lines will be printed between each line of text in your document. Quill will not accept any other characters.

Lines per page - type in the total number of lines to be used for each page of your document. This number includes the blank lines in both the upper and bottom margins. Press **ENTER** when you have typed in the number. If you type in a zero the document will not be split into pages. (You can normally print 66 lines on a standard A4 page.)

Start page number - type in a number, followed by **ENTER**. This number is used to number the first page of your document. Successive pages are numbered consecutively from this value. **QUILL** initially sets this number to be 1. You may want to change it if your document is a continuation of another document.

Type density - type in either S or D to select between single or double type density. Quill will not accept any other characters.

Upper margin - type in the number of lines space to be left blank at the top of each page of your document. Press **ENTER** when you have typed in the number.

View a wide document layout - use this option to switch to the display mode for examining the overall appearance of a document which is too large for you to see the full width of a page on the screen. Selecting this option a second time will return to the normal form of display.

At the end of each option you may select another of the options, or press the X key to leave the command. If you press **ESC** you will leave the command without having changed any of the design options.

This command allows you to erase text from your document. You are first asked to move the cursor (with the cursor keys) to the first character that you want to erase, and then press **ENTER**. You should then move the cursor to the character space following the last character to be erased and press **ENTER** again. The marked text is erased immediately.

ERASE

There are five options provided in this command:

FILES

- Backup** - to make a second, security copy of a document on a Microdrive cartridge. You are asked to type in the name of the document and the name you want to give to the new copy.
- Import** - to insert another file from a Microdrive cartridge into your document, at the position of the cursor. The file must be a file exported from one of the other programs in the Psion 4D package.
- Rename** - to change the name of a document or other file on a Microdrive cartridge. You are asked to type in the old name and then the new name.

Delete - to delete a named document or file from a Microdrive cartridge. You are asked to type in the name of the file you want to delete.

FOOTER This command allows you to specify a line of text to be used as the last line on each *printed* page. Note that the footer does not appear in the display of your document on the screen, but it is shown when you are viewing a wide document. (See the Defaults command).

You are first asked to select the position of the footer from the four options:

- None - no footer text
- Left - at the left margin
- Centre - centred in the page
- Right - at the right margin

You press the space bar until the required option is highlighted and then press **ENTER**. you are then asked to type the text for the footer, ending by pressing **ENTER**. You can include the page number anywhere in the text. The page number can be in one of three forms, depending on the three characters you type into the text to mark the position of the page number. The three options are:

Characters	Page Number Style
nnn or NNN	Arabic Numerals eg 1, 2, 3, 4
rrr or RRR	Roman Numerals eg I, II, III, IV
aaa or AAA	Alphabetic eg A, B, C, D

You are finally asked to type in a number, from 0 to 9 to indicate the number of lines space to be left between the bottom of the text and the footer.

GOTO You may use this command to move the cursor to the top, bottom or to a specified page in your document. You are offered three options:

- Top - to move the cursor to the beginning of your document.
- Bottom - to move the cursor to the end of your document.

- typing in a number, followed by **ENTER** moves the cursor to the start of that page of your document. If there are no page breaks in your document this option will move the cursor to the end of your document.

HEADER This command allows you to specify a line of text to be used as the first line on each *printed* page. Note that the header does not appear in the display of your document on the screen, but it is shown when you are viewing a wide document. (See the Defaults command).

You are first asked to select the position of the header from the four options:

- None - no header text
- Left - at the left margin
- Centre - centred in the page
- Right - at the right margin

You press the space bar until the required option is highlighted and then press **ENTER**. you are then asked to type the text for the header, ending by pressing **ENTER**. You can include the page number anywhere in the text. The page number can be in one of three forms, depending on the three characters you type into the text to mark the position of the page number. The three options are:

Characters	Page Number Style
nnn or NNN	Arabic Numerals eg 1, 2, 3, 4
rrr or RRR	Roman Numerals eg I, II, III, IV
aaa or AAA	Alphabetic eg A, B, C, D

You are finally asked to type in a number, from 0 to 9 to indicate the number of lines space to be left between the header and the top of the text.

PROVISIONAL

This command allows you to specify a point within a word where it can be split, with an automatically-inserted hyphen, if it extends beyond the end of a line. Words not marked in this way will, if necessary, be moved to the next line in their entirety.

HYPHENATE

You should move the cursor to the first character following the position where you want to allow a split to be made and press **ENTER**.

The command will have no apparent effect on the word if it is not at the end of a line.

You should use this command to select the type of justification you want. It takes effect from the start of the paragraph containing the cursor, and remains in effect to the end of the document, or to the next change of justification.

JUSTIFY

You are offered the following options, selected by pressing the key corresponding to the first letter of the option:

Left - the text is aligned at the left margin, but the right margin is left uneven.

Centre - the text of each line is centred within the line.

Right and left - additional spaces are inserted between words in each line so that both the left and right margins are aligned.

This command allows you to load a document into memory from a Microdrive cartridge, ready for printing or further editing.

LOAD

You are asked to type in the name of the document (the name you gave it when you saved it). If you just press '?' you will be shown a list of the names of all the documents saved on the current Microdrive cartridge, and again asked to type in a document name.

You should use this command to set or change the positions of the left, indent and right margins for your document. All changes in the margins are shown in the displayed text as you make them.

MARGINS

The control area shows the words **LEFT**, **INDENT** and **RIGHT**, and on first entering this command the word **LEFT** is highlighted. This means that you can use the left and right cursor keys to move the left margin.

You can select any other margin by pressing the space bar until the correct margin is highlighted in the control area. In each case you can move the selected margin by pressing either the right or left cursor key.

The change in each margin takes effect from the paragraph containing the cursor. It remains in effect to the end of your document, or to the next change of position of that margin.

When you have finished changing the margins you should press **ESC** to leave the command.

This command allows you to switch to the display of a further set of commands in the control area. The list of commands in the control area is switched between the alternative lists each time you use **Other**.

OTHER

Since several commands start with the same letter, you must make sure that the command you want is one of those displayed, before you choose it.

You can use this command to mark a point in your document where you want a new page to start.

PAGE

You should move the cursor to the point where you want the new page to start and press **ENTER**.

You can cancel a page break by moving the cursor to any point on the page break line and then pressing **SHIFT** and the right cursor key together.

PRINT This command prints all or part of either a named document from the current Microdrive cartridge, or the current document.

You are asked to type in the name of the document which you want to print - typing in a '?' at this point will give you a list of names of all the files on the current Microdrive cartridge. If, instead of typing in a name, you just press **ENTER** you will select the current document (in memory) for printing.

You are then asked if you want to print the whole document. If you reply by pressing the Y key the document will be printed immediately. If you press any other key you will be asked for the page numbers at which you want printing to start and stop. You should type in the two numbers (each followed by **ENTER**) and the pages within that range will be printed.

QUIT This command allows you to leave **QUILL** and return to SuperBASIC. You then have three options:

ENTER - to save your current document before returning to SuperBASIC. You are given the further option to type in a name for the saved document. If you again just press **ENTER** the document will be saved with its old name, replacing the old version of the document on the Microdrive cartridge.

A - to abandon your current document and return to SuperBASIC without saving it.

ESC - to cancel the command and return to your document.

REPLACE You can use this command to replace some or all occurrences of one piece of text by another.

You are first asked to type in the text to be replaced, followed by **ENTER**, and then are asked to type in the replacement text, again followed by **ENTER**.

QUILL searches from the start of the document until the first occurrence of the old text is found. It then offers you the option to replace the old text with the new. You should press the Y key to replace the text, or any other key if you do not want to replace it.

QUILL will then search for the next occurrence and again offer you the option to make the replacement. This process will continue until you reach the end of the document.

SAVE You use this command to save a copy of your document to a Microdrive cartridge.

You are asked to type in a name for your document, so that it can be identified. The document is then saved under that name. If, instead of typing in a name, you just press **ENTER**, the document will be saved with its old name, replacing the old version.

The text of the document remains in the computer's memory and at the end of the command you can continue working on it.

SEARCH This command searches your document for a particular word or phrase.

You are first asked to type in the text which you want to find. When you press **ENTER** **QUILL** starts at the top of your document and searches for the first occurrence of the text.

If, instead of typing in text, you just press **ENTER**, **Quill** will continue a previous search to find the next occurrence of the text you last typed in when using the Search command.

TABS The Tabs command allows you to specify the positions and types of tab stops on a line of text. Each change of the tab stops will take effect from the start of the current paragraph (the one containing the cursor). It will remain in effect to the end of your document, or until the next change of tab stops.

The tab positions are drawn on the display and the cursor positioned at the start of that line. You can move the cursor along the line by using the left and right cursor keys. Pressing the **TABULATE** key moves the cursor to the next tab stop to the right; pressing **SHIFT** and **TABULATE** moves to the next tab stop to the left.

PROVISIONAL

You can change the type of the tab stop by:

- a) pressing the space bar until the correct type is highlighted in the control area.
- b) pressing the L, C, R, or D key.

There are four types of tab stop provided:

Left - the tab stop behaves like a left margin; the text is positioned to the right of the tab stop.

Right - the tab stop behaves like a right margin; the text is positioned to the left of the tab stop.

Centre - the text will be centred around the tab stop.

Decimal - this is used for aligning decimal numbers. Each number will be positioned so that its decimal point is at the tab stop. Until a decimal point is encountered it behaves like a right tab.

You can remove a tab marker by deleting it (press **SHIFT** and the left or right cursor key).

To insert a tab marker you should select the type you want, move the cursor to the appropriate point and press **ENTER**. When you have made all the changes you want you should press **ESC** to leave the command and return to the command menu.

This command deletes the whole of your current document, without saving it on a Microdrive cartridge, allowing you to start again. QUILL returns to the same state as when it has just been loaded (see Section 9.8).

ZAP

You can change the style of the text in your document by pressing **F4** and then the first letter of one of the four options listed below. The selected style affects all text subsequently typed in.

9.6 STYLE

You are offered the following options:

Bold - text is converted to a bold, or heavy, typeface.

High script - text is printed in the upper half of the line.

Low script - text is printed in the lower half of the line.

Underline - text is underlined.

You then have the option of immediately pressing **ESC**, in which case all text that you type will be in the new style (until the next change of style).

Alternatively you can move the cursor from its present position with the cursor keys. All text that the cursor passes through will have the new style added to it. Press **ESC** to end this option and return to the main display.

You may select Bold style independently of the other three options, but you may only select one of High script, Low script or Underline to be active at any one time. If you select any one of these three the other two will be deselected automatically.

You can switch off any of the style options by the same method as you used to turn it on - that is by pressing **F4** and then the appropriate key (B, H, L or U).

When you have just loaded QUILL it is in insert mode, as shown at the left of the status area. In this mode any text that you type in will be inserted into your document at the position of the cursor. Any surrounding text will be spread out to make room for the inserted text.

9.7 INSERT
AND
OVERWRITE
MODES

If you press **SHIFT** and, while keeping it held down, press **F4**, QUILL will switch to overwrite mode. In this mode any text that you type in will replace, character by character, any text from the cursor position onwards.

You can switch back to insert mode by the same method, that is by holding **SHIFT** down and pressing **F4**.

9.8 GLOSSARIES

You can define a glossary action for any alphabetic key on the keyboard. Such a key can be made to produce any sequence of characters or commands, when used together with the glossary key (F5).

To define a glossary entry you should press **SHIFT** and, while it is still held down, press **F5**, release both keys and then press the key whose action you want to define. This key will then "learn" any sequence of key presses that you subsequently make (the sequence of actions is carried out at the same time). You should mark the end of the learning sequence by pressing **F5** once more.

At any later time you can use the glossary entry by pressing **F5** once, followed by the key whose action you have previously defined. If you attempt to use a key that you have not defined, nothing will happen.

You can define each key on the keyboard to have a different action, and there is no restriction on the number of characters stored - either under each key or in total. The stored sequences do, however, occupy some of the space in the computer that could otherwise be used for your document.

When you have defined one or more glossary entries they will be saved on the system Microdrive cartridge automatically when you use the Quit command. The glossary definitions are reloaded from the system cartridge every time you load QUILL.

If you add to or change the glossary definitions during a session with QUILL, the new glossary will replace the old one when you Quit.

9.9 THE START-UP PARAMETERS

When you first load QUILL it is in the following state:

Mode:	insert
Display width:	80 characters
Left margin:	0
Indent margin:	10
Right margin:	80
Upper margin:	6
Bottom margin:	6
Justification:	Right and Left
Tab stops:	Left, at columns 10, 20,30, ..., 80
Characters/inch:	12
Lines per page:	66
Gaps between lines:	0
Page header:	off
Page footer:	on, centred number of type NNN
Start page number:	1
Type density:	single
Style: Bold	off
Underline	off
High script	off
Low script	off

sinclair

QL

QL Abacus

1.1 INTRODUCTION

ABACUS is a 'thinking' *worksheet* which can be used for planning, budgeting, tabulating data, calculation, information storage or for the presentation of information. This information is represented, conceptually, on a huge, tabulated *grid* divided in 256 rows and 64 columns. The data area you see on the computer screen is a *window* on the grid, which you can move about the grid rapidly. The intersections of the rows and columns represent more than 16,000 *cells* or boxes in the grid. You can enter text into any cell or cells, or the cells may be used for the storage of numbers or data.

The real power of ABACUS, however, comes from the use of rules or formulae which can connect different blocks, rows or columns, or even cells of the grid. This means that data inserted in one area can immediately be evaluated and represented in another form elsewhere in the grid.

For example, you can use twelve of the columns to represent months of the year and you can then enter sales data along a "sales" row. The next two rows can contain formulae to calculate the cost of sales (as a percentage of sales plus a fixed cost, say) and the profit. The monthly profits will then be evaluated automatically each time you type in a sales figure. The yearly totals can also be summed by another formula, so that a change in the sales of, say, March will immediately lead to a completely different profit profile and total for the year. All the figures are evaluated by ABACUS automatically.

ABACUS is uniquely powerful as a worksheet in many respects. It will use the text that you enter to refer to columns, rows, cells and blocks on the grid. It also automatically applies the formulae you enter to whole rows, columns or blocks of cells - without recourse to any complex command structure.

ABACUS can be used in a very wide variety of planning and office tasks in finance, science, engineering, management and many other fields.

ABACUS is an "intelligent" worksheet because it uses the names you use, because it always prompts you with the most likely choices of parameters required and because it always informs and guides you in the entry of data and in the decisions which you must make. The software is self-documented and includes comprehensive Help files which you may enter at any stage, whatever you are doing in ABACUS.

ABACUS is also hugely powerful with an enormous range of in-built functions for operating on text as well as numbers. In addition, as you learn more about the use of this powerful package, you will come to learn of facilities and commands which allow great versatility and flexibility in your work. These commands include the joining of grids, the use of multiple windows, the variation of column widths, justification of text and the use of different units (including monetary, integer, decimal and exponential forms). You can also represent the data from ABACUS in graphics or in a table in the word processor, through the *export* commands of PSION 4D.

In fact in many respects ABACUS is like a visual programming language - but one which is easy to use. You may manipulate text, data, or formulae, you may use input and output statements and text variables. It is only one's imagination which limits the use of the applications and the possibilities of this program.

PROVISIONAL

CHAPTER 2 BASIC OPERATIONS

When you switch on the computer it will only respond to commands in BASIC. You will have to load ABACUS from its Microdrive cartridge. You will normally do so by inserting the ABACUS cartridge in drive 1 (the left hand drive) and then typing:

```
LGO MDV1__ABACUS ENTER
```

After a few seconds the screen will show the message:

```
ABACUS - Copyright Psion Ltd 1983  
Press any key to start
```

You should then press any key on the keyboard to start ABACUS.

When you have loaded ABACUS the display on the screen should look like that shown in Figure 2.1. This is known as the main display.

ABACUS allows you to choose whether to display eighty, sixty four or forty characters per line of the display. (You can change between them with the Design command, described in Chapter 6.) If you are using a domestic television the display may not be clear enough for you to see eighty or sixty four characters per line. If this is the case you will need to use the forty character display mode, and the main display will look like that of Figure 2.2. Apart from the difference in appearance ABACUS works in exactly the same way in all three modes. Most of the diagrams in this manual are shown for the eighty and sixty four character modes.

2.1 INTRODUCTION

2.2 GENERAL APPEARANCE

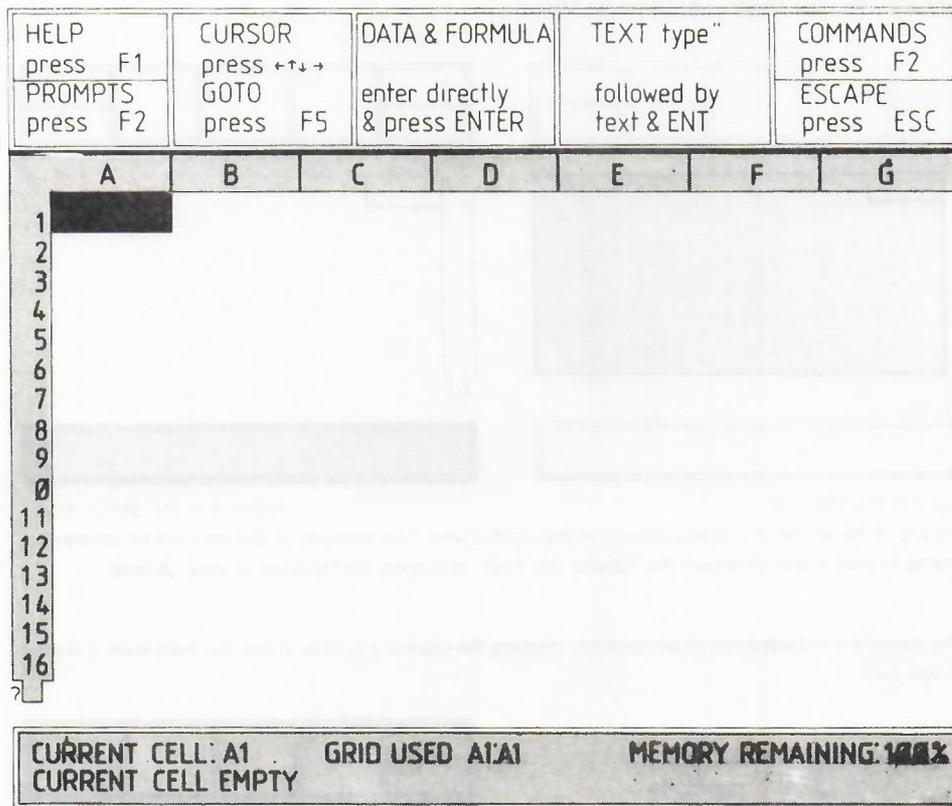


Figure 2.1 The Main Display (80 and 64 Character Mode).

The largest area of the screen contains the *window* showing part of the grid. The bottom two lines of the display contain the *status area*, which gives information about the current state of the grid. It is described more fully later in this chapter.

PROVISIONAL

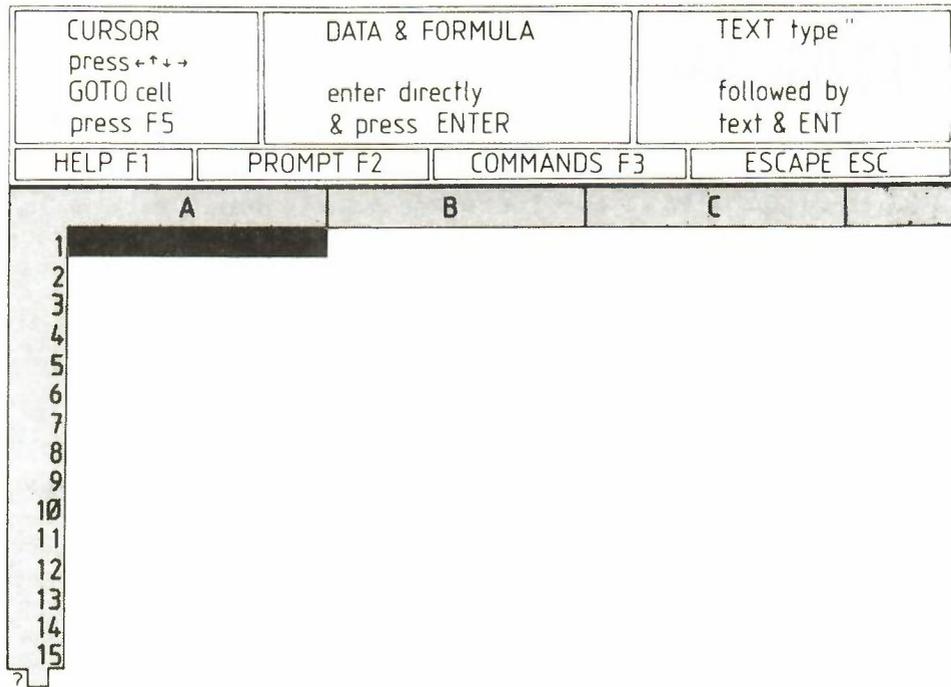


Figure 2.2 The Main Display (40 Character Mode).

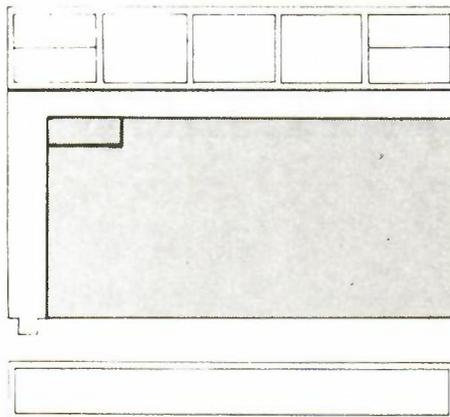


Figure 2.3 The Window.

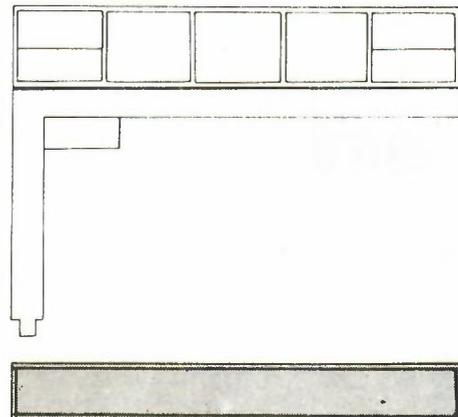


Figure 2.4 The Status Area.

At the top of the screen is a region, known as the control area. The contents of this area will be constantly changing so that it always shows the options you have, and gives confirmation of your choices.

At the moment it is divided into seven sections, showing the options you have at this, the main level. It shows that you may:

- press the Help key,
- turn off the prompts
- type in a number,
- move the cross wires,
- type in text,
- type in a formula,
- use a command,
- press the ESC key.

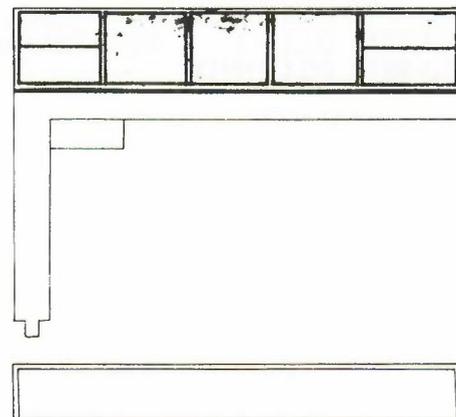


Figure 2.5 The Control Area.

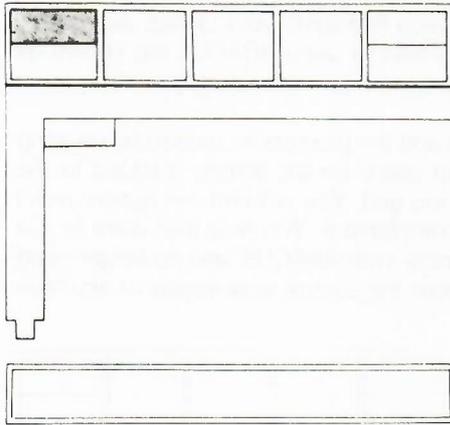


Figure 2.6 Help

The first option, displayed at the extreme left of the control area, shows that you may ask for Help by pressing function key 1 (**F1**). Regardless of any other changes in the control area display, the Help option will always be shown. This indicates that the Help facility is always available, no matter what you are doing.

Try pressing the Help key (**F1**) now. When you do, the grid display will disappear, to be replaced by one giving more details about the options open to you at the main level.

In addition, you will see that there is a list of topics about which more help is available, shown across the bottom of the display. You may ask for further information about any one of these topics by typing in its name and pressing **ENTER**. You do not need to type in the whole of a name; you only need to type in the first few letters - enough to distinguish it from any of the other topics in the list.

When, after typing in one of these names, you press **ENTER** you will find that further, more detailed, information is shown about the topic you have selected, and another list of sub-topics will be shown. You may then select one of these sub-topics by typing in the first few letters of its name, as described before.

You may continue this process until no further information is available. At any stage you may return to the previous screen by simply pressing **ENTER**. Repeatedly pressing **ENTER** will eventually take you back to the main display of your grid, with the control and status areas. At this point you will have left the Help facility and will have been returned to the exact situation before you pressed the Help key.

A faster way to return from Help is to press **ESC**. This will return you from any point within Help, back to the state from which it was first called.

Try using the Help facility to examine some of the many pathways through the information. Don't worry if you do not understand some of the information that is shown - it will make sense when you actually need to use it. All you need to do at the moment is to become familiar with the way in which the Help facility is used. When you have finished, press **ESC** to return to the main display.

It must be emphasised that Help is always available, at any time. Whenever you are not sure what you should be doing, just press the Help key, even if you are, for example, in the middle of typing in numbers or text as part of a command. You will not always start at the same point in Help, but will be presented with the information most relevant to what you were doing when you pressed the Help key.

When you have found the information you need and leave Help (by use of the **ESC** or **ENTER** keys) you will always be returned to the exact point from which you started, as though there had been no interruption.

Use the Help key as often as you like - it is there to assist you and will usually be the quickest and simplest way of solving your problems.

PROVISIONAL

2.4 THE PROMPTS

In addition to showing your options, the control area highlights your choice and, when necessary, suggests what you should do. These aids to using ABACUS are known as prompt messages or just *prompts*.

You can switch off the display of the control area and the prompts it contains by pressing function key 2 (F2). When you do this the extra space on the screen is added to the window, so that you can see a larger section of the grid. You will find this option useful for examining your application when you have completed it. You may also want to use this form of display when you have become familiar with ABACUS and no longer need the information in the control area. You can restore the control area display at any time by pressing F2 again.

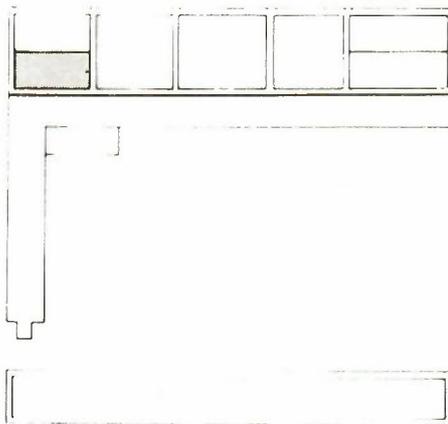


Figure 2.7 Prompts.

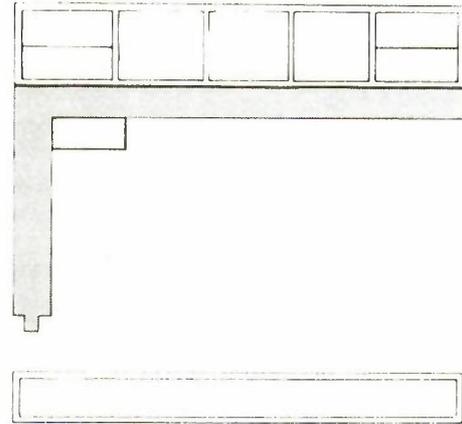


Figure 2.8 The Grid Labels.

2.5 THE CURSOR

At the top of the window, just below the control area, you will see a line in which a number of letters appear. These letters label vertical columns of cells making up the grid. As you can see, columns A,B,C, and so on are visible. Down the side of the window there is a series of numbers, from 1 to 15. These numbers label the rows of cells in the grid.

A combination of a letter and a number will therefore identify one particular cell, and is known as a cell reference. Such a reference is, for example, A1. This refers to the cell which is in column A and row 1, (the top left hand cell in the window). You will see that this cell is different from all the others in that it is filled by a large red rectangle. This is known as the cursor and it marks the current cell, that is the cell which will receive any data you type in.

The cell reference of the current cell is given by the CELL label in the status area at the bottom of the screen. In addition, the status area shows the extent of the used portion of the grid, the amount of memory left (as a percentage of the total) and the contents of the current cell. This cell is, of course, empty at the moment.

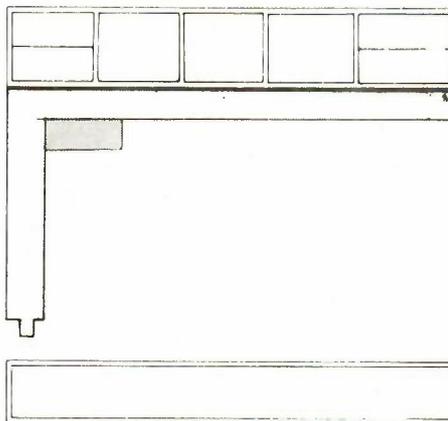


Figure 2.9 The Cursor.

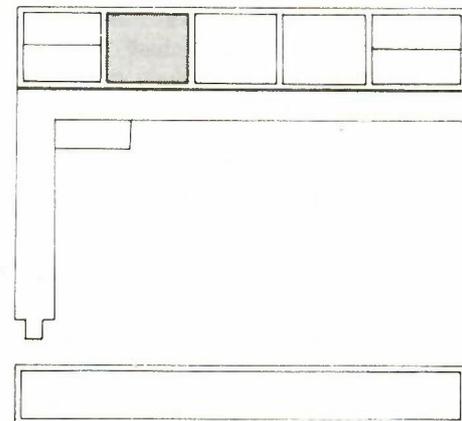


Figure 2.10 Moving the Cursor.

You can move the cursor by means of the cursor keys - try pressing the right cursor key once. You will see that the cursor moves one column to the right and the current cell indicator now shows B1. If you then press the left cursor key once the cursor returns to cell A1. Pressing the left cursor key again will have no effect because you are at the

PROVISIONAL

extreme left hand edge of the grid. Try using the four cursor keys to move the cursor around the window.

You may have noticed, in your experimenting with the cursor keys, what happens when the cursor reaches the right hand side, or the bottom, of the window. If you have not, try it now. Keep pressing the right cursor key until the cursor reaches the extreme right hand side of the window. When you press the right cursor key again the cursor will not appear to move, but you will see that the letters across the top of the window will change. When you attempt to make the cursor leave the visible area of the grid the window will move across the grid so that the cursor remains in view. The window is always adjusted automatically to keep the current cell in the visible part.

By now you will have realised that the cursor keys are a useful method of moving the cursor, provided you only wish to move it one or two cells. It is very inefficient for making large movements across the grid. For such large movements it is more convenient to go directly to the required cell. You can do this by pressing function key 5 (**F5**) and then typing the required cell reference.

Notice how the cursor movement option box in the control area is highlighted as soon as you press **F5**, to confirm your selection. Each of the options shown in the three central panels of the control area works in this way. As soon as you select the option the relevant panel is highlighted and remains so while you are typing your input. It gives reassurance that you have selected the option you want.

As an example of using the option to go to a particular cell, let us move the cursor to cell D11. First press **F5**. You will see that the words 'Go to cell,A1' appear below the window. ABACUS is suggesting that the cursor is to be moved to the top left hand corner of the grid. If you accept this suggestion (by just pressing **ENTER**) the cursor will move to that point. To move the cursor to another cell you should type in the cell reference - in this case you should type:

d11

and then press **ENTER**. The cell reference you type in replaces that suggested by ABACUS and the cursor moves directly to the cell you have specified.

You should now move the cursor back to the top left hand corner of the grid by using this option again. this time you can accept the suggested cell reference (A1) so all you have to type is:

F5 ENTER

You will find that you go back to the original state of the display, with the cursor at the top left hand corner of the window, in cell A1. Try using this command to move around the grid and finish by returning the cursor to cell A1.

Now move the cursor to cell Y1, by typing in **F5 Y1 ENTER** Look at the letters labelling the columns across the top of the window and you will find what happens if you go beyond column Z. The column to the right of column Z is labeled AA, the next one is labeled AB, and so on. This is obviously necessary for you to be able to refer to more than 26 columns.

Can you guess what happens to the column labels after column AZ? Move the cursor to cell AY1 to see if you were right. Continue moving the cursor to the right until you can go no further; you will then have found the label of the last column in the grid. How many columns are there in total? (You will find that the last column is labeled BL - there are 64 columns.) You can also move down the grid to find the last row but you will have to go a long way; there are 256 rows in the grid.

Return the cursor to cell A1 and then type:

100

but don't press **ENTER** just yet. Notice that the number or formula option box in the control area is now highlighted, to confirm your action. The number 100 will also have appeared in the line just below the window onto the grid, to the right of the question mark. (This is the same place that the cell references appeared when you were moving the cursor around the grid by use of **F5**.)

All typed input appears in this line, which is known as *the input line* (surprise!).

2.6 ENTERING NUMBERS

PROVISIONAL

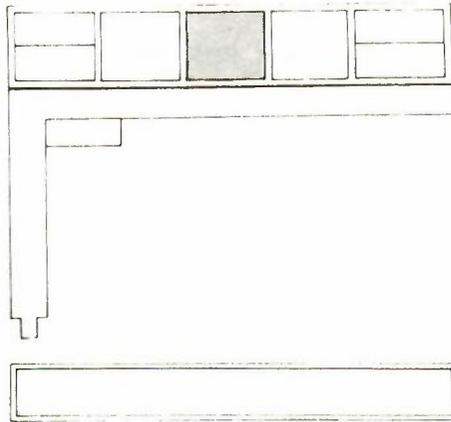


Figure 2.11 Numeric Entry

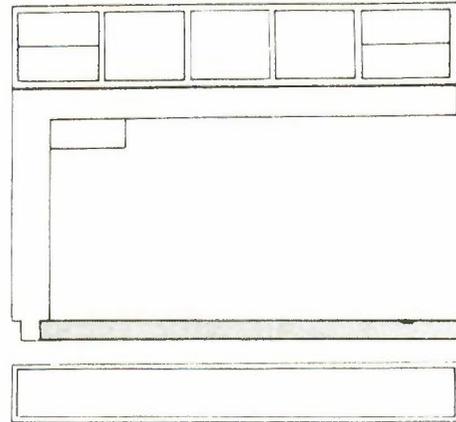


Figure 2.12 The Input Line

The small flashing rectangle in the input line marks where the next input character will appear, and is known as the input cursor, to distinguish it from the main cursor in the grid. If you make a mistake at any time during typing to the input line, you can correct it by holding down the shift key and then pressing the left cursor key to delete the input, character by character. The correct input can then be typed.

OK, you can press **ENTER** now! When you do, the value 100 will be transferred to the current cell (A1) and the input line will clear, ready for more input. You will see that the value 100 also appears in the status area, at the bottom of the display. Although this facility might seem rather pointless at the moment, you will find it useful when you start to use formulae, as described in Chapter 5. You may like, at this stage, to practise entering a few numbers at different points in the grid.

2.7 ENTERING TEXT

Once you are familiar with number entry, putting text into a cell is simple. You follow exactly the same procedure, except that you start by typing quotation marks (") into the input line. As soon as you type the quotation marks, ABACUS responds by emphasising the text entry option box in the control area. You then type in exactly what you want to appear in the cell, followed by pressing **ENTER**. There is no need for a closing quotation mark since ABACUS already knows you are typing in text. Try entering text into a few cells and, in particular, notice the difference between entering, say:

1000 ENTER (a number)

and

"1000 ENTER (a text string)

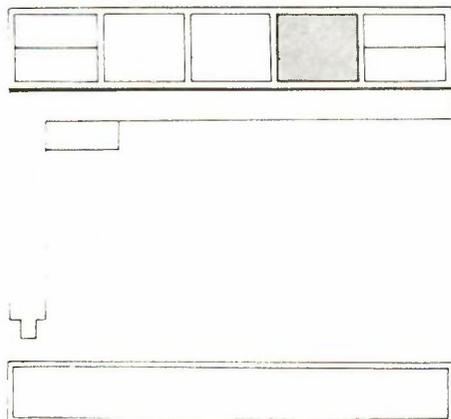


Figure 2.13 Text Entry.

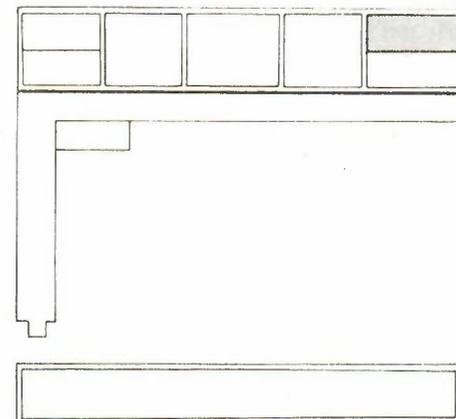


Figure 2.14 The Commands.

As indicated by the top right hand option in the control area, you can use a command by first pressing **F3**. This changes the central part of the control area to show a list, or menu, of the available commands. It is known as the *command menu* and is illustrated in Figure 2.15.

PROVISIONAL

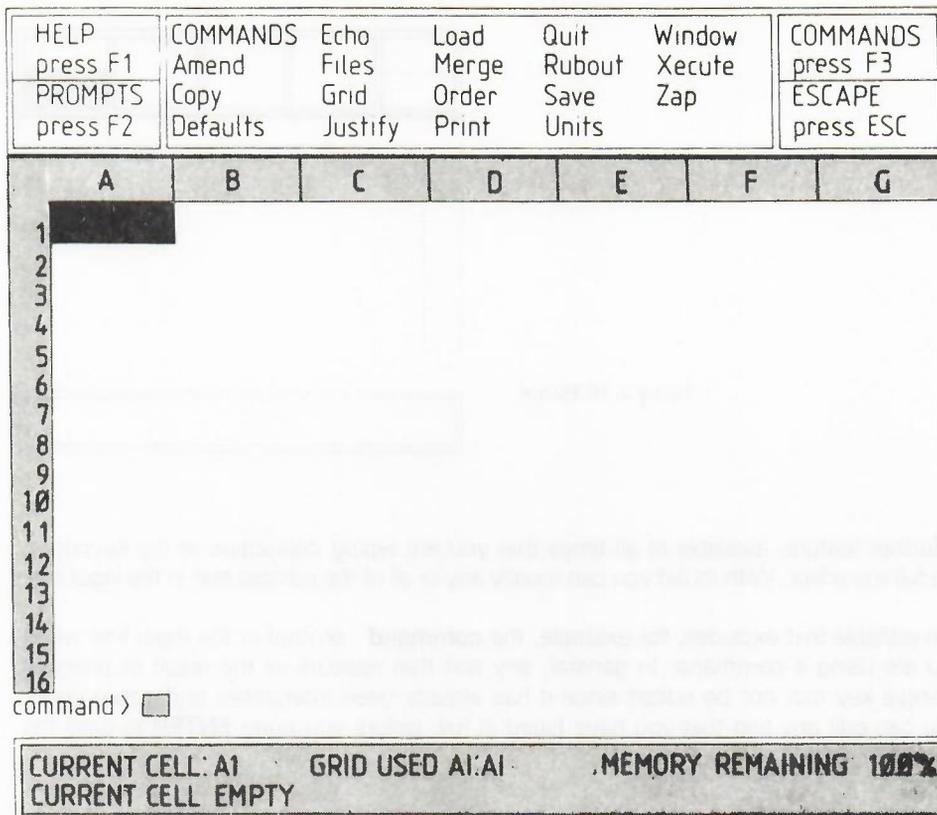


Figure 2.15 The Command Menu.

Most of the commands are described in later chapters but we can take a quick look at two of them. These are Zap, which you can use to clear the whole grid, and Quit, which allows you to leave ABACUS and return to BASIC.

Try the Zap command first. Press **F3** and locate the Zap command in the displayed menu. Press the Z key, when the word Zap will appear in the input line - you never need to type more than the first letter of any command because ABACUS does most of the work for you. Also, the command box in the control area changes to show the menu for Zap, telling you that you have two options. Try pressing **ESC** first, to return to the main level without any deletions. Since you can not recover a zapped grid, you will find this option useful if you call the command accidentally. (It is also worth remembering that you can get out of any command, at any stage, by pressing **ESC**.) Now return to the command menu by pressing **F3** again and then press Z to call the Zap command again, but this time press **ENTER** next, to clear the grid. You will be left with a blank grid and with the cursor in cell A1, ready to start afresh.

Whenever you want to leave ABACUS and return to BASIC, you must use the Quit command. This works in a similar way to Zap, in that you first press **F3** and then the first letter of the command (Q). Returning to BASIC causes you to lose the contents of your grid, so you are again given the option of going back to the command menu by pressing **ESC**. If you are sure you want to quit, press **ENTER**, to exit to BASIC.

You can generally use the **ESC** key to cancel the current action, or to leave a particular sequence of operations.

We have already seen how you can use the **ESC** key to leave Help, from any level (Section 2.4) and to leave the command menu (Section 2.10).

You can also use **ESC** to cancel input to the input line. Try typing in a number and, rather than pressing **ENTER**, press **ESC**. The entry is cancelled, as you would expect.

2.9 ESCAPE

PROVISIONAL

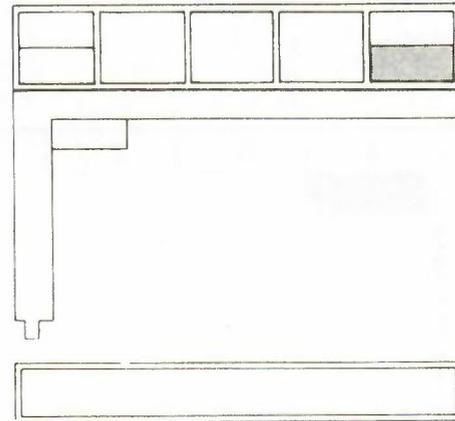


Figure 2.16 Escape.

2.10 THE LINE EDITOR

A further feature, available at all times that you are typing characters at the keyboard, is a full line editor. With its aid you can modify any or all of the editable text in the input line.

The editable text excludes, for example, the **command** prompt in the input line when you are using a command. In general, any text that appears as the result of pressing a single key can not be edited since it has already been interpreted and acted upon. You can edit any text that you have typed in full, before you press **ENTER** to pass the text to ABACUS.

Two situations where the line editor is particularly useful are when modifying a formula by means of the Amend command (see Section 6.7) and when you have made a mistake in a formula. If you make a mistake and type in a formula that gives an error you will be told the nature of the error. The text of the formula is shown in the input line, ready for correcting.

At all times each character that you type will be inserted to the left of the input line cursor position, and the cursor will move one space to the right. Regardless of the position of the cursor, all the text in the input line is accepted as input when you press **ENTER**.

The line editor uses the four cursor keys, together with the **CTRL** and **SHIFT** keys.

Left and Right Cursor Keys

The left and right cursor keys, used on their own, move the input line cursor by one character to the left or right.

If you press **SHIFT** and, while holding it down, press the left or right cursor key the input line cursor moves left or right by units of a word, that is to the next space or comma.

If you press **CTRL** and, while holding it down, press the left cursor key you will delete the character to the left of the cursor. Pressing **CTRL** together with the right cursor key deletes the character under the cursor. The following text closes up to fill the gap.

Up and Down Cursor Keys

If you press the up cursor key the cursor moves to the beginning of the editable text in the input line; the down cursor key moves the cursor to the end of the text.

Holding down the **CTRL** key and pressing the up cursor key will delete all editable text to the left of the cursor. Pressing **CTRL** and the down cursor key deletes all text to the right, including the character under the cursor.

2.11 MORE ABOUT NUMBERS AND TEXT

Here is some more information about how ABACUS handles the display of numbers and text in the grid. As well as going into some detail about the different methods of display, it uses several new commands and could be omitted at a first reading.

ABACUS stores all numbers to the full accuracy of 16 significant figures. This full accuracy might not, however, be shown in the grid cells. To illustrate this difference, put the number 123.4567 in cell A1. The value displayed in the cell and the exact value shown in the status area will agree.

PROVISIONAL

HELP press F1	UNITS Defines units or numerical Formats			COMMANDS press F3		
PROMPTS press F2	INTEGER	DECIMAL	EXPONENT	ESCAPE press ESC		
	MONETARY	PERCENT	GENERAL			
	Type the first letter of the option					

	A	B	C	D	E	F	G
1	123.46						
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							

command>Units,Decimals

CURRENT CELL: A1	GRID USED A1:A1	MEMORY REMAINING 98%
CURRENT CELL VALUE	123.4567	

Figure 2.17 The Units Command.

Now call the Units command (by pressing **F3** and then the U key). There are two main options; Cells or Defaults. You should press **ENTER** to select the (suggested) Cells option.

The display will look like that shown in Figure 2.17 and you are now offered the option of several different forms of display for numbers in the grid - let's try a few.

First, press the M key (to select the monetary form of display). You are then asked to specify the range of cells which are to be affected, and could reply by typing in a range reference (eg, A1:B3) or just the reference to a single cell. In this case we want to change only the current cell (A1). You should therefore type in A1 and press **ENTER**.

The display in cell A1 will change to 123.46, even though the actual value (123.456) is still kept, and shown in the status area. The monetary form of display always shows the number rounded to two decimal places and with a leading currency sign. (You can change the sign to \$, or anything else, by using one of the options in the Design command, as described in Section 4.4. After having made this change in the display you are returned to the main level of ABACUS.

Let us now change the display in cell A1 to integer (whole number) format, by calling the Cells option of the Units command again, but this time press the I key, and then A1 and **ENTER** (again we need only affect the current cell). The cell display now shows 123 - the decimal point and all figures following it are not shown in integer format.

We can now try decimal format. For this format you must also specify the number of figures you want to be displayed after the decimal point; let's use five decimal places. After calling the Cells option of Units, in the usual way, press **ENTER** (since Decimal is the default option) and then specify five decimal places by typing:

5 ENTER

Finally, in response to the 'range' prompt, type A1 again, to affect only the current cell. Cell A1 will now show 123.45670, as required.

Now use the command again, but this time press the P key, to specify the percent format. Use one decimal place and select the current cell (type: **1 ENTER A1 ENTER**). The display will now show 12345.7% - the percent option shows a number multiplied by 100 with an added % sign. Note that the stored value, as shown in the status area, is still 123.4567, regardless of the cell display.

We can now try the exponential format, with three decimal places, by typing:

F3 U ENTER E 3 ENTER A1 ENTER

Before you press **ENTER** for the third time the input line should contain:

Command Units,Cells,exponent,decimal places 3,range A1

and, after pressing it, the cell display will be 1.235 E+02. The exponential format is commonly used to display numbers which are too large or too small to be written in decimal format. The number is written as a value between 1 and 10, multiplied by the appropriate power of ten. The number 2 300 000 000, for example, can be written as 2.3 multiplied by 1 000 000 000. Now 1 000 000 000 is ten raised to the ninth power (nine tens multiplied together), so that 2 300 000 000 could be written in exponential format as 2.3 E+09. Very small numbers are written in exponential format by using negative powers of ten. Thus, the number 0.000123, which is 1.23 divided by 10000 (ten raised to the fourth power), can be written in exponential format as 1.23 E-04.

The remaining option in the Cells option of the Units command is the general format, which you can see in cell A1 by typing:

G A1 ENTER

This format again does not require you to specify the number of decimal places. Using the general option lets ABACUS choose a sensible format for the display of each number. It does the best it can to display each number in the space that is available.

Before we leave the Units command, try displaying the number in cell A1 in decimal format, with nine decimal places (type: **D 9 ENTER A1 ENTER**). Cell A1 now shows , since the required display will not fit in the space available. Whenever you see a cell filled like this you know that there is not enough room for the number to be displayed. You should then either use the Units command to change the display format, or increase the width of that column with the Grid command. These commands are used in many of the examples in Chapter 5.

Now clear the grid by using the Zap command and, with the cursor at cell A1, type:

"this is a long bit of text ENTER

Although the text is too long to be contained in one cell, it is all shown and overflows across the following cells. Now put the number 1 into cell B1. The text is cut off at the end of cell A1 as it is not allowed to overflow across another filled cell. Move the cursor back to cell A1 to verify that the whole of the text is still stored (see the status area) even if it is not displayed in full.

Move the cursor back to cell B1 and use the Rubout command to erase the contents of this cell. When you call this command (press **F3** and then **R**) you will be asked to specify the range of cells whose contents you want to delete. In this case we only want to delete the contents of the current cell (B1) and can do so by just pressing **ENTER**. Now that the following cell is empty, the full text appears in the grid again.

2.12 THE MICRODRIVES

Once you have loaded ABACUS, as described in Section 2.1, you should not remove the ABACUS cartridge from drive 1. It is, however, sensible to make at least one copy on another cartridge, remove it and replace it in its case in order to protect it. This removes any risk of accidentally losing or damaging your only copy of the program.

You can do this by using the Backup option of the Files command. For example, if you want to copy ABACUS from drive 1 to a blank, formatted cartridge in drive 2 you would type:

F3 F B MDV1__ABACUS ENTER MDV2__ABACUS ENTER

You can use the Save command to save, on a blank, formatted, cartridge in drive 2, any application that you have written (remember to specify drive 2 if it was not the last drive used).

Alternatively you can insert a cartridge containing previously saved applications. You can then load any of these applications with the Load command.

These two commands will Save and Load applications on the cartridge in the *current drive*. This is the drive that was last used and, when you have just loaded ABACUS as described in Section 2.1, this will be drive 1.

PROVISIONAL

To change the current drive to drive 2 you must include the drive specifier in the file name. (See Section 6.7 for a full description of Microdrive file names.)

You could, for example, load a file called "JACK__ABA" from drive 2 by using the Load command as:

F3 L MDV2__JACK__ABA ENTER

You do not need to include the extension. The Load command will assume that it is __ABA unless you type in something different. You could therefore just use:

F3 L MDV2__JACK ENTER

See the descriptions of the file-based commands (Load, Save, Print and Files) for the extensions assumed in each case.

Remember that the command used in the last two examples will make drive 2 the default drive so that you could now load another file, called "JILL__ABA" from drive 2 by:

F3 L JILL ENTER

Drive 2 will remain the default drive until you specify a different drive specifier in a file name.

Much of the power of ABACUS lies in its ability to handle whole rows, columns or ranges of cells in a single operation. This is done by the use of simple expressions which allow you, for example, to fill all or part of a row of cells. The values in the cells may all be made the same or they may vary in a regular way.

This chapter describes some of the properties of cells and the ways in which you can refer to them.

The cell is the basic unit for holding information in ABACUS. Each cell can contain one item of information which may be text, a number or a formula. Chapter 2 shows how you put either text or a number into a cell and Chapter 4 explains how you can use a formula.

For each cell that contains information, ABACUS also keeps a record of how that information is to be displayed. You can, for example, display numbers or text at the left, centre or right of the cell, and you can (as described in Section 2.11) display numbers in several different formats.

You can use the Justify command to change the position of the display within a cell. It allows you to select the position of numbers or of text independently.

Put a value of 100 in cell A1 and then try using the Justify command - press *f3* and then the J key. You will see that you are first asked to select between a Cells and a Defaults option. These two have different effects, as we shall see later. For the moment you should select the Cells option by pressing **ENTER**. You are then asked to choose between text (the default) or numbers and should select numbers, by pressing the N key. Next you must select from the options of Left, Centre or Right justification, with Right being the default. Choose Left justification by pressing the L key. Finally you are asked to specify the range of cells that are to be affected. In this case you should just type in A1 and then press **ENTER**. You will see that the value of 100 in cell A1 will move to the left hand side of the cell.

Try the other possibilities in the Cells option of the Justify command to see its effects on cells which contain text as well as those showing a numeric value.

The Cells option of the Units command (to modify the display format for numbers) is described in Section 2.11 and follows a similar pattern.

Note that you can change the numeric format or numeric justification of a cell which currently contains text, but nothing will appear to happen. If, however, you later change the contents of the cell from text to numeric you will see that the new format or justification has been stored. This also applies to a change of text justification for a cell which currently contains numeric information.

Cells that contain no information do not exist as far as ABACUS is concerned, and use no memory. They can therefore have no properties. If you attempt to use the Cells option of either the Justify or the Units command on an empty cell you will find that they have no effect.

Clear the grid with the Zap command and then use the Cells option of the Units command to change the display format of cell A1 to percent format with one decimal place. Now type the number 0.5 into cell A1. You will see that it is not displayed as a percentage (you would expect to see 50.0%) but is in general format.

The reason is that each time you put information into a previously empty cell, it is created (memory is reserved for it) with a set of *default* properties. When you have just loaded ABACUS these defaults are that text is justified left and numbers are justified right. All numbers are displayed in general format.

3.1 INTRODUCTION

3.2 CELLS

3.2.1 Properties of Cells

3.2.2 Empty Cells

If you want to change these defaults you must use the Defaults option of the Justify and Units commands. For example, use the Defaults option of the Units command, (press **F3**, U and D) to select a default of percent format with one decimal place. You will see that the choices are similar to those in the Cells option, but you are not asked for a cell range. The default is set for all empty cells.

Move the cursor to an empty cell and type in the number 0.5. You will see that it is now displayed as 50.0%.

The Defaults option of the Justify command works in the same way. Again you are not asked to type in a cell range because the new default will be used each time you put information into a previously empty cell.

Try using the Defaults options to make changes to the default justification for both numbers and text. See how they affect the numbers or text that you put into empty cells. Remember to restore the defaults to their original state; numbers justified right, text justified left and numbers displayed in general format.

3.3 ROWS

As a first example, let us fill the first row, from column B to column D, with the value 100. One way of doing this is to use the *range identifier*, row, as follows. Place the cursor in cell A1 and then type:

```
row = 100 ENTER
```

This means that the value 100 is to be placed in the cells of the current row (row 1), but so far we have not specified which columns are to be involved. As you will see, a prompt appears in the input line suggesting that the row be filled starting at column A (the column containing the cursor). The system will always make a reasonable suggestion for the starting point. If this is what you want you can accept it simply by pressing **ENTER**. In this case, however, we want to start at column B so you should press:

```
B ENTER
```

The input line changes to show that the filling of the row is to start at column B and a further prompt will appear with a suggestion of BL (the last column in the grid) for the end column. Again this will have to be changed, since we want to end at column D, so you should press:

```
D ENTER
```

The instruction is now complete and will be carried out - the value 100 will appear in cells B1 to D1 and the input line will clear, ready for your next input.

You must be careful to distinguish between **row**, used as a range identifier, and the function row(), which returns a row number (see Chapter 4).

You use the range identifier, **row**, to indicate that the data following the equals sign is to apply to the cells of the current row, rather than just to a single cell. Every time you use it, ABACUS will ask for the start and end points in the row, suggesting reasonable values based on your previous work. You can accept the suggestion by pressing **ENTER** or you can type in a replacement value, as described above.

3.4 COLUMNS

Filling a column follows a very similar pattern except, of course, that you refer to a column by one or two letters rather than the number that identifies a row. Suppose we want to put the text "hello" in the cells of column D, from row 5 to row 11. We can do this by using the second range identifier, col. Move the cursor to cell D5 and type:

```
col = "hello" ENTER
```

This time ABACUS suggests the correct starting point (row 5) as this row contains the cursor, and you can accept this suggestion by pressing **ENTER**. Row 256 will then be offered as a suggested end point and you should change this by typing:

```
11 ENTER
```

in exactly the same way as in the previous example. The text will then appear in cells D5 to D11 inclusive and the input line will clear, ready for the next input.

Again, you must be careful not to confuse **col** with the function col(), which returns a column number.

PROVISIONAL

As with **row**, each time you use **col** you will be asked to specify the first and last cell to be affected. You may, as usual, accept or replace the values suggested by ABACUS.

In addition to this way of using the range identifiers **row** and **col**, you can also use them as arguments to functions in place of range references (see Section 3.6). The following example combines both ways of using **row** and **col**.

```
col = sum(row)
```

It fills each cell of a column with the total of the values in the cells of the corresponding row.

Chapter 5 contains many examples of using them in both ways.

The previous examples referred to rows and columns by an explicit use of their number and letter cell references. An important alternative for identifying rows or columns is to use *labels*, that is names which you may choose yourself. These labels are then used to refer to specific rows, columns or cells.

3.5 LABELS

Any text that you put into a cell can be used as a label. You can use labels in any command or formula where you would otherwise use a letter and number reference. The advantage is that you can use meaningful and memorable names, removing the need to remember what row and column a particular cell is in.

This is an extremely powerful and flexible method which you can use to great advantage to simplify the setting out and operation of a grid. The following two sections explain how you can use these labels. The examples in Chapter 5 use this technique extensively but in this chapter we shall keep to simple situations.

A label may refer to either a row or a column, depending on the contents of the other cells in the grid. The basic rule when you use a label to identify a row, is that ABACUS searches along the row and column which intersect at the cell containing the label name. The first filled cell to be found below and to the right of the position of the label will determine whether the label should refer to a row or to a column. Figures 3.1 and 3.2 should help make this clear.

3.5.1 Row and Column Labels

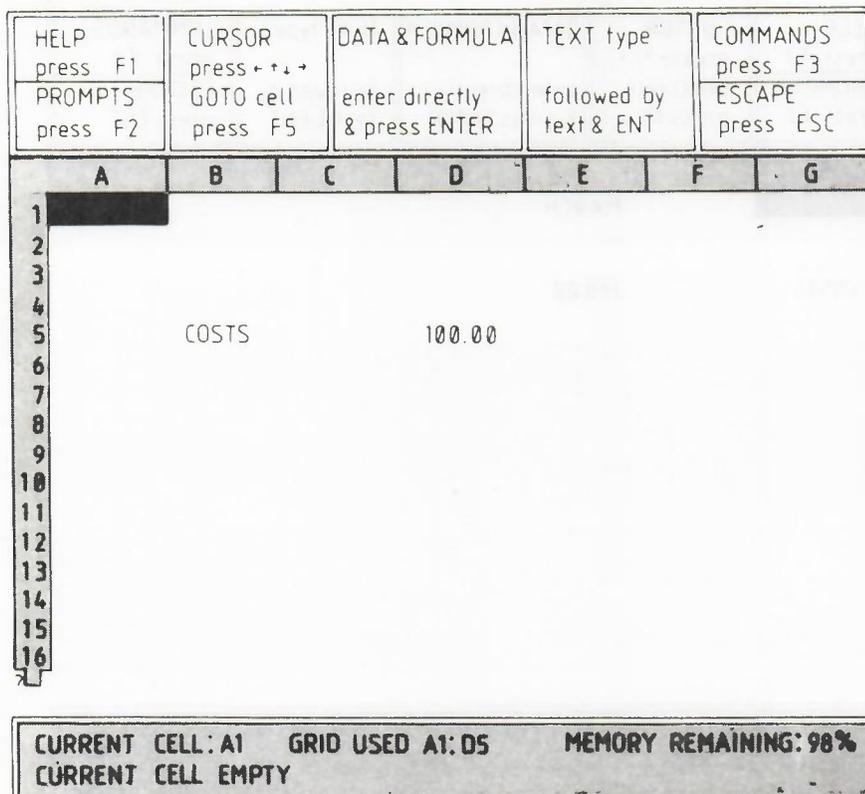


Figure 3.1 Labeling a Row.

In Figure 3.1 the label refers to a row and in the second example it refers to a column.

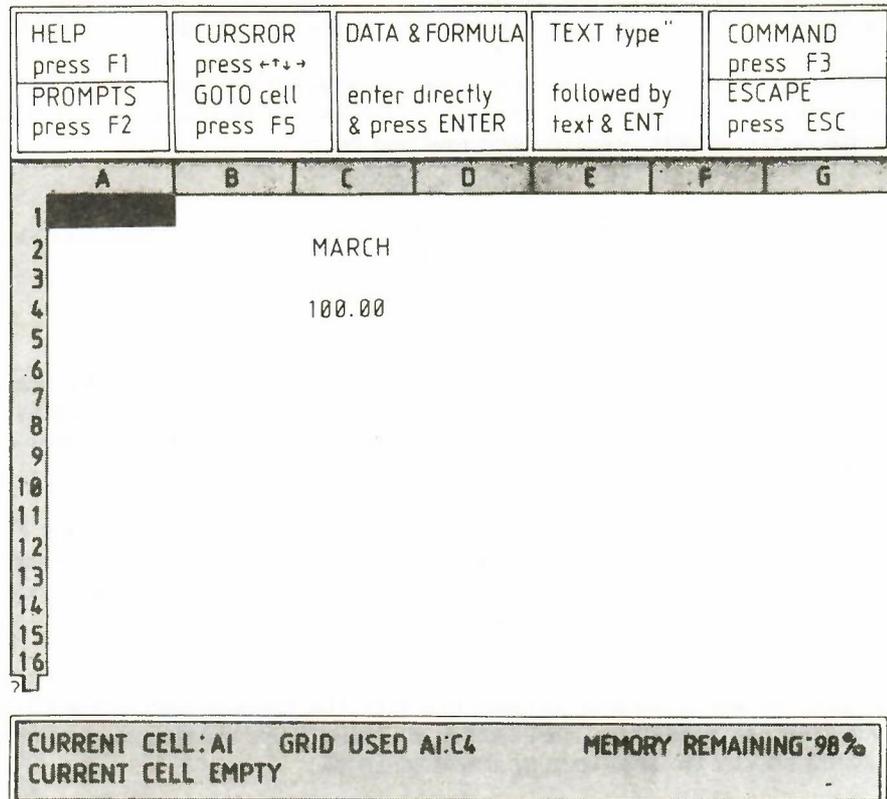


Figure 3.2 Labeling a Column.

In more complex cases other factors will be taken into account to resolve a row or column reference, and these are described in Section 6.2.

3.5.2 Labeling Cells

You can also use labels to refer to single cells, but in this case two labels are needed. In the following example the labels 'March' and 'Costs' can be used to refer to the shaded cell.

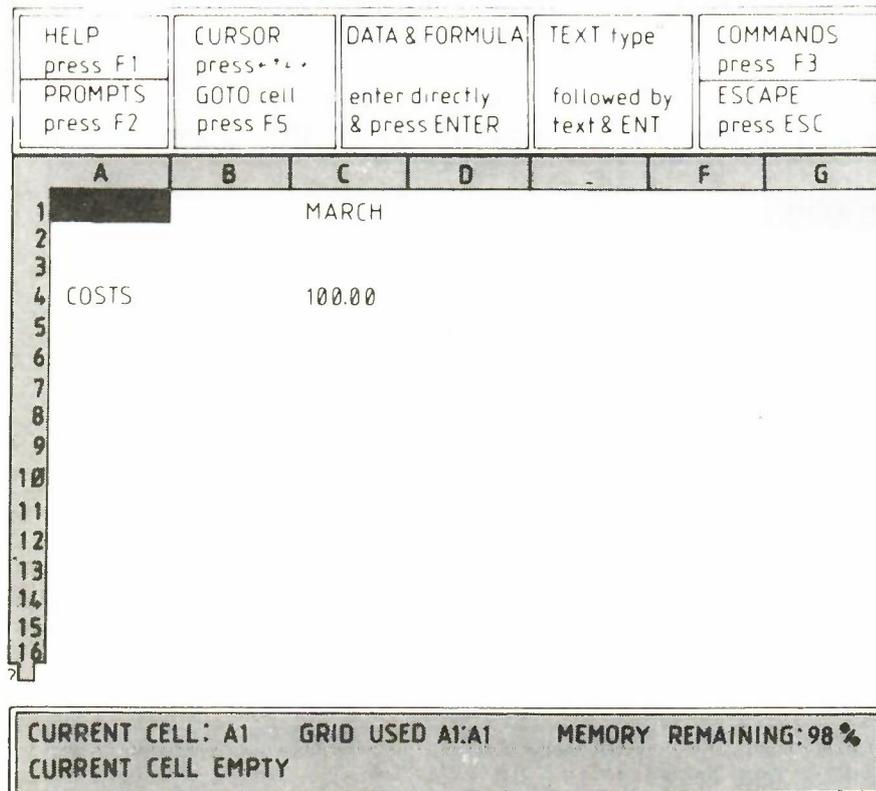


Figure 3.3 Labeling a Cell.

PROVISIONAL

The reference is made up of the names of the two labels, separated by a full stop (eg, March.Costs). It is not necessary to give the full names, and no distinction is made between upper and lower case letters. Only enough letters of each name are needed to make sure that the identification is unique. In the earlier case, 'mar.cos' would be perfectly adequate. The order of the labels is also irrelevant, so you could also use 'cos.mar' to refer to the same cell.

As the examples in Chapter 5 show, labels are powerful tools which result in enormous savings in time and effort during the creation or modification of an application. They are, however, only tools and, like all tools, need a little care in their use. You would not expect to be able to construct a chair by taking a few bits of wood, cutting them up in a haphazard fashion and then tacking them together as they they came to hand. Everyone knows that some sort of plan is required. In a similar way, you must plan your grid entries to make maximum use of the advantages of labels.

In addition to being able to refer to a whole row or a whole column, you can make an instruction work on a rectangular block, or range, of cells. Again, you can use labels to replace letter and number references.

A range reference is made up of two parts. The first part is the row and column reference of the top left hand cell of the range. This is separated by a colon from the second part, which is the row and column reference of the bottom right hand corner of the range. Range references will therefore appear as:

A2:D27

An example of the use of a range reference would be the use of the Copy command to copy the contents of a range of cells to a similar range at a different place in the grid. Figure 3.4 shows a range reference being used with the Copy command.

3.6 RANGES

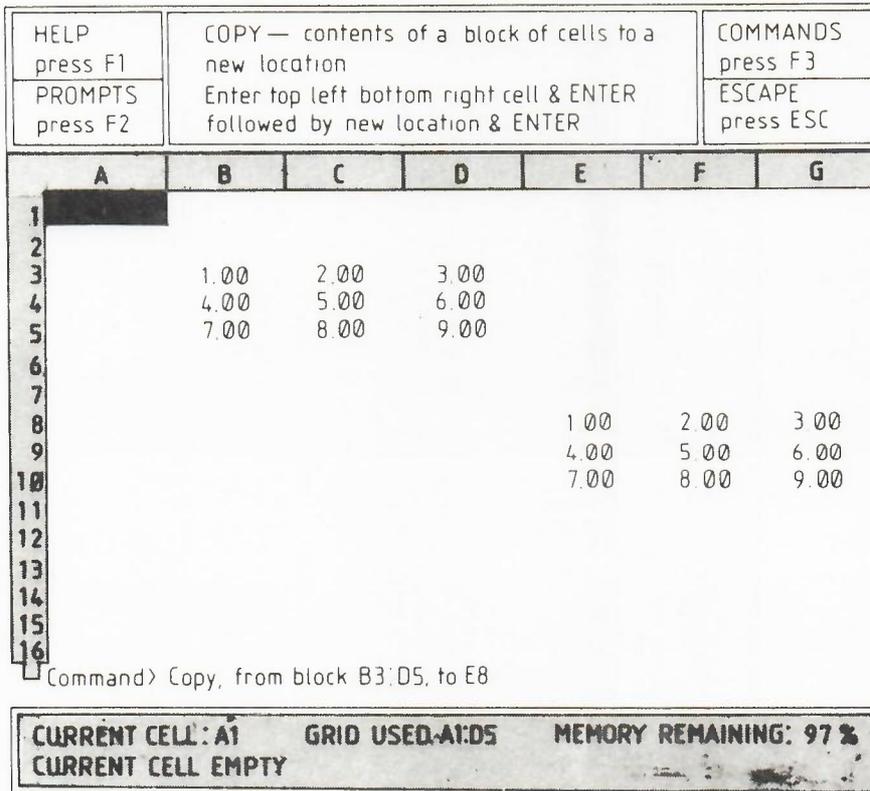


Figure 3.4 A Range Reference.

In all cases ABACUS will suggest starting and finishing points in a sensible way, depending on the current state of the grid. The starting point, for example, is normally chosen to lie in the current row or column, as appropriate. The end point is usually selected to be the same as the end point of the last similar operation. You may accept the suggestions or type in a replacement value.

)

)

)

)

ABACUS contains a number of pre-defined functions which are used to perform specific calculations on the contents of one or more cells. Each function takes a number of input values, known as arguments, and from them calculates a specific result.

4.1 FUNCTIONS

The arguments are placed in brackets after the name of the function and, if there is more than one, are separated by a comma. Most of the functions provided return a numeric value, for example, the function `sum()`. This takes, as an argument, a range reference and returns a value equal to the sum of the numeric values contained in all the cells within the range. Some functions, such as `month()`, return a text value (`month(1)`, for example, returns the text "January"). A few functions require no arguments, but the brackets are still needed. An example of such a function is `pi()` which returns the numerical value of the mathematical constant pi (approximately 3.14).

Two particularly useful functions are `col()` and `row()`. These return the number of the current column and row (containing the cursor) respectively. For example `col()` will return a value of 1 from column A, 2 from column B, and so on. The function `row()` simply returns the row number.

As an example we can use the two functions `month()` and `col()` to label columns of the grid. The object will be to place the headings January, February, and so on at the top of columns B to M. Type in

```
row = month( col() )
```

and select the column range from B to M. You will see that the result is not quite what we want in that, although the labels start at column B, the first label is February and not January. The reason for this is that, in column B, `col()` returns the value 2 and `month(2)` is the text "February". We can correct this mistake by making sure that the argument for the function `month()` is 1 when in column B, 2 when in column C, and so on. All we have to do is to alter the instruction so that 1 is subtracted from the value returned by `col()`, before calculating the month. Typing in

```
row = month(col()-1)
```

and selecting the column range from B to M will now give the correct result.

A formula is usually used to relate the contents of one cell to the contents of one or more of the other cells in the grid. The idea of formula is very important in the use of ABACUS as it allows you to describe even the most complicated calculations in a simple way. A formula is entered into a cell in exactly the same way as is used for numbers. Anything that is not recognised as a number (starting with a numeric digit) or a text value (starting with a quotation mark) is assumed to be a formula.

4.2 FORMULAE

Let us first try a very simple example. Move the cursor to cell B3 and enter the number 100, move the cursor to cell C3 and enter 200. Now move the cursor to cell D3 and type in the following formula

```
B3 + C3
```

When you press **ENTER** you will see two things happen. Firstly the value 300 will appear in cell D3; the result of the formula has been calculated by adding together the contents of cell B3 and cell C3 and the total has been placed in cell D3. In addition you will see that the status area at the bottom of the screen shows the formula used to calculate the value in this cell. When a cell contains a formula the actual formula will always be shown at the bottom of the screen, but the cell will show the result of the calculation.

The rest of the examples using formulae will make use of the labeling facility and the **row** and **col** range identifiers first described in Chapter 3. They allow much more efficient methods of entering information into the grid than the direct use of letter and number cell references, such as in the last example.

4.3 A SIMPLE CASH FLOW EXAMPLE

	A	B	C	D	E
1		January	February	March	April
2	Sales	£1000.00	£1050.00	£1102.50	£1157.62
3	Costs	£722.00	£749.50	£778.37	£808.69
4	Profit	£278.00	£300.50	£324.13	£348.93
5					

Figure 4.1 Simple Cash Flow Analysis.

You should start this example with a grid containing month headings in cells B1 to M1. If you have anything else in the grid you should clear it with the Zap command and enter the month headings, as described in Section 4.1.

Now move the cursor to cell A2, enter the text "Sales" and then put the value 1000 in cell B2. Now move the cursor to cell C2 and type in a formula in the form:

$$\text{row} = \text{sales.january} * 1.05$$

Accept the range selection given by ABACUS (column C to column M) by pressing ENTER twice. Note that ABACUS knows the end of the row is at column M because of your previous use of this end point, when you inserted the month headings. When you press ENTER a second time you will see a whole series of values appearing in row 2, from column C onwards, and the formula $B2 * 1.05$ will appear in the status area at the bottom of the screen.

If you move the cursor along row three you will see that the formula for each cell is slightly different. In each case the formula takes the contents of the cell on the immediate left and multiplies it by 1.05 to obtain the value to place in the current cell. This process was completely automatic. The system has remembered that the original definition of the formula was in 'sales.february' (cell C2) and referred to the contents of 'sales.january' (cell B2) one column to the left.

In ABACUS all formulae work in this way unless you specify otherwise. Each formula remembers the relative positions of all cells to which it refers. When such a formula is used in more than one cell the references are adjusted to maintain such a *relative cell reference*. The examples in Sections 5.4 and 5.8 explain how you can change this normal behaviour.

It may prove helpful to point out that the initial value of 1000 placed in cell B2 was necessary for two purposes; firstly to ensure that the label "Sales" was recognised as a row reference (see Section 3.4.1) and secondly to specify the first value to be used by the formula. The system now recognises that "Sales" should refer to a row.

Now position the cursor at cell A3 and enter the text "Costs". Without moving the cursor, type in the formula:

$$\text{costs} = \text{sales} * 0.55 + 172$$

This formula calculates the cost from two components. They can be regarded as manufacturing costs (55% of sales) and fixed costs totalling 172.00.

Use the suggested start and end points of column B and column M. Since the contents of the row is defined in terms of "Sales", the label "Costs" will also be taken as a row reference, with the same range as "Sales".

Again you should move the cursor along the row, examining the different formulae shown at the bottom of the screen, in order to understand how the results have been calculated.

Finally, put the text "Profit" in cell A5 and type in a further formula

$$\text{profit} = \text{sales} - \text{costs}$$

with the same range selection as before (i.e. columns B to M). The system will do all the rest of the work for you, producing a simple, but complete, example. If you now change the display to monetary format with the command (remember to press F3):

Units,Cells,Monetary,B2:M4

you should find that the first few columns appear as in Figure 4.1.

PROVISIONAL

4.4 AUTO-CALCULATION

When you have typed in the simple cash flow application described in the previous section, try changing the number in cell B2 (Sales.January).

Move the cursor to this cell - the easiest method is to press **F5** and then type in the cell reference (either B2 or sal.jan) followed by **ENTER**.

Now type in any number you like. When you press **ENTER** you will see that all the numbers in the grid will change!

The reason is that the values of all the formulae in the cells of the grid are recalculated automatically each time you make an entry to a cell. Since all the formulae in this example refer, directly or indirectly, to the value held in cell B2, all their values will change when you alter the contents of this cell.

This facility makes it very easy to use ABACUS as an aid to making management decisions. You can alter a value and see immediately what effect it has on the rest of the figures in the grid. Even in this simple cash flow example, you can see how changes in sales, manufacturing costs and fixed costs will affect the profits.

You can switch off the auto-calculate facility by means of one of the options in the Design command. This is useful, for example, when you have many complicated formulae in the grid and do not want to wait for a recalculation each time you change a single value. There is an example of this type of use in Section 5.10.

Try switching off the auto-calculate facility by pressing **F3** and then the D key, to call the Design command. The display changes to show a list of the options, as shown in Figure 4.2. You can select any one of these options by typing its first letter. Select the Auto-calculate option by pressing A. You will see that the auto-calculate state changes automatically.

HELP press F1	DESIGN Allows modification of options Press one of the opts. A,B,C,D,F,G,L,M,P,S	COMMANDS press F3
PROMPTS press F2	When finished press KEY X	ESCAPE press ESC

AUTO-CALCULATE on input ----- YES
 BLANK if zero ----- NO
 CALCULATION order row or columns ----- ROW
 DISPLAY 80,64,40, columns (8,6,4) ----- 80
 FORM feed between pages ----- YES
 GAPS between lines on printers ----- 0
 LINES per page of printer page ----- 66
 MONETARY sign £, \$, or other ----- £
 PRINTER paper width (characters) ----- 80
 STATIONARY continuous, single page ----- CONT

CURRENT CELL: A1	GRID USED A1:A1	MEMORY REMAINING: 100%
CURRENT CELL EMPTY		

Figure 4.2 The Design Command

This command is an exception in that you leave it by pressing X, instead of the more normal **ESC**. You are left in the main display.

If you now change the contents of cell B3 you will see that there is no change to the contents of any of the other cells. You can also force a recalculation of all the formulae in the grid at any time by using the Xecute command. While you have the auto-calculate turned off, try using this command. Make sure that the command menu is displayed

PROVISIONAL

in the control area (press **F3**) and then press the X key. The values-in-the cells of the grid will be recalculated, just as for a normal auto-calculate. The Xecute command returns you, as usual, from the command menu to the main display.

Before you go any further you should restore the auto-calculate facility by using the Design command again. Select the Auto-calculate option as before. Remember to leave the command by pressing X.

The following sections illustrate the use of ABACUS by developing a number of examples. In addition to explaining the way a number of features work, the examples have been chosen to show something of the wide range of applications to which ABACUS is suited. The best way of learning about ABACUS is to use it and the examples have been written with this in mind.

You are recommended to work through all the examples yourself, typing them in as you go along. Each contains some additional information, as well as giving more practice with the topics covered in earlier examples. You may well be able to think of modifications and improvements to the examples - they should give you ideas about how to construct applications of your own.

Above all, you are encouraged to experiment. You can not do any harm to either the computer or to ABACUS, and the more things you try out, the faster you will learn.

In all the examples the text, numbers and formulae are shown exactly as you would type them in. If a cell range is required it will be given in brackets at the end of the line. In many cases the range you need will be that suggested by ABACUS and you can select it simply by pressing **ENTER**. In other cases you will have to type in the range yourself, in the format requested by ABACUS. Where the cursor needs to be positioned on a particular cell, the cell reference will be given in square brackets at the beginning of the line. For example, the line

```
[A4] row = month(col()-1) (columns B to M)
```

should be read as:

```
move the cursor to cell A4 and type in  
row = month(col()-1)  
selecting the range from column B to column M.
```

Where you have to type in an explicit range reference, eg b3:e15, it will be given in that form.

When commands are given in full they are shown exactly as they will appear in the display. Remember that you only need to type in the first letter of each option and the rest is filled in by ABACUS. If you want to use the default option (the one suggested by ABACUS) you should just press **ENTER**, as indicated in the control area.

Each example assumes that you start with a completely blank grid. If necessary you should clear the grid with the Zap command before starting to type in the example.

This is a more complete version of the simple cash flow example of Chapter 4. When you have finished the grid it should look like Figure 5.1 (which shows only the first five columns).

The first two cell entries produce an underlined title for the grid,

```
[C1] "CASH FLOW  
[C2] rept("' = ",len(c1))
```

We shall use such a heading for each example. The second entry underlines the title with ' ' signs, to the exact length of the title. If you decide to change the title there is no need to alter the formula in cell B2 since it uses the len() function to read the length of the text in cell C1.

```
[A4] row = month(col()-1) (columns B to M)  
[A5] row = rept("'-' ,width() + 1) (columns A to M)
```

These row entries produce month headings, as described in Chapter 4, and rule a line across the whole of the used part of the grid. The function width() returns the width, in character spaces, of each column. It can therefore be used to rule lines across a grid with columns of different widths. There is one extra character space separating each column of the grid, which is why the additional + 1 is needed.

PROVISIONAL

	A	B	C	D	E
1					
2					
3					
4		January	February	March	April
5					
6					
7	SALES	4000.00	4080.00	4161.60	4244.83
8	COST OF SALES	2750.00	2790.00	2830.80	2872.42
9					
10	GROSS PROFIT	1250.00	1290.00	1330.80	1372.42
11	EXPENSES				
12	wages	700.00	700.00	700.00	700.00
13	advertising	100.00	100.00	100.00	100.00
14	rent	200.00	200.00	200.00	200.00
15	electricity	50.00	50.00	50.00	50.00
16	depreciation	90.00	90.00	90.00	90.00
17					
18	TOTAL EXPENSES	1140.00	1140.00	1140.00	1140.00
19					
20	PROFIT	110.00	150.00	190.80	232.42
21					

Figure 5.1 The Completed Cash Flow Grid. (first five columns)

|A6| "SALES
 |B6| 4000
 |C6| row = sal.jan*1.02 (columns C to M)

These entries fill in the sales figures for the year, assuming that the January sales were 4000 and that sales are increasing at 2% per month.

|A7| "COST OF SALES
 cos = sal*0.5 + 750 (columns B to M)

(The costs are assumed to be half of the selling price plus a fixed amount of 750.00.)

|A8| row = a5 (columns A to M)
 |A9| "GROSS PROFIT
 gro = sal-cos (columns B to M)

This rules off the grid again and calculates the monthly gross profit figures.

|A11| "EXPENSES
 |A12| "wages
 row = 700 (columns B to M)
 |A13| "advertising
 row = 100 (columns B to M)
 |A14| "rent
 row = 200 (columns B to M)
 |A15| "electricity
 row = 50 (columns B to M)
 |A16| "depreciation
 row = 90 (columns B to M)

These entries fill in the expense figures, assuming them to be constant throughout the year. You can, of course, change the expense headings and amounts to suit yourself. You can include more or fewer entries, as long as you make the necessary changes to the cell references in the rest of the example. You may want to have different values for each month, but it is faster to set up the table with fixed values and modify them later. A simple way to change any value is given at the end of the example.

|A17| row = a5 (columns A to M)
 |A18| "TOTAL EXPENSES
 |B18| row = sum(col) (rows 12 to 16, columns B to M)
 |A19| row = a5 (columns A to M)

You now have the totals of the monthly expenses.

PROVISIONAL

The sum() function adds the contents of all the numeric cells in the range specified as its argument. All empty cells, together with those containing text, are ignored. The range could be given as an explicit range reference - B12:B16 for example. In this case, however, each range is only a single column so we have used the range specifier 'col'. All you need to do is to answer the range questions asked by ABACUS, just pressing **ENTER** if the suggested range is what you want.

Note that this formula uses the range identifiers row and col in the two different ways that were mentioned in Chapter 3. Firstly, row is used to indicate that the formula is to be placed in several cells of the current row. Secondly, col is used to specify the range of cells over which the addition should take place. Both of the range identifiers need you to confirm (or change) their beginning and end points. In this case ABACUS deals with the range for the sum() function first.

```
|A20| "NET PROFIT
net=gross-tot (columns B to M)
|A21| row= rept("=",width()+1) (columns A to M)
```

The table is now complete, with the net profit figures calculated as the difference between the gross profits and the total expenses. All that you have to do now is to adjust the appearance of the table by using a few commands. Remember to press **F3** each time you want to use a command.

```
grid width, 15, FROM a TO a
```

Note that the Grid command has its own menu of options. At the conclusion of one of these options the command returns to its own menu. You must press **ESC** to return to the main display level. (The Files command works in a similar way.)

```
Justify, Cells, Text, Right, a4:m4
Justify, Cells, Text, Right, a12:a16
Units, Cells, Decimal, Decimal places 2, a1:m21
```

We have chosen to display the figures in decimal format, with two decimal places. If you prefer the pound sign to appear you should replace the last command by

```
Units, Cells, Monetary, a1:m21
```

It is very simple to alter any of the figures. Suppose you want to increase the February advertising figure. All you have to do is press **F5** (go to a cell) and type the cell reference

```
feb.adv
```

The cursor will move to that cell and you can type a new value.

Remember that the sales figures were calculated by a formula which assumed a 2% increase each month. If you change one of these cells to a numeric value you will destroy the formula in that cell. The formulae in the other cells of the row will, however, be unchanged. The amounts in the following cells will still increase by 2% per month, starting from the new value.

This is a 'quickie' to produce a simple graphic representation of a set of figures in bar chart form. You have seen most of the formulae before, so you will not need too many comments.

Figure 5.2 shows the appearance of the chart with a few numbers added.

Remember to clear the grid with the Zap command before starting to type it in.

```
|C1| "SIMPLE BAR CHART
|C2| rept("=",len(c1))
|A4| "Values
|B4| col="!" (rows 4 to 15)
|A5| row=rept("=",width()+1) (columns A to F)
|C6| col=rept("*",a6) (rows 6 to 15)
```

These cells in column C are set to display a row of asterisks. The number of asterisks shown in each cell is governed by the value in the cell in column A of the same row. The cell reference (a6) in this formula is a relative one. Look at the formulae in the cells of row C. The cell reference in each one is different - it has been adjusted to refer to the correct row of column A. ABACUS treats all cell references like this unless you specify otherwise. How to use different types of cell references is described in Section 5.4.

5.3 A SIMPLE BAR CHART

	A	B	C	D	E	F
1						
2						
3						
4	Values					
5						
6			10		*****	
7			17.35		*****	
8			17		*****	
9			3		***	
10			27		*****	
11			38		*****	
12			0			
13			15		*****	
14			23		*****	
15			8		*****	

Figure 5.2 A Simple Bar Chart.

We might as well reduce the width of column B to one character, since it is only dividing the values from their display. Press **F3** and use the command:

```
grid width, 1, FROM b TO b
```

To use this bar chart, all you have to do is to put numbers in the cells of column A, between A6 and A15 inclusive. You will find that the rept() function doesn't like negative numbers, but anything else will be accepted. Of course, if you use too large a number you won't see the end of the bar, as it will disappear out of the end of the window!

5.4 MULTIPLICATION TABLES

This simple example may prove useful to a child who wants to learn the multiplication tables. It allows you to request a particular table and then displays it.

The table in Figure 5.3 shows an example of the display it produces.

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

Figure 5.3 A Multiplication table.

Firstly we title the application as normal.

```
|B1| "MULTIPLICATION TABLES
|B2| rept("=",len(b1))
```

The next three lines give a heading to the table.

```
|B3| "The
|C3| askn("Which multiplication table do you want? ")
|D3| "times table
```

Here we have used the askn() function to request input; it allows you to choose which table you want, by typing in a number.

PROVISIONAL

This function takes a text string as its argument and displays the text in the input line. It then waits for you to type in a number, followed by **ENTER**. The number that you type in will be displayed in the cell which contains `askn()`.

Note that `askn()` will not wait for input during a normal auto-calculation of the grid. It will only display the message and request input when you force a recalculation of the grid by using the `Xecute` command.

While you are constructing the application the cell containing the `askn()` function will show zero. Once you have input a value to the cell it will be retained until the next time you force a recalculation with the `Xecute` command. The remaining grid entries use the column-filling facility to produce the body of the multiplication table.

```
|B4| col=str(row()-3,2,0)+"*" (rows 4 to 15)
```

This is the most complicated formula of the example. It is used to display the multiplier in each row of the table. The number is converted to a text string so that we can combine it with the multiplication sign and display them both in a single cell.

The `str()` function performs the conversion of a number to the equivalent string of digits. It takes three values; the number to be converted, a code for the format (integer, exponential, general, etc) in which the number is to be displayed, and the number of decimal places to be shown. The codes for the different formats are given in the entry for `str()` in Section 6.9 of the reference chapter.

In this case the value is obtained from the expression `'row()-3'`, whose value is 1 in row four, 2 in row five, and so on, up to 12 in row 15. The next value (2) selects display as an integer (whole number). The third number normally specifies how many decimal places are to be used. Its value must always be given but is ignored for integers which, by definition, have no fractional part. It has been given a value of zero (any other value could have been used).

Finally the result is concatenated with the string `"*"`, so that both the multiplier and the multiplication sign are displayed in a single column.

```
|C4| col=$c3 (rows 4 to 15)
```

Column C contains copies of the value typed in in answer to the `askn()` function. This is the second number of the product in each row of the table. The cell reference is preceded by a `$` sign to make it an *absolute cell reference*. When you have entered the formula, look at the contents of the cells of column C. You will see that they all contain the cell reference `$C3`. The reference has not been adjusted in each row, unlike the example in Section 4.3. You can make any cell reference absolute by adding a leading `$` sign.

```
|D4| col="=" (rows 4 to 15)
```

```
|E4| col=$c3*(row()-3) (rows 4 to 15)
```

These last two column entries are almost self-explanatory. They are used to produce the equals sign and the answer for each row of the table. The last formula multiplies the value from the `askn()` function (another absolute cell reference) by the `row()-3` expression which, as we saw earlier, gives numbers from one to twelve in successive rows.

We now need to use a few commands to change the display of the table to a more convenient form. Use the following commands:

```
Justify,Cells,Text,Right,b3:b15
Justify,Cells,Text,Right,d4:d15
Justify,Cells,Numbers,Centre,c3
grid width, 5 FROM b TO b
grid width, 3 FROM c TO c
grid width, 2 FROM d TO d
grid width, 4 FROM e TO e
```

Remember to press **ESC** to leave the Grid command.

You use the table by forcing a recalculation of the grid with the `Xecute` command. You are prompted for input - the text of the `askn()` function will appear in the input line - and should type in a number between one and twelve.

5.5 CHEQUE BOOK RECONCILIATION

This example allows you to keep a check on your bank account. You enter details of your cheques in the spaces provided. At the end of the month you add the details of your salary, standing orders etc, by use of the Xecute command. You are then provided with a balance which you can compare with your bank statements.

The result, with a few figures added, is shown in Figure 5.4.

	A	B	C	D
1		<u><u>CHEQUE BOOK RECONCILIATION</u></u>		
2				
3				
4			Month	January
5				
6	Opening balance		200.00	
7	Salary		527.35	
8	Miscellaneous income		0.00	
9				
10	CREDIT		<u>727.35</u>	
11				
12				
13	Standing orders			130.00
14	Charges			0.00
15				
16	Cheques	Date	Cheque no	Amount
17		3/01/84	123456	50.00
18		10/01/84	123457	50.00
19		14/01/84	123458	32.21
20		17/01/84	123459	50.00
21		24/01/84	123460	50.00
22		31/01/84	123461	50.00
23		---	---	---
24		---	---	---
25		---	---	---
26		---	---	---
27				
28	DEBIT			<u>412.21</u>
29				
30	Closing balance		<u>315.14</u>	
31				

Figure 5.4 Cheque Book Reconciliation.

```

[B1] "CHEQUE BOOK RECONCILIATION
[B2] rept("=",len(b1))

[C4] "Month
[D4] askt("Enter month? ")

```

The askt() function works in a similar way to askn(), but the expected input is a text string. When you use Xecute you should type in the name of the month for your balance.

```

[A6] "Opening balance
[A7] "Salary
[A8] "Miscellaneous income

[C6] col = askn(a6 + " for " + $d4 + "? ") (rows 6 to 8)

```

The prompt string for askn() is constructed from the text of other cell entries, using both relative and absolute cell references.

```

[B10] "CREDIT
[C10] sum(col) (rows 6 to 8)

```

Cell C10 is used to contain the total of all credits for the month. This cell is labeled; its reference is 'credit.month'.

Its contents are calculated using the sum() function which we first met in Section 5.2. It adds the numeric contents of all cells in the range specified by its argument. Remember that it ignores any cell in the range that is empty or that contains text.

PROVISIONAL

In this case we have again used it as `sum(col)`, which specifies that the cells to be summed lie in the current column. As normal, ABACUS asks you to specify the exact range, suggesting reasonable values based on your previous work.

```
|C11| rept("=",len(str(credit.month,0,2)))
```

Cell C11 underlines the total, using the usual `rept()` and `len()` functions. In this case, however, we do not know in advance the number of characters to underline. We therefore have to convert the number to a string of characters with the `str()` function, assuming that it is to be shown in decimal format with two decimal places. The length of this string gives the correct number of characters to underline.

```
|A13| "Standing orders
|A14| "Charges
|A13| col=askn(a13+" for "+$d4+"? ")
```

These allow you to enter the monthly debits in response to prompts, using the same method as described earlier.

```
|A16| "CHEQUES
|B16| "Date
|C16| "Cheque no
|D16| "Amount
|B17| row="----" (columns B to D)
```

These cells set up an area of the grid which you will later use to enter the details of your cheques.

```
|B28| "DEBIT
|D28| sum(col) (rows 13 to 26)
```

This calculates the total of the debits. Remember that `sum()` only adds numeric values in the cells of the specified range. Cells containing text, and empty cells, are not included. The sum will therefore ignore all unused entries in the list of cheques, as well as the table heading in column D.

```
|A30| "Closing balance
|C30| credit.month - debit.amount
```

The calculation of the closing balance completes the grid entries. You should now use the commands to tidy up the appearance of the application.

First we can use the Echo command to fill the rest of the cheque table and complete the underlining of the totals.

```
echo,cell b17, range b18:d26
echo,cell c11, range d29:d29
echo,cell c11, range c31:c31
```

Next we need to set the numeric display to decimal, with two decimal places, for the whole of the application, with integer format for the cheque numbers:

```
Units,Cells,Decimal,Decimal places 2,a1:d30
Units,Cells,Integer,c17:c26
```

Section 3.2 explained that empty cells do not exist as far as ABACUS is concerned and they do not take up any space in memory. The change to decimal format (or to any format other than the default of general format) will therefore only affect non-empty cells. This is why we filled the cheque table with "----" before making the change to decimal format.

An alternative method is to change the default format, as described in Section 3.2.

Finally we can modify the justification of the text, including the underlining, to improve the final appearance.

```
Justify,Cells,Text,Right,b16:d26
Justify,Cells,Text,Right,c11
Justify,Cells,Text,Right,d29
Justify,Cells,Text,Right,c31
```

The part of the grid that is used is too large for it all to be visible in the window at once. In order to see the final results, together with the values entered via the askt() and askn() functions, you might like to use the split window facility. A vertical split is most suitable for this grid and you can set it up by moving the cursor to the centre of the window and then using the command:

Window,Vertical,Separate

The split will occur at the position of the cell containing the cursor so that you can divide the screen in any proportion that you want. Then set the top left corner of the left hand window to cell A1, and the right hand window to cell B15 for the best effect. (f4 switches the cursor between the two windows.)

5.6 STANDARD DEVIATION

This example calculates the mean and standard deviation of a set of numbers. It makes use of the labelling facilities of ABACUS so that the formulae used in the calculations are mostly self-explanatory.

	A	B	C	D	E
1			<u>STANDARD DEVIATION</u>		
2					
3					
4		Value	Deviation	Square of dev.	
5		5.00	-4.50	20.25	
6		6.00	-3.50	12.25	
7		7.00	-2.50	6.25	
8		8.00	-1.50	2.25	
9		9.00	-0.50	0.25	
10		10.00	0.50	0.25	
11		11.00	1.50	2.25	
12		12.00	2.50	6.25	
13		13.00	3.50	12.25	
14		14.00	4.50	20.25	
15					
16	Mean	9.50	Variance	8.25	
17			Std. Dev.	<u>2.87</u>	
18					

Figure 5.5 Standard Deviation Calculation.

In addition it uses a grid layout which requires calculation in column order, rather than the normal row order. In general, a formula should only refer to cells that are in the region above and to the left of the cell containing a formula (including the row and column containing the formula - it is perfectly valid for the formula to contain a reference to its own cell).

If you do not follow this rule, as in this example, it is likely that the results may be incorrect. In most cases you can obtain a correct result by forcing a recalculation of the grid with the Xecute command or, as in this case, calculating the grid in column order.

```
|B1| "STANDARD DEVIATION
|B2| rept("=",len(b1))
|B4| "Value
|C4| "Deviation
|D4| "Square of dev.
|B5| col=row() (rows 5 to 14)
```

This last formula inserts a set of dummy values in the cells of column B for testing the application. When the grid entries are complete you can replace them with any other values you like. The table described in this example will only hold ten values - you can change this to cope with more if you want.

```
|A16| "Mean
|B16| ave(value) (rows 5 to 14)
```

deviation = value-\$mean.value (rows 5 to 14)
square = dev*dev (rows 5 to 14)

```
|C16| "Variance
|D16| ave(square) (rows 5 to 14)
```

These formulae show that the variance of a set of numbers is defined as the average of the squares of the deviations from the mean,

PROVISIONAL

```
|C17| "Std. Dev.  
|D17| sqr(variance)
```

and that the standard deviation can be calculated as the square root of the variance.

```
|D18| rept("-",len(str(std.sq,3,0)))
```

The numbers in this example are left in general format so that it can handle any range of values. The underlining therefore uses the length of the text string corresponding to the number in the cell above (with cell reference 'std.sq') expressed in general format.

You can improve the appearance of the display by changing to centre justification for the text in the range B4:D4, and using left justified numbers in the range B16:D17.

If you try using this example by putting different values in the cells of column B, you will find that it does not give the correct answers. The reason is that the recalculation of the grid is performed row by row, from the top downwards. Any alteration you make will therefore be worked out on the basis of an incorrect mean value (since the new mean will not be calculated until after the deviations from the mean). The solution is to make the recalculation of the grid to be in column order, from left to right. You do this by use of the Design command.

Try it now, using the 'C' option to change to column order. Leave the command by pressing the X key, as indicated in the control area. When you next change a value in column B, the calculation will be correct, since the new mean is now calculated before the deviations. Although this ability to change the order of calculation is very useful, you should not get into the habit of using it too often - calculating in column order is much slower than when using row order.

If you save a grid to a Microdrive file, the current settings of all the default options (made with the Design command) are saved with it. They are used to set the defaults whenever you reload the file, so you do not have to change them yourself each time you use it. All you need to do is to set the default options to the values you want before saving your application with the Save command.

This example will allow you to plan your household expenditure over the year. You can enter your estimated expenditure under a number of headings for each quarter. You are then provided with quarterly totals, your expenditure for the whole year and the averaged monthly cost.

5.7 A HOUSEHOLD BUDGET

```
|D1| "HOUSEHOLD BUDGET  
|D2| rept("=",len(d1))
```

Now we can set up the structure of the table with its ruled divisions.

```
|A4| row=rept("-",width()+1) (columns A to K)  
|A5| col="!" (rows 5 to 20)
```

The following commands complete the table structure.

```
grid width, 16 FROM b TO b  
grid width, 8 FROM d TO j  
grid width, 1 FROM a TO a  
grid width, 1 FROM c TO c  
grid width, 1 FROM e TO e  
grid width, 1 FROM g TO g  
grid width, 1 FROM i TO i  
grid width, 1 FROM k TO k
```

Remember to press ESC to leave the Grid command.

```
echo,cell a5, range c5:c22  
echo,cell a5, range e6:e22  
echo,cell a5, range g6:g22  
echo,cell a5, range i6:i22  
echo,cell a5, range k5:k22  
echo,cell a4, range b7:j7  
echo,cell a4, range b21:k21  
echo,cell a4, range c23:k23
```

	A	B	C	D	E	F	G	H	I	J	K	
	<u>HOUSEHOLD BUDGET</u>											
			ESTIMATED EXPENDITURE									
		Item	Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec						
8		Mortgage/Rent	400.00	400.00	400.00	400.00						
9		Rates		450.00								
10		Gas	150.00	80.00	60.00	150.00						
11		Electricity	40.00	30.00	30.00	40.00						
12		Water rates		35.00		35.00						
13		Telephone	150.00	150.00	150.00	150.00						
14		Insurance										
15		Clothing										
16		Hire-purchase										
17		Car tax										
18		Petrol										
19		TV licence										
20		Savings										
22		Quarterly tots	740.00	1145.00	640.00	775.00						
25			Yearly	Monthly								
27			Payments	<u>£3300.00</u>	<u>£275.00</u>							

Figure 5.6 Home Budget Example

Fortunately this takes much less time to type in than it takes to describe!

```

|A7| ""-
|F5| ""ESTIMATED EXPENDITURE
|B6| ""Item
|D6| ""Jan-Mar
|F6| ""Apr-Jun
|H6| ""Jul-Sep
|J6| ""Oct-Dec

|B8| ""Mortgage/Rent
|B9| ""Rates
|B10| ""Gas
|B11| ""Electricity
|B12| ""Water rates
|B13| ""Telephone
|B14| ""Insurance
|B15| ""Clothing
|B16| ""Hire purchase
|B17| ""Car tax
|B18| ""Petrol
|B19| ""T.V. licence
|B20| ""Savings

|B22| ""Quarterly tots

|B22| sum(col) (rows 8 to 20)
|D22| sum(col) (rows 8 to 20)
|H22| sum(col) (rows 8 to 20)
|J22| sum(col) (rows 8 to 20)

|D25| ""Yearly
|F25| ""Monthly
|B27| ""Payments

|D27| sum(d22:j22)
|F27| year.pay/12
|D28| rept("" = "",len(str(year.pay,4,0)))
|F28| d28
    
```

Note that the underlining of the two final figures assumes a monetary format.

You also should use a few more commands, to justify text right in the range B22:B27 (the list of items) and to justify numbers left over the cells containing the yearly and monthly payments.

You must also modify the numeric display format. Since many of the cells are still empty, simply changing the format will have no effect on them (see Section 3.2). You must change the default format of the cells to make the effect permanent, regardless of whether the cells are empty or not.

The following command will change the display default to monetary units over the whole of the budget application.

Units,Defaults,Monetary,

The display of Figure 5.6 uses decimal format, with two decimal places, except for the yearly and monthly payments, which are in monetary format. The appropriate commands are:

**Units,Defaults,Decimal,Decimal places 2
Units,Cells,Monetary,d27:f27**

This last command can use the Cells option since the cells concerned already exist.

You can enter values in this table by moving the cursor to the appropriate cell and typing in the number. The easiest way of moving the cursor is to press *f5* (Go to cell) and then use a cell label, such as

apr.gas

This example presents a more sophisticated bar chart display than that produced by the example of Section 5.3. In addition to extending the use of absolute and relative cell references, it introduces several new functions.

5.8 AN AUTO-SCALING BAR CHART

The chart displays twelve values, labelled by month. The values are read from twelve cells above the chart. The vertical scale is adjusted automatically to make sure that all values will fit the chart. It is only suited to displaying positive values.

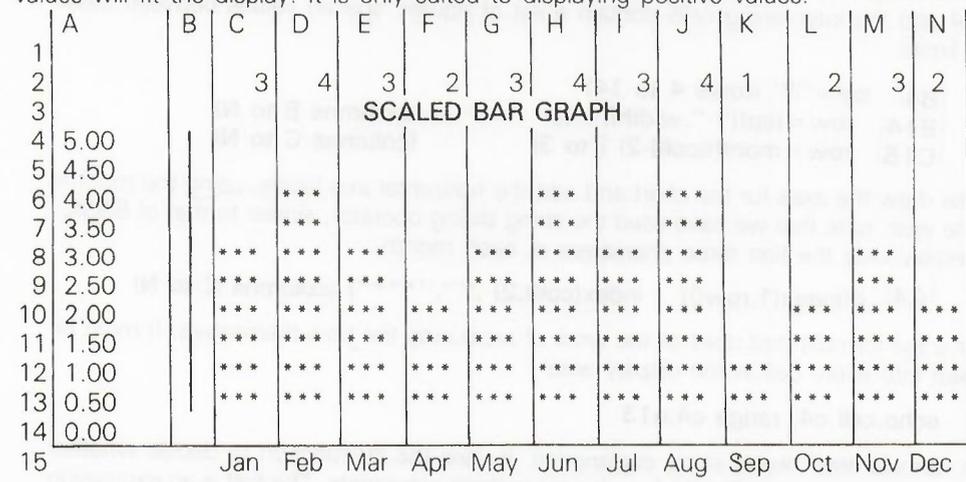


Figure 5.7 A Scaled Bar Chart.

First you should set the column widths to five in column A, one in column B and three in columns C to N, using the Width option of the Grid command.

|C2| row=0 (columns C to N)

Row two will contain the values to be displayed - for the moment it is filled with zeroes, as dummy values.

|F3| "SCALED BAR GRAPH"

|P2| int(max(c2:n2)/5 + 1)*5

|Q2| int(min(c2:n2)/5)*5

Cells P2 and Q2 contain the maximum and minimum values for the vertical scale of the graph. These cells are chosen so that they do not appear in the final display of the chart. Their initial values (when all the displayed numbers are zero) are five and zero respectively.

The max() function finds the maximum, or largest, numerical value in the range of cells specified by its argument. Similarly, the min() function finds the minimum, or smallest, value in the range.

Let us first examine the formula in cell Q2. The min() function finds the minimum, or smallest, value in the specified range and this is then divided by five. The int() function then removes the fractional part of the result of the division. If, for example, the minimum value is 13, dividing by 5 gives a value 2.6, and int(2.6) is 2. When this is multiplied by 5 we end up with a value of 10, which is the largest multiple of 5 that is less than the minimum.

The formula in cell P2 is similar, except that it finds the largest value in the range and adds 1 to the number before the final multiplication by 5. If, as an example, we assume that the maximum value is 21, you can verify that the formula will give a value of 25 - the smallest multiple of 5 that is greater than the maximum.

The two values in these cells will therefore always bracket the values in the cells from C2 to N2. Their difference is always a multiple of five.

The next formula displays the vertical scale of the graph in column A.

|A4| col = \$q2 + (14-row())*(\$p2-\$q2)/10 (rows 4 to 14)

The interval between successive numbers in the scale is (P2-Q2)/10. Note that, because we made the difference between the contents of P2 and Q2 a multiple of five, this interval can not have an awkward value.

This interval is multiplied by a number (14-row()) which starts at zero in row fourteen and increases by steps of one to a value of ten in row four. The result is added to the smallest value, from cell Q2, to produce the number for each cell.

The net result is that the value in cell Q2 is displayed in A14, the value from P2 is displayed in A4 and the intervening cells contain a set of equally spaced values between these two limits.

|B4| col = "" (rows 4 to 14)
|B14| row = rept("-",width() + 1) (columns B to N)
|C15| row = month(col()-2) (to 3) (columns C to N)

These draw the axes for the chart and add the horizontal axis labels, using the months of the year. note that we have used the string slicing operator, similar to that of BASIC, to display only the first three characters of each month.

|C4| if(index(1,row()) index(col()),2) ,"" , "*") (columns C to N)**

This is the formula that does all the work of producing the bars themselves. It must be copied into every cell in the display area:

echo,cell c4, range c4:n13

The formula itself needs some explanation. It uses the if() function to decide whether to display part of a bar. The if() function takes three arguments. The first is an expression which must give a numeric result. If this result is non-zero the cell displays the second argument, which may be text or numeric. If, however, the result is zero the third argument is displayed in the cell. Again this may be text or numeric.

In each cell the formula compares the number in column one of that row (the value labelling the vertical axis) with the number in row two of that column (the value to be displayed in the graph). If the axis label is greater than the display value, the condition is true (it evaluates to 1) and nothing is displayed. If the axis label is less than or equal to the display value, the condition results in a value of zero, and three asterisks are shown in the cell. The net result is that a bar is drawn to the correct height in each column.

Since a single formula is used for all the cells in the display, the cell reference can be neither absolute nor relative, as a few moment's thought will show. The reference to the display values must change as we move from column to column (ie it must be relative

PROVISIONAL

along a column) but must always refer to row two as we move down, from row to row. We need a form of cell reference which is relative with respect to columns, but absolute with respect to rows.

Fortunately the `index()` function can be used to produce this effect. It takes two parameters, a column number and then a row number, returning the contents of the specified cell. With this we can construct any combination of absolute and relative references, the following examples show:

Function	Column Ref.	Row Ref.
<code>index(5,5)</code>	absolute	absolute
<code>index(col(),5)</code>	relative	absolute
<code>index(5,row())</code>	absolute	relative
<code>index(col(),row())</code>	relative	relative

The function `index(col(),2)` therefore returns the contents of the cell in row two of the current column, and `index(1,row())` returns the contents of the cell in column one (A) of the current row.

Try putting different values in cells C2 to N2 and see what effect they have on the display.

This example enables you to calculate the monthly payments due on a repayment mortgage. You are asked to input the amount of the loan, the interest rate, the length of the loan in years and the month of the first payment. The required repayments are calculated and displayed, together with a complete repayment table for the whole period of the loan. This table shows you the outstanding sum at the beginning of each month until the loan is repaid.

5.9 MORTGAGE CALCULATOR

Several of the calculations in the grid make use of values that are input by use of the `askn()` function. As was pointed out in Section 5.4, these cells will not contain their correct values until you use the `Xecute` command.

In this section we shall produce the part of the grid that accepts your input and calculates the monthly repayments. When you have typed in the formulae and added a few figures in response to the `askn()` functions it should look like Figure 5.8.

5.9.1 Mortgage Repayment Calculations

	A	B	C	D	E
1			<u>MORTGAGE REPAYMENT CALCULATOR</u>		
2					
3					
4		Loan	£25000.00		
5		Int rate	14.00%		Mnth
6		Term	25	Start	April
7					
8				<u>REPAYMENTS</u>	
9					
10			Annual	£3774.65	
11			Monthly	£314.55	
12					

Figure 5.8 Calculating the Repayments.

```

|C1| "MORTGAGE REPAYMENT CALCULATOR
|C2| rept("=",len(c1))

|B4| "Loan
|C4| askn("Amount of loan? ")
  
```

The next three entries request the input of the interest rate. The original input is to a cell well away from the displayed portion of the grid so that it can be entered as a percentage figure. The value needed by the rest of the formulae is a fractional value (eg 12| must be used as 0.12) which is calculated from the input by the formula in cell C5.

```

|H4| askn("Percentage interest rate? ")
|B5| "Int rate
|C5| h4/100

|B6| "Term
|C6| askn("Period of loan in years maximum 35 ? ")

|E5| "Mnth
|D6| "Start
|E6| askn("Month of first payment Jan = 1, Feb = 2, etc? ")

|D8| "REPAYMENTS
|D9| rept("-",len(d8))
|C10| "Annual
|D10| mor.loan *mor.int/(1-(1 + mor.int) (-mor.term))

```

This formula, which calculates the annual repayment, assumes that the interest is calculated annually and added to the loan before the twelve monthly repayments are made.

```

|C11| "Monthly
|D11| ann.rep/12
|D12| d9

```

The grid is now sufficiently complete to calculate mortgage repayments. Try using the Xecute command and enter the figures requested, so that you can see it working.

To make the appearance more acceptable, we can change the format of some of the numbers with the Units command. In this example there is no need to alter the default numeric format (see Section 3.2.2) since you do not need to make new entries in any grid cell once the application is completed.

```

Units,Cells,Percent,Decimal places 2,c5
Units,Cells,Monetary,c4
Units,Cells,Monetary,d10:d11

```

5.9.2 Mortgage Repayment Table

This section describes how you can add a repayment table to the mortgage calculator. The first part of a repayment table for the values appearing in Figure 5.8 is illustrated in Figure 5.9.

	A	B	C	D	E
15			<u>REPAYMENT TABLE</u>		
16					
17					
18		Year	1	2	3
19					
20		April	28500.00	28186.90	27829.96
21		May	28185.45	27872.34	27515.41
22		June	27870.89	27557.79	27200.86
23		July	27556.34	27243.24	26886.30
24		August	27241.78	26928.68	26571.75
25		September	26927.23	26614.13	26257.19
26		October	26612.67	26299.57	25942.64
27		November	26298.12	25985.02	25628.08
28		December	25983.57	25670.47	25313.53
29		January	25669.01	25355.91	24998.98
30		February	25354.46	25041.36	24684.42
31		March	25039.90	24726.80	24369.87
32					
33		Year	1	2	3
34					
35	End-of-year	balance	24725.35	24412.25	24055.31

Figure 5.9 The Repayment Table.
(First 5 columns)

PROVISIONAL

If you have a mortgage, type in your own figures. Don't spend too much time over the results for the first few years - they make rather depressing reading!

```
|C15| "REPAYMENT TABLE
|C16| rept("=",len(c15))
|B18| "Year
|C18| row=col()-2 (columns C to AK)
|B19| row=rept("-",width()+1) (columns B to AK)
|B20| col=month(row()-20+$mnth.start) (rows 20 to 31)
```

These entries set up the headers for the table: now we must add the formulae that will calculate the values. We start with the first item which is the initial amount due. It is calculated by adding the first year's interest to the amount of the loan.

```
|C20| mor.loan*(1+mor.int)
```

Then the rest of the first row is calculated by subtracting the yearly payment and adding the interest for the current year. These values should not be calculated beyond the year in which the loan is repaid and we allow for this by using the if() function. If the year number (given by col()-2) is greater than the term of the mortgage, zero is placed in the cell.

```
|D20| if((col()-2) $mor.term,0,(c20-$ann.rep)*(1+$mor.int))
(columns D to AK)
```

The remainder of the table can be filled with a single formula. We fill the first column with a formula which just subtracts the monthly repayment from the amount in the cell above. Again we use the if() function to prevent the calculations extending beyond the year in which the loan is repaid.

```
|C21| if((col()-2) $mor.term,0,c20-$mon.rep) (rows 21 to 31)
```

You can then use the Echo command to copy the formula from cell C21 to the region starting at D21 and ending at AK31.

We can now complete the table by adding a final row to give the outstanding balance at the end of each year. It is probably a good idea to add a copy of the year, from row 18, for easy reference.

```
|B33| row=year.term (columns B to AK)
|A35| "End-of-year balance
|C35| if((col()-2) $mor.term,"",c31-mon.rep) (columns C to AK)
```

The entire table, and the end-of-year balances should be set to either monetary format or to decimal format with two places of decimals. The ranges for these changes are C20:AK31 and C35:AK35 respectively.

The French scientist Fourier showed that a repetitive wave of any shape can be built up from a set of sine or cosine waves of the correct amplitudes and frequencies. The building up of complex waves from pure sine and cosine waves is known as Fourier synthesis and is employed, for example, in many of the music synthesisers in use today.

5.10 FOURIER ANALYSIS

The opposite process, decomposing a complex wave shape into a number of pure sine and cosine waves, is known as Fourier analysis. This example allows you to perform a Fourier analysis of any shape of wave. All you have to do is type in the height of the wave at sixteen equally-spaced intervals and let the formulae in the grid do the rest. The formulae assume that the wave repeats its shape after the sixteenth value, ie that the seventeenth value is the same as the first, the eighteenth is the same as the second, and so on.

The mathematical process that is carried out in this example is known as a Fourier transform. A graph of the input values shows the shape of the wave and a graph of the output shows how much of each frequency of each sine or cosine wave is present.

Some examples of the results of performing Fourier transforms on a series of input waveforms are shown in Figure 5.10. The 'frequency' axis of each output graph is in terms of the number of complete cycles of each component that are present in the input. In Figure 5.10 this axis is labelled and the values extend from zero to fifteen. Figure 5.10(A), for example, shows the results of performing a Fourier transform on one cycle of a cosine wave.

You may be puzzled by the appearance of two peaks, at frequencies of 1 and 15, when only one input frequency was present. The mathematical process of the transform is not capable of measuring frequencies greater than 8 (in general the upper limit is half the number of sample points in the original function). The calculation introduces spurious results, so that the second half of the graph of the transform is a mirror image of the first. For practical purposes you should ignore the right hand side of each transform when interpreting the results. In Figure 5.10 the spurious frequencies are shown as broken lines.

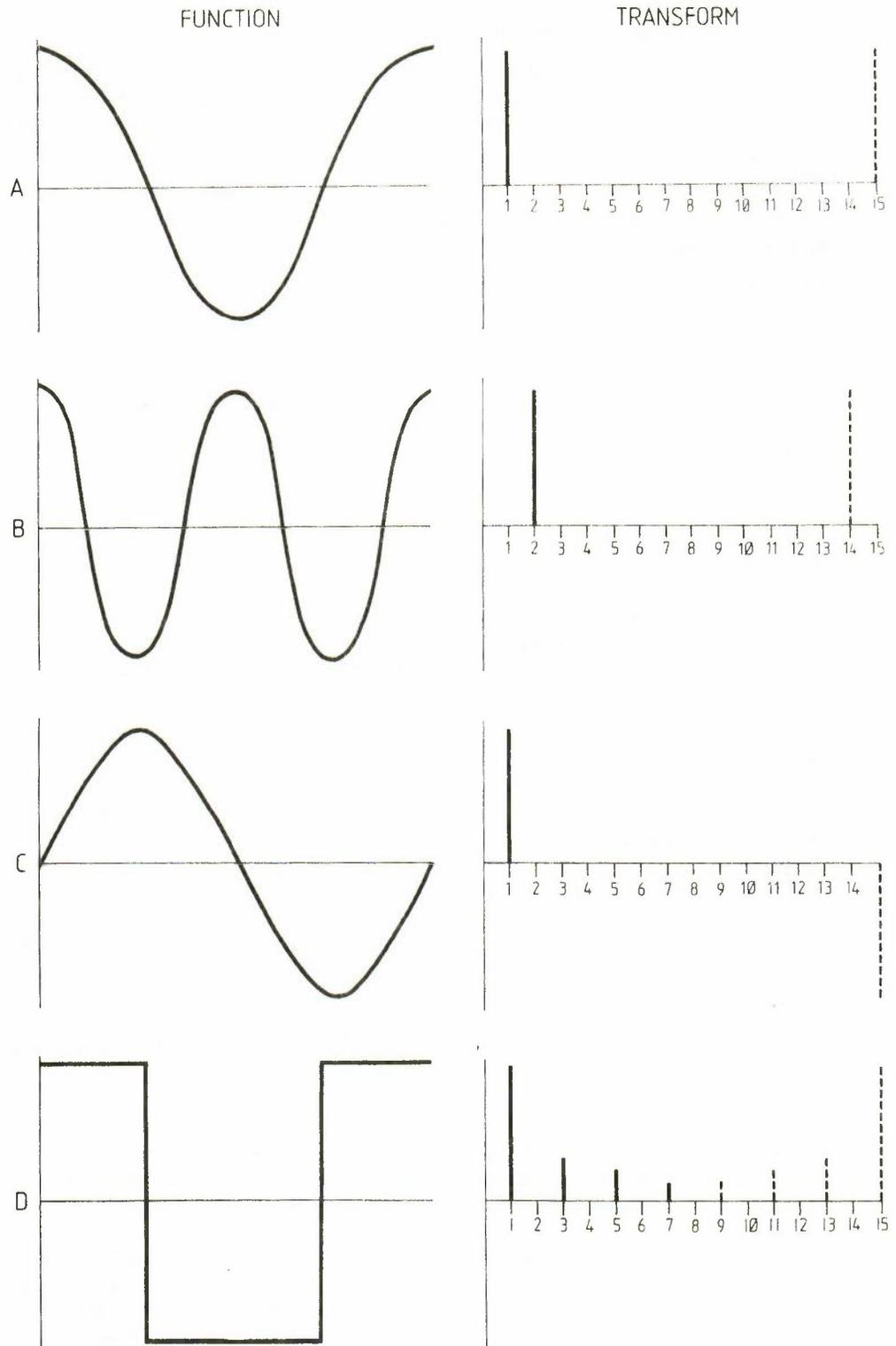


Figure 5.10 Fourier Transforms

Figure 5.10(B) shows the transform of two cycles of a cosine wave - double the frequency of that in Figure 5.10(A) - and, as you would expect, the graph of the transform shows the higher frequency. Figure 5.10(C) shows the transform of a single cycle of a sine wave. In this case the spurious result turns out to be negative.

The final pair of graphs, in Figure 5.10(D), shows an example of a waveform that is made up of a number of components of different frequencies. The original waveform is, for obvious reasons, known as a square wave. The diagram shows that such a wave is composed of decreasing amounts of the odd harmonics (frequencies 1, 3, 5, 7, etc. are present).

Since the calculation takes an appreciable time it is worth turning off the auto-calculate, by use of the Design command, before typing in the example.

5.10.1 Calculating the Fourier Transform

```
|C1| "FOURIER ANALYSIS
|C2| rept("=",len(c1))

|B3| "Function:
|A8| "Input
|A9| "Values
```

The input values are placed in the sixteen cells from B9 to B24 inclusive.

We shall now set up the headings for the table which will calculate the cosine components of the wave. The result contains the amounts of all cosine-like waves in the input.

The Cosine Components

```
|E3| "Transform:
|E4| "Cosine
|D6| "Cycles
row = col()-5 (columns E to T)
|E8| "Sample
col = row()-9 (rows 9 to 24)
```

Surprisingly, the entire cosine transformation can be performed by a single formula. In each row the input value is multiplied by the cosine of an angle (in radians) which is calculated as follows:

$$\text{angle} = 2 * \text{pi}() * \text{rownumber} * \text{colnumber} / 16$$

The row number and column number are the values given in the row labelled 'Cycle' and the column labelled 'Sample' respectively. They each count up from zero to fifteen. The final divisor is simply the number of points in the input (or output).

```
|E9| row = index(2,row())*cos(pi()* (row()-9)*(col()-5)/8)
(colums E to T)
```

Now use the Echo command to copy the contents of cell E9 to the cells in the range from E10 to T24.

The final result is calculated by summing the contents of each column to produce the sixteen output values.

```
|A26| "Components
|E26| row = sum(col) (rows 9 to 24, columns E to T)
```

All the trigonometric functions deal with angles measured in radians, rather than in degrees. The difference is only one of using different units - rather like measuring a length in either inches or centimeters. A complete circle contains 360 degrees or $2 * \text{pi}()$ radians. You can convert between degree and radian measure as follows:

Radian Measure

$$\begin{aligned} \text{radians} &= \text{degrees} * \text{pi}()/180 \\ \text{degrees} &= \text{radians} * 180/\text{pi}() \end{aligned}$$

The calculation of the sine components follows exactly the same pattern as for the cosine ones. The resulting values are the amounts of all sine-like waves in the input.

The Sine Components

```
|X4| "Sine
|X6| row = col()-24 (columns X to AM)
|X9| row = index(2,row())*sin(pi()* (row()-9)*(col()-24)/8)
(colums X to AM)
```

Now Echo the contents of cell X9 over the range from X10 to AM24, to fill in the rest of the table, and Echo the contents of cell C9 to column V, from V9 to V24 (this makes

The next set of entries graphs the cosine components.

```
|F31| "" = max
|E31| max(e26:t26)
|F32| "" = min
|E32| min(e26:t26)
|E33| col = rept("'",(index(row()-28,26)-$cos.min)
      * 18/($cos.max-$cos.min + 1)) + "" (rows 33 to 48)
```

The final set of entries gives a graph of the sine components.

```
|Y31| "" = max
|X31| max(x26:am26)
|Y32| "" = min
|X32| min(x26:am26)
|X33| col = rept("'",(index(row()-9,26)-$sin.min)
      * 18/($sin.max-$sin.min + 1)) + "" (rows 33 to 48)
```

All of these graphs will cope with negative values. In such a case they add an offset to all the values. When this happens, the zero level of the graph is shifted towards the mid-point of each display. Negative results then appear as bars which do not extend as far as the zero point on the graph.

As was mentioned earlier, you should put the input values in cells B9 to B24 inclusive. You may try any set of values you like, but here are a few suggestions. The first uses a formula to place a single cycle of a pure cosine curve in the input.

```
|B9| col = 10*cos(pi()*(row()-9)/8) (rows 9 to 24)
```

The input and output curves are illustrated in Figure 5.11 and, in slightly more detail, in Figure 5.10(A).

The second example shows a cosine curve which includes two complete cycles. It is illustrated in Figure 5.10(B).

```
|B9| col = 10*cos(pi()*(row()-9)/4) (rows 9 to 24)
```

The third example, illustrated in Figure 5.10(C), is for a single cycle of a pure sine wave. In this case the output appears in the sine transform, and all cosine components are zero.

```
|B9| col = 10*sin(pi()*(row()-9)/8) (rows 9 to 24)
```

The fourth example represents a single cycle of a square wave. The easiest way of entering this one is to use the `sgn()` function on the cosine curve of the first example. This function returns +1, -1 or 0, depending on whether its argument is positive, negative or zero.

```
|B9| col = 10*sgn(cos(pi()*(row()-9)/8)) (rows 9 to 24)
```

The illustration of Figure 5.10(D) shows that a square wave contains more than one frequency. It is composed of decreasing amounts of all odd harmonics (frequencies of 1, 3, 5, 7, etc).

A fifth example (not illustrated) is for a constant input. You will see that the output shows that there is only a zero-frequency component present.

```
|B9| col = 10 (rows 9 to 24)
```

Remember that, since the auto-calculate is turned off, you must use Xecute to calculate each result.

A further advantage of including lots of labels is that you can move the window to most of the interesting points in the grid by using the `goto (f5)` facility, followed by a cell reference in its label form. Here is a list of some of the more useful cell references that you might need:

input.values
cosine.transform
sine.transform
cosine.components
sine.components
power.components

and, if you have added the graphics,

input.graph
power.graph
cosine.graph
sine.graph

5.10.3 Using the Fourier Transform

CHAPTER 6 ABACUS REFERENCE

You can refer to single cells, rows, columns or ranges either by using explicit letter and number references or by using text labels. This section describes the use of explicit references; Section 6.2 explains how you can use labels.

A reference to a single cell consists of two parts, a column and a row reference.

There are 64 columns in the grid and they are labelled from A to BL. There are 256 rows, numbered from 1 to 256. Typical cell references are

A1 AC13 BD200

You can specify a whole block of cells in the grid by means of a block reference. It is made up of two cell references, separated by a colon. You must always type in the colon to separate the two parts of the reference. The first cell reference specifies the top left hand corner of the block and the second one identifies the bottom right hand corner. Examples of range references are:

B5:D9 AZ23:BA155

A part of a row or column can be considered as a range that is only one column wide (or one row deep). You can therefore use a range reference to specify part of a row or column, such as:

**A3:L3 (cells A to L of row 3)
D7:D11 (cells 7 to 11 of column D)**

A range identifier is one of row or col. They refer to the cells of the current row or the current column respectively. (The current row and column are those that intersect at the cell containing the range identifier.)

Each time you use one of them in a formula you will be asked to specify the exact range of cells within the row or column. Abacus will suggest reasonable starting and ending points for the range and you can either accept this choice or change it.

There are two ways in which you can use range identifiers. You can fill the current row or column by use of either

row = (formula) or col = (formula)

You can also use them as the argument for any function that requires a range, for example, count(row). You can, of course, only use them in this way when you only want to refer to the cells of a single row or column.

You can mix the two methods freely, for example,

col = ave(row)

Each occurrence in a formula will result in ABACUS asking you for a particular range.

ABACUS normally assumes that all cell references are relative, ie, that the important thing is the difference in position between the cell containing the reference and the cell to which you refer. When you copy such a reference into another cell, the references are modified to keep this relative difference. For example, imagine that a formula in cell B2 contains a reference to cell A1 (one column to the left and one row above). If the formula in cell B2 is copied into cell D4 it will, in this new location, refer to cell C3 (again one column to the left and one row above).

This is illustrated in Figure 6.1. A formula in cell X contains a reference to the lightly shaded cell. If this formula is copied to cell Y it then refers to the heavily shaded cell. The two cells in each pair have the same relative positions.

6.1 CELL REFERENCES

6.1.1 Single Cells

6.1.2 Range References

6.1.3 Row and Column References

6.1.4 Range Identifiers

6.1.5 Relative and Absolute Cell References

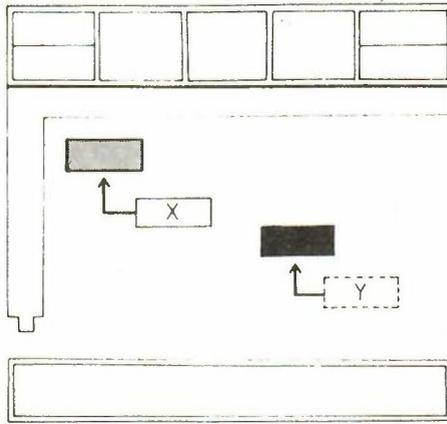


Figure 6.1 Relative Cell References.

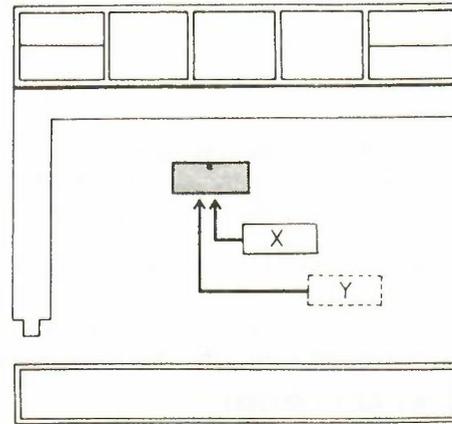


Figure 6.2 Absolute Cell References.

Suppose we put the formula $A1*2$ into cell A2, and then use the Echo command to copy the formula into cells in the range B2:G2. Examining the cells of row 2 will show that they have the following contents:

CELL:	A2	B2	C2	D2	E2	F2	G2
Contents:	$A1*2$	$B1*2$	$C1*2$	$D1*2$	$E1*2$	$F1*2$	$G1*2$

You can make any cell reference absolute by prefacing it with a \$ sign. Such a reference will not be modified when the formula is copied to other cells. For example, if a reference in cell B2 was to $\$A1$, any copy of the formula will also contain the reference $\$A1$. Figure 6.2 shows the effect of an absolute cell reference. A formula in cell X contains an absolute reference to the shaded cell. A copy of the formula in cell Y refers to the same cell.

Let us try the previous example, but this time we shall use an absolute reference. Put the formula $\$A1*2$ in cell A2 and Echo it to cells B2 to G2 inclusive. You will then find that the cells contain the following:

CELL:	A2	B2	C2	D2	E2	F2	G2
Contents:	$\$A1*2$						

See also the `index()` function. Its use is fully explained in Section 5.8.

6.2 LABELS

6.2.1 Row and Column Labels

A label is a cell containing a text string, used to identify a row or column in the grid. You can also use labels to refer to a single cell, but you may not use them to replace a range reference, to refer to a whole block of cells.

Whenever you refer to a label in an expression or formula, ABACUS uses a set of rules to determine whether it refers to a row, a column or a cell. The rules for rows and columns are:

1. The row and column intersecting at the label are scanned (to the right and below) to find a numeric entry.
 - a) If only a row entry is found, the label refers to the row, starting at the found entry.
 - b) If only a column entry is found, the label refers to the column, starting at the found entry.
 - c) If entries are found in both the row and the column the entry closest to the labelled cell is used to make the choice.
2. If no decision can be made under 1), but the label is used on the left hand side of an expression, it will be given the type of any label(s) used on the right hand side.

If both of these rules fail, you are told that ABACUS can not decide the meaning of the label.

6.2.2 Cell Labels

You need to use two labels to identify a single cell and you make the cell reference by giving both labels, separated by a full stop. For example, if you have two labels "fruit" and "apples", you can refer to a cell as

fruit.apples

(or by any unique abbreviation, such as fr.ap). The order of the two labels is unimportant so you could also use apples.fruit, ap.fr and so on.

Such a reference refers to the cell at the intersection of the rows and columns containing the labels but, as Figure 6.3 shows, there are two such cells (labelled X and Y).

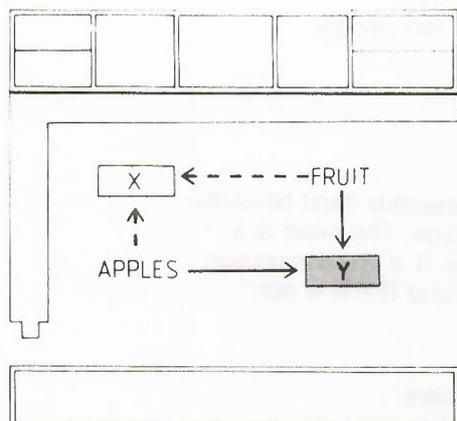


Figure 6.3 Labelling a Cell.

The cell that is selected is the one in the right-most column and the lower of the two rows. In the previous example, the cell labelled Y will be selected. You should, therefore, always place labels above or to the left of the cells to which they refer.

A formula is any allowed combination of functions, cell references, labels and arithmetic operators. Examples are:

```
A1*B3
month(col()-1)
if(instr(B6,"is"),1,0)
rept("=",len(G23))+":"
```

6.3 FORMULAE

Each new formula, in addition to being used in one or more grid cells, is stored separately in a list of master formulae. Each master formula may therefore appear in one cell or in many. When you fill cells by use of the row and column fill operations, or by using the Copy and Echo commands, all the filled cells share a single master formula. If a master formula contains relative cell references they are adjusted, in each cell using the formula, to be valid for that particular location. The formulae may therefore appear superficially different but are all based on the one master formula.

6.3.1 Master Formulae

There are several advantages to this approach, the most important being that less memory is used than if the formula in every cell is stored separately. This allows you to construct much more complex grids without running out of memory. A further advantage is that you can modify all copies of the formula by editing only one of the copies. If you use the Amend command to change any copy of a master formula, the master is also modified and all copies are changed simultaneously. This is why you should use the row and column operations, and the Echo and Copy commands, whenever possible, rather than filling each cell separately.

One case, however, needs explanation to remove a possible source of confusion. When you use the Grid command to insert or delete a row (or column) then any relative cell references in the formulae in the following cells are modified to take their new positions into account. ABACUS locates the first formula in the affected portion and uses this as a model to work out the necessary changes to the remaining formulae. All relevant master formulae are modified. If you have used copies of a master formula on both sides of the change to the grid it is possible, in some circumstances, for the formulae in cells before the position of the insertion or deletion to be made incorrect. This is unlikely to happen if you plan your grid correctly. If, however, you do find that this has happened, you can correct it by typing a replacement formula into the affected cells (so that they use a different master formula). You must not use the Amend command, as this will simply change the master formula and make the formulae in later cells incorrect for their new positions.

6.4 ARITHMETIC

The arithmetic operations in ABACUS follow the same rules as for the arithmetic in BASIC. The valid range for numbers is from $-2.9E-39$ to $+1.7E+38$. All calculations are accurate to 17 significant digits but only a maximum of 16 significant digits may be displayed.

The following arithmetic operations are provided:

- + Addition (on numbers), or concatenation (on strings)
 - Subtraction
 - * Multiplication
 - / Division
 - ^ Raising to a power
 - = Equal
 - > Greater than
 - < Less than
 - >= Less than or equal to
 - <= Greater than or equal to
 - ≠ Not equal to
- Both operands must be of the same type. The result is a number, 1 if the comparison is true and 0 if it is not.

Functions and operations have the following priorities:

Operation	Priority
Subscripting and slicing	12
All functions	11
^	10
Unary minus (ie, minus just used to negate something)	9
*, /	8
+, - (minus used to subtract one number from another)	6
=, <, >, >=, <=	5
not	4
and	3
or	2

In addition, string slicing is provided, again in a form similar to that of BASIC. The slicing operations provided are:

- (n) select the nth character.
- (n to m) select all characters from the nth to mth character inclusive.
- (n to) select from character n to end.
- (to m) select from the beginning to the mth character.

6.5 THE FUNCTION KEYS

The five function keys are used as follows:

- F1 call the Help facility
- F2 remove/restore control area
- F3 call the commands
- F4 move cursor between the two halves of a split window
- F5 Go to a cell

6.6 THE LINE EDITOR

The line editor is always available to modify the contents of the input line.

Key(s)	Action
Left cursor	Move one character to the left
Right cursor	Move one character to the right
Up cursor	Move one word to the left
Down cursor	Move one word to the right
CTRL + Left cursor	Delete one character to the left
CTRL + Right cursor	Delete one character to the right
CTRL + Up cursor	Delete all text to the left
CTRL + Down cursor	Delete all text to the right
SHIFT + Left cursor	Move left by one word
SHIFT + Right cursor	Move right by one word

A full file name consists of three sections, separated by underscores. The three components are:

an optional drive specifier	eg MDV1
a file name of up to x characters	eg FRED
an optional three-letter extension	eg ABA

A full file name for an ABACUS file could therefore be:

MDV2__FRED__ABA

If you do not include a drive specifier in a file name then ABACUS assumes that you are referring to the current drive, that is, the drive that was last used. The one exception is when you are loading ABACUS itself from BASIC, as described in Section 2.1. In this case you must include the drive specifier in the file name.

You do not normally need to specify an extension since ABACUS supplies a default extension for every file access. The Load and Save commands supply a default extension of __ABA. The default extension for Import and Export files is __EXP, and when you Print to a file the default extension is __LIS.

If you include an extension in any file name you type in then it will be used in preference to the default extension normally provided by ABACUS.

Every time that an ABACUS command asks you to type in a file name you have the option of pressing the ? key to obtain a list of the names of files on the current drive. The file name "*"__*" (file name and extension) will appear in the input line and, if you accept this by pressing ENTER, you will be given a list of all files on the current drive.

In this context the "*" character is a *wild card* which stands for any sequence of characters. You may also use the character "?" to represent any single character in a file name.

6.7.2 Wild Cards

You have the option of using the line editor to modify the suggested file name, in order to obtain a list of the names of a particular group of files.

If, for example, you edit the file name to read "*"__TST" and then press ENTER you will be given a list of the names of all files with an extension of __TST. Changing the file name to "X*"__*" would result in a listing of all files, with any extension, whose names begin with X.

You could use the single character wild card as, for example,

MYFILE?__*

which would result in a listing of all files with names such as:

MYFILE1 MYFILE2 MYFILE3

and so on, with any extension.

Note that this facility is only available when you are requesting a list of file names before typing in a file name for any of the file-based commands (Files, Load, Save and Print).

This section contains a full description of all the commands available in ABACUS. Many of the commands require additional options or qualifiers to specify, for example, the range of cells over which they are to act. In such cases the options will be described in the same way that they are presented on the display when you actually use the command. You can access the commands by pressing f3, when a list of the commands is displayed (this is the command menu). Any command can then be invoked by pressing a single key, which is the first letter (given in upper-case in the menu) of the command. In the descriptions given below, this single letter is shown in brackets after the name of the command. All of the commands leave you in the main grid display.

6.8 THE COMMANDS

AMEND (A) This command allows you to change the contents of a cell. The contents of the current cell are copied to the input line, ready for editing with the line editor described in Sections 2.10 and 6.6.

When you press **ENTER** the edited version replaces the original cell contents.

Remember that you will also edit the master formula (see Section 6.3.1) so that any change to a formula in a cell will cause a corresponding change in all other cells that share the same formula.

COPY (C) You use this command to copy one or more cells from one area of the grid to another. You will first be asked to give the range reference of the block to be copied, eg A1:B3, and should then press **ENTER**. You will next be asked to specify the cell reference for the top left hand corner of the area to which the block of cells is to be copied. When you then press **ENTER** the block of cells will be copied.

If any of the formulae that are being copied contain relative cell references, these references will be adjusted during the copy to be valid in the new area of the grid.

DESIGN (D) You use the Design command to modify a number of the 'background' features of ABACUS, such as whether the display should be set for a domestic television or a monitor. The choices remain in force until you modify them again, or until you leave ABACUS. When you save an application these defaults are saved with it so that they are used every time you load the application.

Changing the defaults, however, does not affect ABACUS itself. They must be reset to the values you want each time you reload ABACUS from BASIC to create a new application.

In the Design command the grid display is replaced by a list of options. You should respond by pressing the key corresponding to the first letter of the option you require. You may then be asked to specify further information, which will be either a number or a single letter, as described in the following list. At the end of each default selection you are returned to the Design command menu so that you can make multiple changes.

When you have finished you can return to the main grid display by pressing the X key.

If you press **ESC** you will return immediately to the main grid display. In this case all of the changes you made with that use of the command will be cancelled.

The options are:

(Design) A used to specify auto-calculate or no auto-calculate. Each time you press the A key the auto-calculate option switches between YES and NO.

If you choose YES, the whole spreadsheet will be recalculated after each entry. Selecting NO, however, means that the spreadsheet will only be recalculated when you use the Xecute command. The initial value is YES.

(Design) B switches between two ways of treating zero values in the grid. The default is to display the value zero in the appropriate format for that cell. You may select the alternative, which is to display a blank cell if its contents evaluate to zero.

Note that, in this option, a blank cell will only be shown if the value is truly zero. Suppose you have selected decimal display format, with two decimal places, and the value in such a cell is 0.003. The cell will show 0.00, rather than being blank, since the true value is non-zero.

(Design) C selects between calculating the spreadsheet in ROW or COLUMN order. The option changes each time you press the C key (as for auto-calculate). The specified order will be used for both auto-calculate and the Xecute command. The initial value is for row order.

PROVISIONAL

- (Design) F selects whether or not a form-feed is issued at the end of each page of printed output, in the same way as for auto-calculate. The initial value is YES.
- (Design) G sets the line spacing on printed output by specifying the number of gaps between the lines of text. You are asked to type in 0, 1 or 2 (no **ENTER** is necessary). You can set ordinary double-spaced printer output, for example, by specifying one gap between each line. The initial value is zero.
- (Design) L specifies how many lines on a page of printed output. You should type in a number, followed by **ENTER**. The initial value is 66.
- (Design) M specifies the currency sign to be used in the display of monetary values. You should type in the single character that you want (no **ENTER** is necessary). The initial value is the pound sign.
- (Design) P sets the number of characters per line of printed output. You should type in a number, followed by **ENTER**. The initial value is 80.
- (Design) S switches between the use of continuous or separate sheet printer stationery, in the same way as for auto-calculate. The initial value is for continuous stationery.
- (Design) T selects the form of display for use with a domestic television or a monitor, in the same way as for auto-calculate. The initial value is for a monitor display.
- ECHO (E) The Echo command makes a copy of the data or formula in a particular cell to all the cells in a specified range.
- You are given the option of specifying the cell reference of the cell to be copied, or pressing **ENTER** to use the current cell. You then should type in the range over which the cell contents are to be copied, followed by **ENTER**.
- FILES (F) The appearance of the display for the Files command is shown in Figure 6.4.

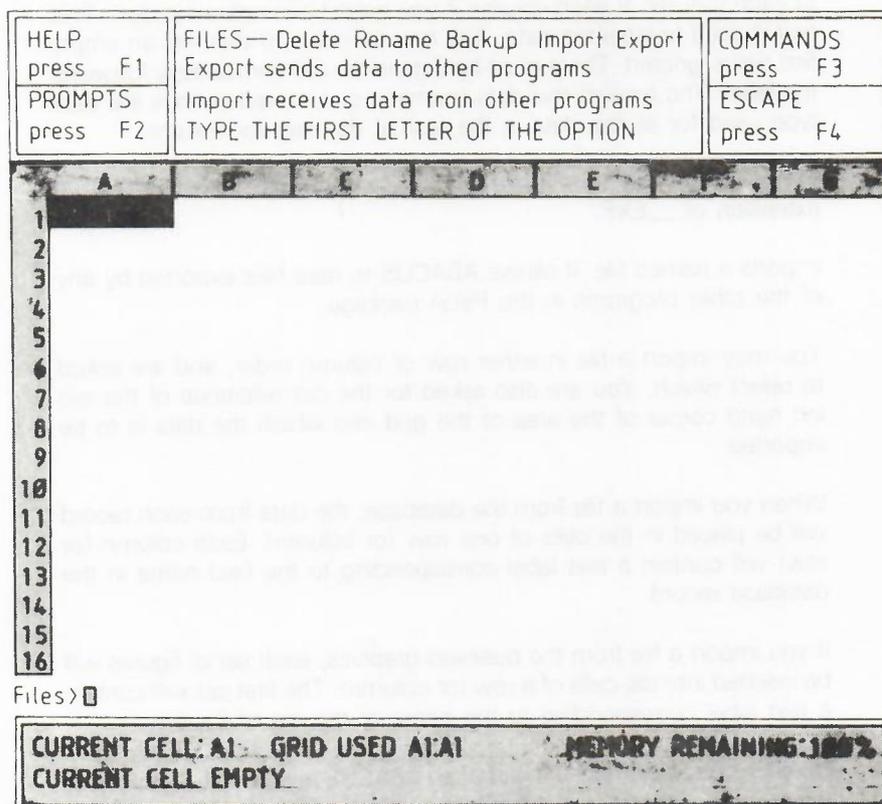


Figure 6.4 The Files Command.

This command allows you to modify ABACUS files, previously saved on a Microdrive cartridge. The options ask you to type in the names of files. Each time you are asked for a file name you can press ? for a list of all files on the current cartridge. You can then accept the suggestion of * __ * to display all the files (by pressing ENTER) or you can use the line editor to change either the file name or its extension, to list any particular subset. This is explained more fully in Section 6.7.

At the conclusion of any option you are left in the Files command menu, ready to use another Files option. You can return to the main display by pressing ESC.

You are offered the following options:

Files >B used to make a backup copy of an ABACUS file. You are asked for the name of the file to be copied. Making copies of your files is strongly recommended, to protect yourself against accidental loss of, or damage to, the cartridge, and against making a mistake which causes your application to be corrupted or deleted.

Files >D deletes a named file from a Microdrive cartridge. **Note that this command is NOT reversible and should therefore be used with GREAT CARE.**

Files >E exports a named file. The file is saved in a form suitable for being read by the database or the business graphics. Note that you should not send files to the word processor by means of the Export option. Such files need additional formatting information and you should therefore use the Print command for this purpose.

You can export the file by rows or by columns and are first asked to press ENTER to accept the suggestion of exporting by rows, or to press the C key to choose export by columns.

You are then asked to type in the range reference for the section of the grid that you want to export, and finally asked to type in a name for the exported file.

The section of the grid being exported must have text in the first cell of each row (or of each column if you export it in column order). This text is used to label the data. Any row (or column) that has an empty first cell is ignored. There must be data in the cell immediately following the label. The type of this data (numeric or text) determines the data type used for all the data in the rest of the row (or column).

If you do not specify a file name extension ABACUS will supply an extension of __EXP.

Files >I imports a named file. It allows ABACUS to read files exported by any of the other programs in the Psion package.

You may import a file in either row or column order, and are asked to select which. You are also asked for the cell reference of the top left hand corner of the area of the grid into which the data is to be imported.

When you import a file from the database, the data from each record will be placed in the cells of one row (or column). Each column (or row) will contain a text label corresponding to the field name in the database record.

If you import a file from the business graphics, each set of figures will be inserted into the cells of a row (or column). The first cell will contain a text label corresponding to the name of the set of figures.

Note that you can invert the grid of an ABACUS application, exchanging rows and columns, by exporting the grid contents in row order and then importing the file in column order.

If you do not specify a file name extension ABACUS will assume an extension of `__EXP`.

Files>R renames a file. You are asked to type in the original file name, followed by the new name you want to give to the file.

If you do not specify a file name extension for the original file ABACUS will assume an extension of `__ABA`. If you do not specify a file name extension for the new file ABACUS will assume that it should be the same as for the old file.

GRID (G) The Grid command is used to make changes which affect the entire spreadsheet. It allows you to insert or delete an entire row or column, or to change the number of characters displayed in one or more columns.

At the conclusion of any option you are left in the Grid command menu, ready to use another Grid option. You can return to the main display by pressing **ESC**.

The options are:

Grid>I allows you to insert an empty row or column into the grid. You are first asked if you want to insert a row (press **ENTER**) or a column (press **C**). You are then asked to give the row (or column) reference. When you then press **ENTER** an empty row (or column) will be inserted above (or to the left of) the one specified. The last row (or column) will be lost from the grid. You will not be able to recover it.

Grid>D allows you to delete one or more rows or columns from the grid. You are first asked if you want to delete rows (press **ENTER**) or columns (press **C**). You are then asked to give the reference of the starting row (or column) of the region you want to delete, followed by **ENTER**. You are then asked for the row (or column) reference of the end of the region.

When you then press **ENTER** the selected region is deleted and the following rows (or columns) close up to fill the gap. Empty rows (or columns) will be inserted at the bottom (or at the extreme left of) the grid.

In both of these options all formulae in the rows or columns that are moved will be adjusted to correct them for their new positions (note the warning in Section xx).

Grid>W allows you to change the width (number of characters) of one or more columns. You are first asked to specify the number of characters in a column, and will then be asked to specify the starting and ending columns over which you wish the change to take effect.

JUSTIFY (J) The Justify command is used to modify the positioning of text and numbers in a range of cells. It has two main options; to modify existing cells, or to set the default justification for future creation of currently empty cells. You should press **ENTER** to select the Cells option, or the **D** key to select the Defaults option.

You are then asked to specify whether you want to modify the justification of text (by pressing **ENTER**) or of numbers (by pressing the **N** key). In either case you can then select left (**ENTER**), right (**R**) or centre (**C**) justification.

In the case of the Cells option you are finally asked to give the range over which the change is to act. You do not have to give a range in the Defaults option. The new default will apply to all newly-created cells, at any point in the grid, until you make a further change in the default justification.

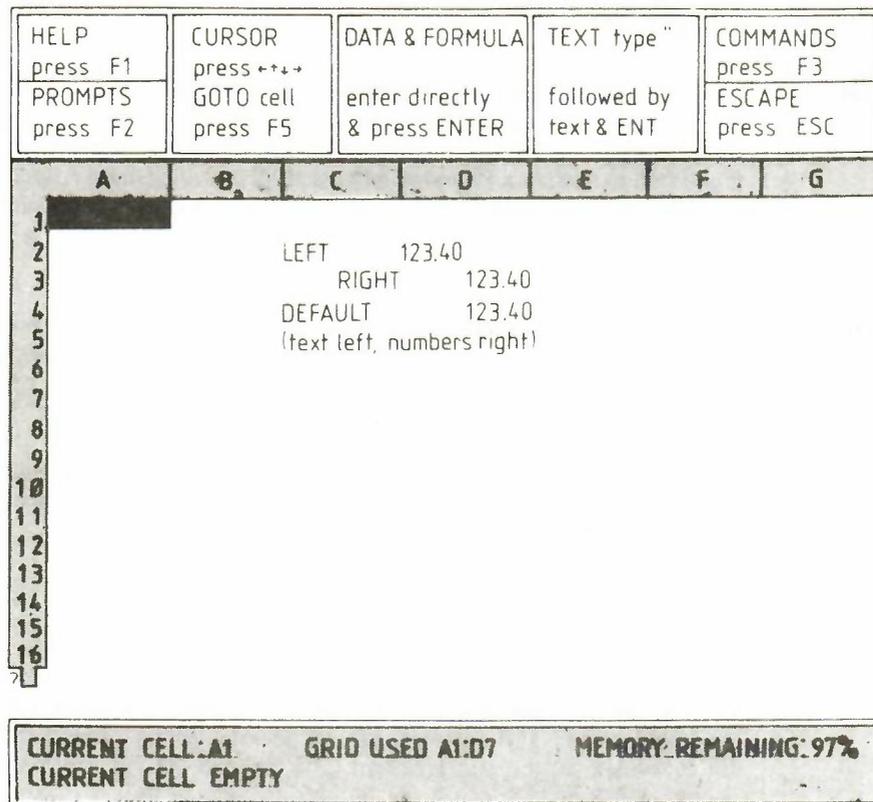


Figure 6.5 Justification.

Some of the different types of justification, together with the default settings (text justified left and numbers justified right) are shown in Figure 6.5.

LOAD (L)

This is used to load a file from the Microdrive. You are first asked to specify the file name; pressing the "?" key at this point gives you a list the files on the current Microdrive. You can, as for the Files and Save commands, choose to list all the files or you can choose to list a subset of the files as described in Section 6.7.

If you do not include an extension in the file name you type in, ABACUS will assume an extension of __ABA.

MERGE (M)

This command is used to combine, or consolidate, data from a previously saved file with the data in the current grid. You are first asked for the file name of the file to be merged from the Microdrive cartridge and will then have to indicate whether the data in this file is to be added to (press **ENTER**) or subtracted from (press **S**) the data in the current grid.

Whenever a cell in the file, containing a number or a formula, matches a corresponding data cell in the grid, the value from the file will be added to or subtracted from the grid data. The command will not have any effect on grid cells containing text, which are therefore protected against alteration.

The resulting grid contains only numeric values; the formulae that produced these values in the original grid cells will be destroyed. The formulae would not have any meaning in the consolidated grid.

This command offers a fast and easy method of combining the data in two similar models. It is, of course, essential that you have laid out the two grids in exactly the same way, using the same cell locations, for the results of the command to make sense.

ORDER (O) You use this command to sort the rows of the grid into ascending order, based on the contents of one particular column.

You are first asked to specify the column on which the sorting is to be based. You are then asked for the first and last rows to be sorted. The rows are rearranged so that the contents of the cells of the chosen column are in ascending order with increasing row number. The exact ordering sequence that is used is:

Empty cells

Numeric cells, in ascending numeric order

Text cells in alphabetic order

You should only use the Order command on rows or columns which contain data. It is likely to invalidate any formulae present in the affected portion of the grid.

PRINT (P) This command is used to send a selected portion of the grid to a printer or to a Microdrive file. You are first asked to specify the range of cells which you want printed. Then you are asked if you want the grid border to be included (press **ENTER**) or not (press the N key). Following this you should specify whether the output should be sent to the printer (press **ENTER**) or to a Microdrive file (press the F key). If you choose to send the output to a file, you are also asked to type in a file name (ending with **ENTER**).

The selected portion of the grid will be sent to the chosen destination.

If you want to send data to the word processor you should not use the Export option of the Files command since this discards the spacing between the cell items. You should use the Print command to send the grid contents to a file and then Import the print file into your word processor document.

If you do not, in the case of the option to print to a file, specify an extension when you type in the file name, ABACUS will assume an extension of **__PRI**.

QUIT (Q) You use the Quit command to leave ABACUS. On leaving the current grid contents are lost, so you are asked to confirm your request. You can cancel the command and return to your spreadsheet by pressing **ESC**. If you press **ENTER** you will confirm your wish to leave ABACUS.

RUBOUT (R) This command is used to rub out, or delete, the contents of one or more cells in the grid. When you use this command you will be asked to specify a range of cells. If you specify a range, all the cells in that range will be cleared; if you just press **ENTER** at this point only the current cell will be rubbed out.

SAVE (S) This is used to save a file to the Microdrive. You are first asked to specify the file name; pressing the "?" key at this point gives you a list the files on the current Microdrive. As for the Load and Files commands you can choose to list all the files, or select a subset by specifying all or part of the file name or the extension, as described in Section 6.7.

If you do not include an extension when you type in the file name, ABACUS will assume an extension of **__ABA**.

UNITS (U) The Units command is used to change the way that numbers are displayed within a cell, or group of cells. It does not affect the values of the numbers in any way.

You are first asked to select whether you want the command to affect existing cells (just press **ENTER**) or to set the default format for subsequently created cells (press the D key).

In either case you are then asked to choose the display format from the following list (the list assumes the Cells option):

- Units,Cells,D** numbers are displayed in fixed-point decimal notation, that is, all numbers are shown in the same way, with a fixed number of decimal places. Numbers which actually contain more decimal places than are displayed will be rounded up or down as necessary. The option asks you to type in the number of decimal places you want to be shown. It will not accept a value greater than 15.
- Units,Cells,E** numbers are displayed in exponential, or scientific notation. Again the displayed number is rounded up or down as necessary. The option asks you to type in the number of decimal places you want to be shown. It will not accept a value greater than 15.
- Units,Cells,P** this displays numbers as percentages so that, for example, the value 0.55 is displayed as 55%. The option asks you to type in the number of decimal places you want to be shown. It will not accept a value greater than 15.
- Units,Cells,I** numbers are shown as integers, or whole numbers. Any decimal fraction is stripped off and no rounding up or down takes place. You are given the option for negative values to be enclosed in brackets, rather than with a leading minus sign.
- Units,Cells,G** this is a general numeric format in which any of the previous formats is chosen, depending on the value of the number, to make best use of the space available in the cell.
- Units,Cells,M** numbers are displayed in fixed-point decimal format, with two decimal places and a leading currency symbol. This is, of course, designed for displaying monetary values. You are given the option for negative values to be enclosed in brackets, rather than with a leading minus sign.
- In the case of the Cells option you are finally asked to specify the range over which the change is to act. If you just press **ENTER**, only the current cell will be affected.
- You are not asked to specify a range if you selected the Defaults option. In this case the selected format will be used for all new cells, as they are created.
- WINDOW(W)** You use this command to control whether the display is a single window or is split into two windows which can be used to show two separate portions of the grid.

You are first asked to choose between a vertical (V) split, a horizontal (H) split, or to join (J) a split display back into a single window. If the window is initially split and you want to change from, say, a horizontal to a vertical split you must first join the two windows before making the new split.

If you choose to split the window then the split will occur at the column or row containing the cursor. You should therefore position the cursor at the point where you want the split to occur before making the split. Whole columns will always be displayed but if you have changed the width of some columns so that they are very narrow you may find that extra columns are shown. Each window in a vertical split will never be less than ten characters wide.

You then are given a further, optional, choice as to whether the two windows should move together (T) or separately (S). If you specify the T option, this means that any change in the position of one window, in the direction parallel to the split, will cause a corresponding change in the position of the other, so that a constant separation is maintained between them. Movements at right angles to the split are not related in this way. The S option allows the two windows to move around the surface of the grid independently. This optional choice will not, of course, be given in the case where you request joining to a single window.

HELP press F2	CUSROR press ←→→	DATA & FORMULA enter directly & press ENTER	TEXT type" followed by text & ENT	COMMANDS press F3 ESCAPE press ESC
PROMPTS press F2	GOTO cell press F5			

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
	L	M	N	O	P	Q	R
29							
30							
31							
32							
33							
34							
35							
36							

CURRENT CELL: A1 GRID USED A1:A1 MEMORY REMAINING: 100%
CURRENT CELL EMPTY

Figure 6.6 A Horizontally Split Window

HELP press F1	CURSOR press ←→→	DATA & FORMULA enter directly & press ENTER	TEXT type" fol followed by text & ENT	COMMANDS press F3 ESCAPE press ESC
PROMPTS press F2	GOTO cell press F5			

	A	B	C	D	E	F	G
1				44			
2				45			
3				46			
4				47			
5				48			
6				49			
7				50			
8				51			
9				52			
10				53			
11				54			
12				55			
13				56			
14				57			
15				58			
16				59			

CURRENT CELL: A1 GRID USED A1:A1 MEMORY REMAINING: 100%
CURRENT CELL EMPTY

Figure 6.7 A Vertically Split Window

The appearance of the display for a horizontal and a vertical split is shown in Figures 6.6 and 6.7 respectively.

XECUTE (X) This command is used to force a recalculation of all formulae appearing in the grid. A recalculation is normally performed automatically when you make any new entry in the grid. You will only need to use this command if you have switched off the automatic recalculation option by use the Design command or if you want to activate any askn() or asks() functions stored in the cells of the grid.

ZAP (Z) This command clears the entire contents of the grid and returns you to the beginning of ABACUS for a fresh start. Since this command is drastic (and irreversible) in its action, you will be asked to confirm your request. If you press **ESC** you will return to the command menu without any deletion taking place. You should press **ENTER** to confirm your wish to clear the grid.

6.9 FUNCTIONS

You can think of a function as a kind of recipe which converts a number of values, known as the function's Arguments into a different value, which is said to be the value that is returned by the function.

The functions provided by ABACUS may take three, two, one or no arguments. The arguments for a function are placed in brackets after its name. You must not leave a space between the name and the opening bracket, but spaces are allowed between items within the brackets. If a function takes more than one argument, the arguments are separated by commas. All functions must be followed by the brackets, even if they take no arguments. The presence of the brackets is a useful reminder that you are referring to a function. They allow you to distinguish between a label and a function, even if they have the same name.

In the descriptions of the functions:

n is either a literal number or a reference to a cell containing a numeric value,

text is either a literal text string (enclosed in quotes) or a reference to a cell containing text,

range is a grid range reference. The following functions are provided.

ABS(n) Returns the absolute value (that is, the value ignoring any minus sign) of the argument.

For example, abs(3) returns 3 and abs(-7) returns 7.

ASKN(text) This function is used for the input of numeric data. It displays the given text (which may be up to 40 characters in length and will normally end with a "?") as a prompt in the input line and waits for a reply to be typed in. The reply is placed in the current cell. Input will only be requested when recalculating the grid by use of the Xecute command, and is not asked for during an auto-calculate after each grid entry.

ASKT(text) the input of text strings. It displays the given text (which may be up to 40 characters in length and will normally end with a "?") as a prompt in the input line and waits for a reply to be typed in. The reply is placed in the current cell. Input will only be requested when recalculating the grid by use of the Xecute command, and is not asked for during an auto-calculate after each grid entry.

ATN(n) Returns the angle, in radians, whose tangent is n.

AVE(range) Returns the average of the numeric values contained in all the cells in the specified range. Empty cells and cells containing text are ignored in the calculation of the average. If there are no numeric cells in the range it will return a value of zero.

CHR(n) This function returns the ASCII character whose code is n. Characters with ASCII codes less than 32 have no effect on the screen, but may be sent to the printer if they are preceded by an ASCII null, ie, chr(0). For example, chr(0) + chr(13) passes the ASCII character for a carriage return to a printer. This is useful if your printer needs control code sequences to produce special effects - refer to your printer manual for any special codes that it needs.

You can send an "A" to the screen with chr(65).

CODE(text) This returns the ASCII value of the first character found in the specified text.

COL() Returns the number of the current column.

COS(n) Returns the cosine of the given (radian) angle.

COUNT (range) Returns the number of non-empty cells in the specified range. Both text and numeric cells are included in the count.

DATE() Returns today's date as a text string in the form:

"DD/MM/YYYY"

You must first have set the system clock, as described in the technical manual.

DEG(n) Takes an angle, measured in radians, and converts it to the same angle in degrees.

EXP(n) Returns the value of e (approximately 2.718) raised to the power n. The returned value will be in error if n lies outside the range from -87 to +88, since the result will then exceed the numeric range of ABACUS.

IF (expression, true,false) The value of the expression is calculated and used to determine which of the following two arguments should be returned. If the expression evaluates to 0 it is considered to be false and the "false" argument is returned. Any non-zero value for the expression is interpreted as being true and causes the "true" argument to be returned. The "true" and "false" arguments may be either text or numeric in nature. Thus all the following examples are valid uses of the function;

if(A1 = B1,"equal","not equal")
if(A1,1,0)

You can also mix a text and a numeric argument as in the following example. Try this one out if you are not sure how if() works.

A1 1
B1 0
C1 if(A1 or B1,"either",0)

You should see the word 'either' appearing in cell C1 since the first parameter of if() returns a non-zero (true) value if either cell A1 or cell B1 contains a non-zero value. If you change the contents of cell A1 to be zero then you will see a zero displayed in cell C1 since the condition will be false and the third argument will be returned.

INDEX(n1,n2) Returns the contents of the cell in row n1 and column n2.

INSTR (text1,text2) This finds the first occurrence of "text2" within "text1" and returns the position of the first character of "text2" in "text1". It will return a value of zero if no match is found. The match is case-dependent.

instr("January","Jan") returns 1
instr("January","an") returns 2
instr("January","AN") returns 0

INT(n) Returns the integer value of the number, by truncating at the decimal point. The truncation always operates towards zero. Thus;

int(3.7) returns 3
int(-4.8) returns -4

IRR(range,n) Calculates the Internal Rate of Return for the numeric data in the specified range, which may be either a row or a column.

The data in the range represents a cash flow for each of a series of periods, separated by n months. Negative values represent cash outlays and positive values represent cash returns.

The function returns the rate of interest necessary so that investment of your outlay would match the proposed returns.

The function is best explained by means of an example, using the same conventions as those in Chapter 5.

Suppose you are offered a return of twenty thousand pounds at the end of each of the next seven years, in return for an initial outlay of one hundred thousand pounds. Is this a good deal?

```
|A1| "flow
|A2| -100000
|A3| col = 20000 (rows 3 to 9)
```

We can refer to the range of the data by the label "flow" and the interval between successive periods is twelve months:

```
|C2| irr(flow,12) (rows 2 to 9)
```

The completed grid should look like Figure 6.8, showing that the internal rate of return is 9.1%. If you can invest your hundred thousand pounds at a higher rate of interest you should do so, and forget the deal.

	A	B	C
1	flow		
2	-100000.00		
3	20000.00		
4	20000.00		
5	20000.00		
6	20000.00		
7	20000.00		
8	20000.00		
9	20000.00		9.10

Figure 6.8 Internal Rate of Return.

Note that the first item in the range is counted as period zero, the next is period one, and so on. The function assumes that each amount is payable in full at the end of the relevant period.

- LEN(text)** Returns the number of characters in the specified text.
- N(n)** Returns the natural, or base e, logarithm of n. An error results if n is negative or zero, since logarithms are not defined in this range.
- LOOKUP(range,n)** This function implements a look-up table in the grid. Two tables of values are assumed to be present. The first table occupies the specified range (which can be in a row or a column). The second table runs parallel to the first, in the following row or column. For example, if the first table is in column G, from G10 to G25, the second will be assumed to be from H10 to H25. Every entry in the first table should have a corresponding entry in the second. The first table is searched for the largest value that is less than or equal to n. The function returns the corresponding entry from the second table. Note that it is assumed, for the correct operation of this function, that both tables contain numeric values, and that those in the first table are arranged in ascending order.

The first value in the first table is a dummy. It must be less than the second value, which is the lower limit for the table lookup process. It is otherwise ignored. The first value in the second table is the default value. It will be the value returned if lookup() is called with any number less than the lower limit.
- LOWER(text)** Converts the specified text to lower case.
- MAX(range)** Returns the largest numeric value found in the cells in the specified range. If there are no numeric cells in the range the function will return the smallest possible number (-1.7 E + 38).

MIN(range) Returns the smallest numeric value found in the cells within the specified range. If there are no numeric cells in the range the function will return the largest possible number ($+1.7 \text{ E} + 38$).

MONTH(n) Returns, as text, the name of month n.

For example month(3) returns the text "March".

If an argument larger than 12 is used, it is replaced by the remainder after division by 12 so that, for example, month(13) and month(1) will both give the result "January".

NPV(range, n1,n2) Calculates the Net Present Value for the cash flow data in the specified range. The percentage interest rate is n1 (n1 = 14 represents a 14% rate). The data is assumed to refer to a series of periods, separated by equal intervals of n2 months.

The net present value is the amount of money required now to produce a given future cash flow, assuming an interest rate. As with irr(), it is best explained by means of an example.

Suppose you are given the opportunity to buy, for a single payment of seventy thousand pounds, a ten-year lease on a shop which is currently producing a yearly net income of ten thousand pounds. You expect the income to increase by 10% per year. If you did not buy the shop your seventy thousand pounds would earn 14% interest. What should you do?

You should calculate the net present value of the income and compare it with the sum you are asked to pay

A1	"flow
A2	0
A3	10000
A4	col = a3*1.1 (rows 4 to 12)
A14	npv(flow,14,12) (rows 2 to 12)

The result is shown in Figure 6.9

	A	B
1	flow	
2	0.00	
3	10000.00	
4	11000.00	
5	12100.00	
6	13310.00	
7	14641.00	
8	16105.10	
9	17715.61	
10	19487.17	
11	21435.89	
12	23579.48	
13		
14	75088.51	

Figure 6.9 Net Present Value - Method 1

The net present value (in cell A14) of the cash flow from the shop is more than the asking price, so you should go ahead.

The first item in the list is for period zero, the second is for period one, and so on. This is consistent with the assumption, made by the function, that the returns are received at the end of each period. You therefore have to wait for one period before you obtain any return on your investment. In a real situation of this type you would probably work on a monthly basis, rather than on twelve month periods.

An alternative way to tackle the problem is to include your payment as a (negative) cash flow as the period zero cash flow. This is shown in Figure 6.10. The only difference between this example and the previous one is:

|A2| -70000

In this form a positive result (in cell A14) shows that the deal would be profitable.

	A	B
1	flow	
2	-70000.00	
3	10000.00	
4	11000.00	
5	12100.00	
6	13310.00	
7	14641.00	
8	16105.10	
9	17715.61	
10	19487.17	
11	21435.89	
12	23579.48	
13		
14	5088.51	

Figure 6.10 Net Present Value - Method 2

PI()	Returns the value of the mathematical constant pi.
RAD(n)	Takes an angle, measured in degrees, and converts it to the same angle in radians.
REPT(text,n)	This function will fill the current cell with n copies of the given text, which may be up to 255 characters in length. For example, rept("***,5) will put five asterisks in the current cell rept("abc",3) makes three repetitions of "abc".
ROW()	Returns the number of the current row.
SGN(n)	Returns + 1, -1, or 0, depending on whether the argument is positive, negative or zero.
SIN(n)	Returns the value of the sine of the specified (radian) angle.
STR (n,type,dp)	Converts a number, n, to the equivalent text string. The 'type' parameter indicates the form of the converted string as follows: <ul style="list-style-type: none"> 0 decimal (floating point) 1 exponential, or scientific, notation 2 integer. 3 general format 4 monetary format 5 percentage <p>The third parameter indicates the number of figures after the decimal point in the converted string. It should always be specified, although its value is ignored for integer, general and monetary formats.</p>
SQR(n)	Returns the square root of the number n, which must not be negative.
SUM(range)	Returns the sum of all the numeric values within the specified range. Empty cells and cells containing text are ignored.
TAN(n)	Returns the tangent of the specified (radian) angle.
TIME()	Returns, as text, the time of day in the format HH:MM:SS. You must first have set the system clock, as described in the technical manual.

PROVISIONAL

- UPPER(text) Converts the specified string to upper case.
- VAL(text) Val converts the text to its equivalent numeric value. It will only convert text composed of valid numeric characters and the conversion will stop at the first character that can not be interpreted as a digit. For example, val("1.1ABC") will return the value 1.1, and val("ABC") will return 0.0
- WIDTH() Returns the width, in character space

PROVISIONAL

(

)

(

)

sinclair

QL

QL Archive

ARCHIVE is an intelligent database which you can use for any type of information retrieval application, from a card index to a full multi-file relational database. The information can be presented in a customised screen design or in any printed format.

When you have just loaded ARCHIVE it is in the *keyboard interpreter* mode. This means that it will accept what you type at the keyboard and try to interpret and execute it as a known command.

ARCHIVE has a comprehensive set of database-related commands which allow you to make use of its facilities from the moment that you load it. Although the commands form a powerful programming language for the construction of specialised applications, you can create a useful card index in a few minutes, directly from the keyboard.

As soon as you have created a file you can use the available commands to make sophisticated searches or selections from the file, sort the records on any number of key fields and display the results.

At all times you are guided by an informative set of prompt messages which never leave you in any doubt about what your options are or what you are expected to do. If you require further information you can always use the Help files. These contain full details about all the options. You may ask for Help at any stage, no matter what you are doing, and will automatically be given the information that is most relevant to your current needs.

ARCHIVE has a *pyramidal* structure, in that the most commonly-needed commands and options are available immediately. Beneath the surface are many sophisticated options, each of which has a series of sub-levels. The full power of ARCHIVE becomes apparent as you become more familiar with it and dig more deeply into the pyramid.

The real power of ARCHIVE comes when you write your own procedures in the database language. You can create a named procedure to do exactly what you want and then use it as an additional command, in exactly the same way as you use the commands provided with ARCHIVE. Alternatively you can write a complete program that runs independently of the normal commands.

The ARCHIVE database language has a syntax similar to SuperBASIC and is therefore simple to use. Unlike BASIC, however, it is based on program units known as *procedures* which lead naturally to the creation of correct and readable programs.

The mechanics of writing and modifying a program are aided by a full *procedure editor* which, together with the *input line editor* (which is available at all times), make editing a simple and painless task.

The commands include simple and fast sorting, searching and selection of records, together with many string manipulation operators and fast, accurate arithmetic.

The data files themselves use variable length fields and records. Not only does this lead to the most efficient use of available memory and disk space, but, also to simplified file creation. You never need to decide in advance how large a record needs to be.

ARCHIVE is provided with a number of working examples. You can either use them immediately or you can make simple modifications to match them to your exact needs. Try out the examples to see some of the range of things that can be done. All the programs are fully documented in this manual and contain many general purpose procedures which you could include in your own programs.

CHAPTER 1
ABOUT
ARCHIVE

THE ARCHIVE FOR

[Faint, illegible text covering the main body of the page]

C

C

C

C

CHAPTER 2 BASIC OPERATIONS

2.1 LOADING ARCHIVE

When you switch on the computer it will only respond to commands in SuperBASIC. You will have to load ARCHIVE from its Microdrive cartridge. You will normally do so by inserting the ARCHIVE cartridge in drive 1 (the left hand drive) and then typing:

LGO MDV1__ARCHIVE ENTER

After a few seconds the screen will show the message:

**ARCHIVE - Copyright Psion Ltd 1983
Press any key to start**

You should then press any key on the keyboard to start ARCHIVE.

2.2 GENERAL APPEARANCE

When you have loaded ARCHIVE the display on the screen should look like that shown in Figure 2.1.

The display is divided into three main areas, known as the control area, the display area and the work area.

The control area occupies the top four lines of the display and shows the options that you have. Its contents will change from time to time, depending on what you are doing, and also gives confirmation of your choices. At the moment it shows that you have three main choices which are to:

- press the HELP key,
- change the prompts,
- type in a command,
- obtain a further list of commands,
- press ESC.

HELP press F1	COMMANDS delete	create display	look back	open alter	close find	COMMANDS press F3
PROMPTS press F2	first type command &	insert press	last ENTER	next (F3 for more)	quit	ESCAPE press ESC

Figure 2.1 shows the main display area. The control area is at the top of the screen and contains the following options:

- press the HELP key,
- change the prompts,
- type in a command,
- obtain a further list of commands,
- press ESC.

Figure 2.1 The Main Display.

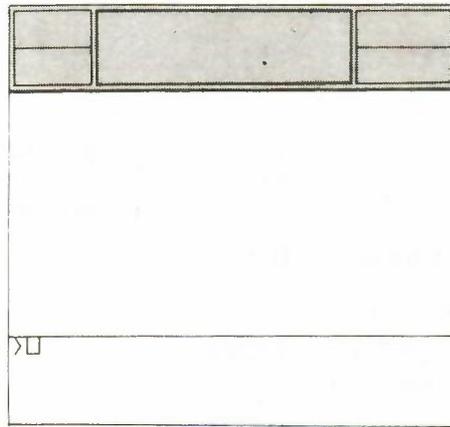


Figure 2.2 The Control Area.

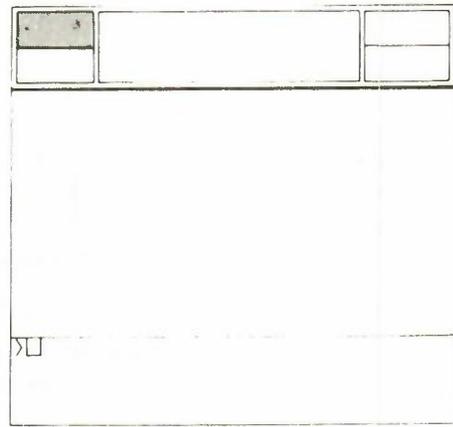


Figure 2.3 Help.

2.3 HELP

The first option, displayed at the left of the control area, shows that you may ask for help by pressing the Help key. This is function key one (**F1**). Regardless of any other changes in the control area display, the Help option will always be shown. This indicates that the Help facility is always available, no matter what you are doing.

Try pressing the Help key now. When you do, the current display will disappear, to be replaced by one giving a complete list of the commands. You may ask for further information about any one of these commands by typing in its name and pressing **ENTER**. You do not need to type the whole of a name; you only need to type in the first few letters - enough to distinguish it from any of the other names in the list.

When, after typing in one of these names, you press **ENTER** you will find that further, more detailed, information is shown about the command you have selected, and another list of sub-topics will be shown. You may then select one of these sub-topics by typing in the first few letters of its name, as described before. You may continue this process until no further information is available.

At any stage you may return to the previous screen by simply pressing **ENTER**. Repeatedly pressing **ENTER** will eventually take you back to the main display, with the control, display and work areas. At this point you will have left the Help facility and will have been returned to the exact state before you pressed the Help key. A faster way to return from Help is to press **ESC**. This will return you from any point with Help, back to the state from which it was first called.

Try using the Help facility to examine some of the pathways through the information. Don't worry if you do not understand some of the information that is shown - it will make sense when you actually need to use it. All you need to do at the moment is to become familiar with the way in which the Help facility is used. When you have finished, press **ESC** to return to the main display.

It must be emphasized that Help is always available, at any time. Whenever you are not sure what you should be doing, just press the Help key even if you are, for example, in the middle of typing in numbers or text as part of a command. You will not always start at the same point in Help, but will be presented with the information most relevant to what you were doing when you pressed the Help key. When you have found the information you need and leave Help (by use of the **ESC** or **ENTER** keys) you will always be returned to the exact point from which you started, as though there had been no interruption. Use the Help key as often as you like - it is there to assist you and will usually be the quickest and simplest way of solving your problems.

2.4 THE PROMPTS

In addition to showing your options, the control area highlights your choice and, when necessary, suggests what you should do. These aids to using ARCHIVE are known as prompt messages or just *prompts*.

You can switch off the display of the control area and the prompts it contains by pressing function key 2 (**F2**).

PROVISIONAL

Try pressing function key 2 (F2). The display will be redrawn without the control area, leaving more room for your work. ARCHIVE works in exactly the same way, whether the prompts in the control area are displayed or not - you are free to choose either option. You will probably find it most useful to work on your programs and files with the prompts in the control area visible, and then turn them off to give you more room for looking at the finished result. The example of Chapter 11 shows an application which does not display the prompts.

You can restore the control area display at any time by pressing F2 again.

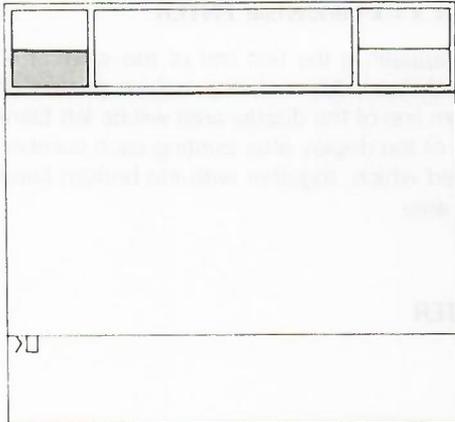


Figure 2.4 Prompts.

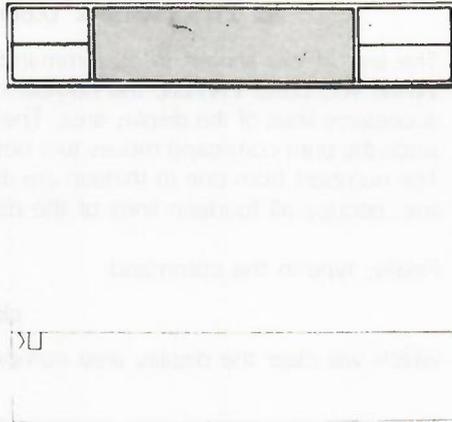


Figure 2.5 The Commands

Another option available from the main display is to use a command. A list of commands appears at the centre of the control area. You can use any of these commands by typing its name and pressing **ENTER**. If the command needs some further text (such as a name) you will be asked to type it in.

2.5 USING THE COMMANDS

There are many commands available and they are described in the rest of this manual. There are too many for them all to be displayed in the control area at the same time, so only the most commonly-needed ones are shown.

2.6 FURTHER COMMANDS

You can display a further set by pressing function key 3 (F3). Each time you press F3 the contents of the control area change to show a further set of commands. This will continue until all the commands have been displayed - then the next press of F3 will display the original list.

You can use any of the commands, even if its name does not appear in the current display in the control area.

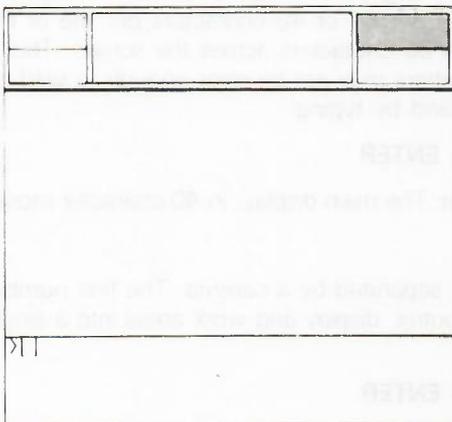


Figure 2.6 Further Commands.

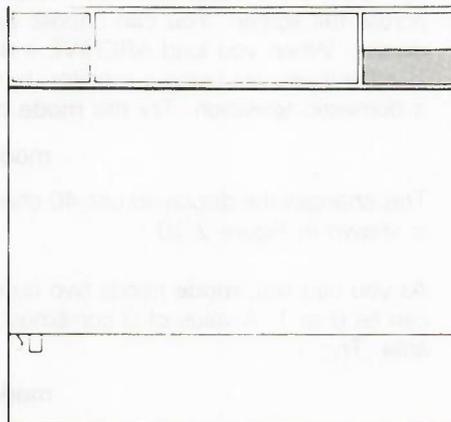


Figure 2.7 Escape.

The **ESC** key is used to cancel an action, or to go back to the main display. We have already seen how it is used in this way to leave Help.

2.7 ESCAPE

2.8 THE DISPLAY AND WORK AREAS

The display area is, as its name suggests, where all information produced by ARCHIVE is displayed.

The work area uses the bottom five lines of the screen. All commands that you type in, together with any error messages, are shown in this region.

These two areas almost invariably work together, since commands typed into the work area produce their results in the display area. As an example, type in the following sequence of commands, exactly as they are shown below.

```
let x = 13:while x > 0:print x:let x = x-1:endwhile ENTER
```

The text of this sequence of commands will appear in the first line of the work area. When you press **ENTER**, the numbers from thirteen down to one will be printed on successive lines of the display area. The bottom line of the display area will be left blank since the print command moves to a new line of the display after printing each number. The numbers from one to thirteen are displayed which, together with the bottom blank line, occupy all fourteen lines of the display area.

Finally, type in the command:

```
cls ENTER
```

which will clear the display area completely.

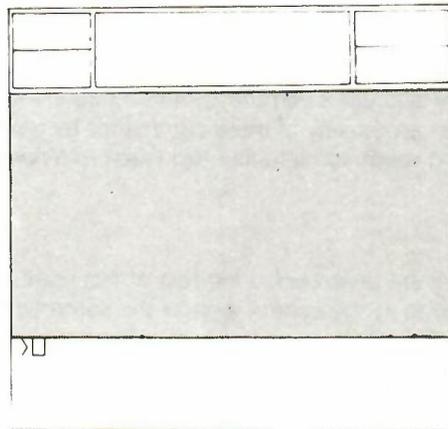


Figure 2.8 The Display Area.

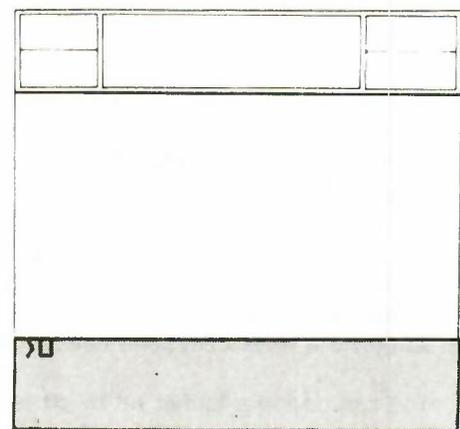


Figure 2.9 The Work Area.

2.9 THE MODE COMMAND

You have the option of combining the control, display and work areas into a single area by means of the **mode** command. **Mode** also changes the number of characters displayed across the screen. You can choose between 80, 64 or 40 characters per line of the display. When you load ARCHIVE it displays 80 characters across the screen. This is suitable if you are using a monitor, but the letters may not be clear enough to read on a domestic television. Try the **mode** command by typing:

```
mode 1,4 ENTER
```

This changes the display to use 40 characters. The main display, in 40 character mode, is shown in Figure 2.10.

As you can see, **mode** needs two numbers, separated by a comma. The first number can be 0 or 1. A value of 0 combines the control, display and work areas into a single area. Try:

```
mode 0,8 ENTER
```

If you then type in the previous example again you will see that your input from the keyboard and the output numbers all share the whole of the screen. A value of 1 separates them into three distinct areas. Try:

```
mode 1,8 ENTER
```

which will restore the display to be as it was when you loaded ARCHIVE.

PROVISIONAL

The second number can be 4, 6 or 8 and selects 40, 64 or 80 characters across the display. Try some different combinations to see the effect on the display. Finish with a command that leaves the screen divided into its three areas, but choose the number of characters that gives a clear display on your television or monitor.

COMMANDS	create	look	open	close
delete	display	back	alter	find
first	insert	last	next	quit
type command & press ENTER (F3 for more)				
HELP F1	PROMPT F2	COMMANDS F3	ESCAPE ESC	
⌘				

Figure 2.10 The Main Display in 40 Character Mode

)

)

)

)

CHAPTER 3 A CARD INDEX

3.1 FILES RECORDS AND FIELDS

3.1.1 Introduction

An ARCHIVE file behaves rather like a card index. A real card index consists of a box containing a set of record cards. Each card has various items of information written on it. For such a card index to be useful, there have to be rules to determine where each piece of information is written on the card. Suppose, for example, that we have a name and address index. You would normally write the person's name across the top of the card, followed by the address and telephone number (if any). It would be very difficult to use if some cards had the name written at the top and others had it written near the bottom. You would normally expect to be able to use the index by flipping through the cards, reading only the top line, until you found the name you were looking for.

If you had two sets of record cards, such as a name and address records and a set of stock records, you would not normally store them both in the same box. You would use two boxes and label them, for example, "Customer Records" and "Stock Records".

The card index system contains most of the ideas necessary for understanding the working of an ARCHIVE *file*. A file is like the card index box and is given a name to identify it. The file is made up of a collection of *records*, each of which serves the same purpose as a record card. A file, then, is simply a collection of related records.

The records within a file all contain the same type of information but each record is different from its neighbours. In a customer record file, for example, each record would contain the name, address and telephone number of a particular customer, together with details of his previous dealings with your company, whether he is entitled to any discount, his credit limit, and so on.

As in a card index the information in each record is organised in a regular way. Some of this information might be placed on a record card as shown in Figure xxx, where specified areas of the card are reserved for each of the pieces of information. A record in an ARCHIVE file is organised in the same way. Each item is stored in a separate region of the record, known as a *field*. A record in a customer file, such as that described above, would contain a name field, an address field, a discount field and so on.

If this were the whole story there would be little point in using an ARCHIVE file in preference to a physical card index. There are, however, many advantages when you use computerised records. A customer record card index would normally be arranged in alphabetical order of customer names which makes it very efficient for finding the information about a particular customer. Suppose, however, you want to send a circular letter to all your customers who have not placed an order with you during the last six months. It would be a very tedious task to go through the entire contents of a card index to compile such a list. In ARCHIVE you can make such a search by using a few simple commands.

Furthermore, it is easy to arrange for a set of address labels to be printed at the same time. This simple example shows that you can save a great deal of time and effort by using ARCHIVE to store and manipulate your records.

ARCHIVE is provided with a simple card index. You can use it by typing:

```
run "cardindex" ENTER
```

You are asked if you want to make any alterations to the file or if you just want to examine its contents. You should reply by pressing either the C key (if you want to make changes) or the E key (if you just want to examine the records). ARCHIVE then opens a file called "index" which contains the address file records.

It then shows you the first record in the file, together with a short list of additional commands which are described in the next section. You can also use any of the standard commands that are described in Chapters 4 and 5.

BYE - closes the file and saves any modifications you may have made. It then clears the computer's memory and returns you to ARCHIVE, as though you had just loaded it.

3.1.2 Files

3.1.3 Records

3.1.4 Fields

3.1.5 ARCHIVE Files

3.2 THE CARD INDEX APPLICATION

3.2.1 The Additional Commands

HELP - clears the display of your file records and displays information on how to use the card index. It works like the Help that you get when you press **F1**. You can get further information about any of the topics listed at the bottom of the help screen by pressing the key corresponding to its first letter. Press **ENTER** to go back to the previous screen, or **ESC** to go back to the display of your records.

SCAN - scans through the whole file, pausing at each record for you to look at its contents. If you press any key, except **ESC**, while the scan is in progress the scan will pause at the current record until you press the space bar. You can abort the listing at any time by pressing **ESC**.

CHAPTER 4 EXAMINING A FILE

In Chapter 3 we used the card index application that is provided with ARCHIVE. It has a specially-designed screen display, together with a small set of commands tailored to that particular application. In this chapter we shall start to look at the more general commands available in the database language itself. Later chapters will explain how you can use these commands to produce a complete application of your own. We shall use the card index file as an example, since by now you should be familiar with its contents. You should try out the commands on the card index to gain familiarity with how they work.

The card index is designed so that you can use most of ARCHIVE's file commands described in this, and the following, chapter on the file that is opened when you run "cardindex", as described in Chapter 3. (The major exceptions are **look** and **open**, since the file is opened automatically.)

The examples in this chapter assume that you are working with the "index" file provided with the card index application.

Before you can use a database file you have to open it. Opening a file makes its contents available to you, but you must first decide if you only want to do read the file contents or want to be able to change the file contents as well as read them. There are two commands which you can use to open a file; **look** and **open**.

The **look** command **opens** a file in such a way that you may only read its contents - any attempt to write new data to the file will not be permitted. The **open** command allows you both to read and to write new information to the file. In both cases the file is opened by making a copy of its contents from the Microdrive cartridge into the computer's memory. When you close a file opened with the **open** command the old version of the file on the Microdrive cartridge is replaced by the copy that is currently in the computer's memory. When you close a file that was opened by the **look** command the version in memory is discarded and no changes are made to the version on the Microdrive cartridge.

Let us first use the **look** command to examine the contents of the card index file. First you should type in the following command:

```
look"index" ENTER
```

ARCHIVE is working in interpreter mode, rather like when you type in a SuperBASIC command without starting with a line number. The command that you have typed is in the work area at the bottom of the screen and is immediately interpreted and obeyed. Nothing much appears to happen, except that the Microdrive switches on for a few seconds and the cursor moves to the next line of the work area with a second prompt sign. The contents of the card index file, however, have now been copied into memory and are available for reading.

The rest of the examples in this chapter assume that you have opened the "index" file in the way just described. If you use the **run** "cardindex" method of Chapter 3 there will be only one difference. The display of the records will use the special screen layout until you first use the **display** command which is described in the next section.

To make part of the file visible you should now type:

```
display ENTER
```

when the entire contents of the first record of the file will be written into the display area (don't forget to press **ENTER**). The form of the display is quite different from that used in the card index application, described in Chapter 4. It is simply a list of the contents of each field of the record, one per line, and each is preceded by the name of the field.

Each full field variable name is composed of two parts. The first part, which in this case is the word "main", is shown on the first line. It is known as the *logical file name* and it identifies which file is being used. Its main purpose is to allow you to use more than one open file at a time and to be able to refer to one file or another. As we shall see later, you can choose this name yourself. If you do not make your own choice for a name (as in this case) ARCHIVE will use the name "main".

4.1 INTRODUCTION

4.2 OPENING A FILE

4.3 DISPLAYING A RECORD

The second part of the variable name for each field is the field name itself. If the field contains text the name must end with a dollar sign; if there is no dollar sign at the end of the name the field is used to hold a numeric value.

If you loaded the card index file with the **run** command you will have started with the first file record displayed in its own special layout. As soon as you use the **display** command this layout changes to the one just described.

The **display** command always uses this layout, regardless of any special screen layout that you have designed (see Chapter 7). Your own design is replaced by the one used by **display**. If, after using **display**, you want to use your own design again you will have to load it again from the Microdrive cartridge. You do this with the **load** command, as described in Section 7.4.

The reason for this behaviour is that it allows you to display your file records in a simple way, without first having to design a display screen.

4.4 EXAMINING OTHER RECORDS

Having looked at the first record of the card index, you may want to move on to the following record. You do this very simply by using the **next** command, which is again interpreted and obeyed immediately. It shows you the next record in the file. Note that there is no need to use the **display** command again. In interpretive mode the display area is continuously updated to show the contents of the current record. You can use the **next** command to step through the file, record by record until you reach the end of the file (it will not pass the last record).

There are three other related commands which you can use to control which record of the file is displayed. These are:

- back** - which displays the previous record,
- first** - which displays the first record,
- last** - which shows the last record of the file

Try using these commands to move around the file, displaying any record you like.

4.5 CLOSING A FILE

When you have finished examining the records of the file you should **close** it by using the close command. Alternatively you may use the **quit** command which will close all open files before returning to SuperBASIC.

If you have made any modifications to the file (as described in the following chapter) this will make sure that the changes are recorded on the Microdrive cartridge. Since we have not made any changes, all that will happen is that the copy of the file in memory will be discarded, leaving the memory free for further use.

During the time a file is open, ARCHIVE may create another, temporary, file for its own use. When you close your file ARCHIVE will make sure that any such temporary file is erased. If you do not close your file properly (for example, if you just turn off the computer when you have finished) any temporary file will be left on your Microdrive cartridge. These files are all named PSITEMPx, where x may be a number from 0 to 9. If you see such a file on one of your Microdrive cartridges it is a sure sign that you did not close a file at some time. You can safely delete any such file whenever you see it.

4.6 SEARCHING A FILE

4.6.1 Introduction

The commands described in Section 4.1 will allow you to examine any record within a file and search through the file record by record to find the information you want.

This technique is quite suitable for searching through a file with only a small number of records but would be very inefficient on a large file. The commands described in this section will allow you to make such searches automatically, selecting any one or more records that you require.

4.6.2 Find

The first and simplest of these commands is **find**. This will search through the records of a file looking for the first occurrence of a specified piece of text in any of the text fields.

If you do not have the "index" file open, use the look command to open it, and use **display** to show the first record. Now type:

```
find"smith" ENTER
```

PROVISIONAL

When you press **ENTER** there will be a slight pause and then the first record containing the word "smith" in any of its text fields will be displayed. Note that this search is independent of whether the text is in upper or lower case and will therefore find "Smith", "SMITH" or "smith".

If the first record that is found containing this text is not the one that you want, you can find the next occurrence by typing:

```
continue ENTER
```

The **continue** command will repeat the previous search, looking for the next occurrence of the text in any text field of the following records. If, at any stage, no match is found in the remaining records of the file the display will keep the last record shown.

4.6.3 Continue

A second method of locating a particular record is to use the **search** command. **Search** must be followed by a condition which results in a numeric value. The records of the file are scanned for the first one in which this number is non-zero. For example:

4.6.4 Search

```
search name$="Jones" and city$="London" ENTER
```

will find the first record in the file which matches both conditions. Unlike the **find** command, you can specify particular fields within the records and only these fields will be searched for the match. Again you can use the **continue** command to find the next occurrence, if the first is not the one which you want.

In many cases, you may want to look a sub group of the records within a file. Suppose, for example, you only want to look at the details of people who live in London. You can use the **select** command to pick out from the file all those records which satisfy a certain condition and could, in the present case, use the command:

4.6.5 Select

```
select city$="London". ENTER
```

This will pick out only those records which match the condition, and the file will then behave as though only those selected records are present.

Try this command on the card index file to see how it works. First type:

```
print count() ENTER
```

which will tell you how many records there are in the file. Then type:

```
select city$="London" ENTER  
print count() ENTER
```

and you will see how many records are left in the file. The records removed from the file are still held in the computer's memory. You can restore them to the file at any time by using the **reset** command. Type:

```
reset
```

and print the value of count() again, to check that the file has been restored to its original state.

The records of the file may not always be in the correct order for your purpose and you will then want to be able to sort them into the order which you want. Suppose you want to sort the records of your card index file alphabetically by city. You can do this by using the **order** command as follows:

```
order city$a ENTER
```

This specifies that you want to sort the file in ascending order of the contents of the city\$ field, which becomes the *sort key* for the file. You can specify more than one sort key by giving a list of keys after the **order** command. For each of the keys you must specify whether the sort is to be in ascending or descending order. The following command, for example, will sort the file into ascending (alphabetic) order by surname and descending order (reverse alphabetic) by city.

4.7 SORTING A FILE

```
order surname$a,city$d ENTER
```

Note that a semicolon separates each field name from the "a" or "d" that specifies ascending or descending order, but that each pair (of field name and letter) is separated from the next by a comma.

When more than one field is specified for sorting purposes, the first is known as the primary sort key and the following ones are secondary sort keys. The records are initially sorted on their primary sort keys but all records with equal primary sort keys are then

ordered according to the next key in the list. If records exist which are equal in respect of both of these two keys they are ordered according to the contents of the third key field, and so on.

4.8 LOCATE

When a file has been sorted, you can use the **locate** command to make any particular record the current record in the file. **Locate** is followed by an expression, and its action is to find the first record whose primary sort key matches the given expression. This record becomes the current record in the file. For example, if the card index file has been sorted as described in the last example, the command:

```
locate "Smith" ENTER
```

sets on the first record in the sorted file which has the word "Smith" as surname.

You can locate a record on the contents of more than one sort key by using **locate** with multiple expressions, separated by commas. For example,

```
locate "Smith","London" ENTER
```

will find the first record for someone named Smith, living in London.

The only restriction on the number of expressions that you can use with **locate** is the number of fields on which the file has been sorted.

4.9 LOGICAL FILE NAMES

As was mentioned in Section 4.3, ARCHIVE will automatically supply a logical file name of "main" when you open a file. If you want to use your own logical file name you must specify it at the time that you open a file.

Suppose, for example, that you want to examine the card index file in the way described in Section 4.4, but would like to use the logical file name "card" instead of "main". You can do this by using the **look** command in the following way:

```
look "index" as "card" ENTER
```

This opens the file for reading only, as described earlier, but with a logical file name "card" instead of the default name "main". You can display a record from a particular file by giving its logical file name after the **display** command:

```
display "card" ENTER
```

when you will see the same display of the first record of the file, except that the logical file name starting each of the field variable names will be the word "card". You can add the logical file name to all the commands that were described in the previous section in the same way. For example the command:

```
next "card" ENTER
```

will display the following record of the file whose logical file name is "card". This will mainly be of use when you are using more than one file. When you only have a single file it is not necessary to give the logical file name, even if you have specified one. All these commands will act on the current file, regardless of its logical file name if they are used without the optional logical file name. If you only have one file open it must always be the current file. The idea of a current file and the use of more than one file are described in Chapter 11.

5.1 OPENING A FILE

If you **open** a file with the open command, instead of the **look** command (which was described in Chapter 4) you will be able to write to the file as well as reading from it. This means that any additions, deletions or modifications will make a permanent change to the copy on the Microdrive cartridge when you close the file. As with **look**, you have the option of specifying your own logical file name, as shown in the following examples.

```
open "index" ENTER
```

```
open "index" as "card" ENTER
```

You must not use the commands described in this chapter with a file opened with **look**. If you attempt to do so you are given an error message to indicate that you are attempting to modify the file.

All the commands allow you the option to specify a logical file name. As before, the main use of this option is when you have more than one file open at the same time.

The easiest way to add one or more records to an open file is by using the **insert** command. When you use this command you will be asked to type in the contents of each field of the new record, one by one.

5.2 INSERT

When you have typed in the contents of a field you should press **TABULATE** (not **ENTER**) to move on to the next field. You can press **TABULATE** (or **SHIFT** and **TABULATE** together, to move back to the previous field) at any time and you can make as many changes as you like to the contents of the fields.

If, after you have typed in the contents of the last field of the record, you press **TABULATE** again **ARCHIVE** will move back to the first field of the record. You can then, if necessary, make further changes to the contents of any field.

When the whole record is completed to your satisfaction you can press **ENTER** to insert the new record in your file. You will immediately be asked to type in the contents of the first field of another new record. You can add as many new records as you want. When you have no more records to insert you can leave the command by pressing **ESC**.

A second method of adding a record to the file is with the **append** command. This adds a new record whose fields are filled by the current contents of all the field variables for the records of that file.

Before using **append** you should therefore give the field variables the values you want them to have, by using the **let** command, eg:

```
let fname$ = "John" ENTER
let surname$ = "Smith" ENTER
etc.
```

If you then type:

```
append ENTER
```

the new record will be added to the file.

As with **insert**, the position in the file where the new record is inserted will depend on whether the file has previously been sorted or not.

If you want to remove a record from the file you can do so by using the **delete** command. **Delete** removes the current record (the one shown by the **display** command) from the file. All you have to do to remove a particular record is to display it, and, having made certain that it is the correct one, type:

5.3 DELETE

```
delete ENTER
```

5.4 CHANGING A RECORD

It is also simple to modify the contents of any or all of the fields within an existing record. First you make the record which you wish to change to be the current record (by displaying it). You then change the contents of the field variables until the displayed record is exactly as you want and then type in the command:

```
update ENTER
```

Suppose, for example, that you discover that John Smith spells his surname with a y. You can modify his record by locating it by, say,

```
find "Smith" ENTER  
display ENTER
```

You can then use the **let** command to change the contents of the surname\$ field and put this change into the record by typing:

```
let surname$ = "Smyth" ENTER  
update ENTER
```

Remember that you must close the file with the **close** or the **quit** command, before switching off the computer, so that your changes are stored on the Microdrive cartridge.

Suppose you want to use ARCHIVE to make a catalogue of your books. To do this, you will have to create a new file called, for example, "books". The first thing to do when creating a file is to decide what information it is going to contain; that is, what fields you will use in each record. In this case you will obviously need to record the author, title and subject; you may also like to include other details, such as the type (fiction or non fiction), ISBN (International Standard Book Number) shelf location, a brief description and so on. In this example we shall simply use three text fields to contain the author, title and subject and one numeric field which will be used to hold the ISBN.

6.1 INTRODUCTION

You create a file by using the **create** command. You must specify the name of the file to be created and the names of the fields to be used in each record (the names of fields which are to hold text strings must end with a dollar sign). When you have finished defining the fields of a record you end the **create** command with **endcreate**. You can create a simple book catalogue file as described above by typing in the following sequence. (From now on we shall not always show the **ENTER** that you must use at the end of every line of input.)

6.2 CREATE

```
create "books"
author$
title$
subject$
isbn
endcreate
```

Note that you do not have to type in the final **endcreate** command. You can do so if you want, but you can end the creation in the file simply by pressing **ENTER** on a blank input line. (You must, of course, include **endcreate** if you use **create** to create a file from within an ARCHIVE program.)

One of the great advantages of ARCHIVE is that you do not have to decide in advance how much memory is to be reserved for each field within a record. If you had to decide the length of each field at the time the file was created you would have to allow for the longest possible record that you would expect to appear. This would mean that a record containing less than this maximum amount of information would have a lot of wasted space, reducing the number of records that you could keep. ARCHIVE allows each field within a record to be of variable length; the space used for each field is automatically adjusted to match the amount of information stored in it. This relieves you of the responsibility of having to decide in advance how much space should be reserved for each field and also ensures that the computer's memory is used to maximum efficiency.

At some stage in the future you may find that a file which you have created contains a field which you do not use or, more likely, that you need an additional field which you forgot to specify when you created the file. In this case you will have to create a new file containing the new set of fields and copy the contents of the old file into the new one. You will therefore need to have two files open at the same time and the method for doing this is described in Chapter II. As with **open** and **look** you can, when using the **create** command, specify a logical file name to identify one of a number of open files.

Although it is quite straightforward to change the fields used in a file, it is worth taking a little care in deciding what fields to use before you create the file.

When you have created a file as described above it is in the computer's memory, as a file which is open for both reading and writing, but as yet it contains no records. You can add records to the file by the methods described in Chapter 6, that is by using the **append** or **insert** commands. The easiest way is to use **insert** as in the following example, which uses the book catalogue file that we have just created.

The display area will now appear as shown in Figure 6.1, with the names of the fields listed out.

6.3 ADDING RECORDS

insert

```

logical name : main
author$      :
title$       :
subject$     :
isbn         :
    
```

Figure 6.1 The Display for insert.

All you have to do is to type in the contents of each field, ending each one by pressing **TABULATE** (not **ENTER**). After typing:

```

Bloggs, J TABULATE
A Boring Manual TABULATE
Cannon Making TABULATE
1234567 TABULATE
    
```

the display area should appear as in Figure 6.2.

```

logical name : main
author$      : Bloggs, J
title$       : A Boring Manual
subject$     : Cannon Making
isbn         : 1234567
    
```

Figure 6.2 Inserting a record.

You can then insert the record into the "books" file by pressing **ENTER**. The display area will then return to the appearance of Figure 6.1, ready for inserting another record. When you have finished inserting records into the file you should press **ESC** to leave **insert**.

You must remember, before you switch off the computer, to make sure that the file contents are saved on the Microdrive cartridge by using the **close** or **quit** commands.

6.4 LOGICAL FILE NAMES

Suppose we want to create a file called "book2" and use a logical file name "second". You can do this by writing the first line of the Create command as:

```
create "book2" as "second"
```

The rest of the command follows exactly as before. If you do not specify the logical file name, ARCHIVE uses the logical file name "main", as you will have seen from the earlier example.

When you use the **display** command on a file that you have created yourself, for example the book catalogue of Chapter 7, the records are shown in a simple form. The logical file name is shown at the top of the screen, followed by a list of all the field names in a record of the current file. The current value of each variable is displayed to the right of its field name.

You will have seen that the example application provided with ARCHIVE does not use this simple form of display but has a specially designed layout for the information within a record.

It is very simple to produce a screen layout of your own and you may like to try producing one for the book catalogue described in Chapter 6, using the one provided as a model.

When you create a screen layout, anything that appears in the display area will be part of the screen. It is probably therefore a good idea to start by clearing the display area entirely with the **cls** command. You should then select screen editing with the **sedit** command - you should type in:

```
sedit ENTER
```

ARCHIVE puts you in the main level of the **sedit** command. At this level you can type text anywhere in the display area; you can move the cursor around the display area by means of the four cursor keys.

You can divide the display area into various sections, by drawing horizontal and vertical lines (using, for example, the minus sign and the exclamation mark respectively). You can then add the appropriate labels, such as Author, Title and so on. If you do not like anything that you have drawn on the screen, you can erase it by typing over it with spaces. If more drastic surgery is required you can press **F5** to leave **sedit**, clear the screen with the **cls** command and start again.

While in the main level of **sedit** the control area shows that you have two further options, selected by pressing either **F3** or **F4**. Pressing **F3** allows you to select a colour for both the ink and the paper. The change in colour takes effect from the current position of the cursor to the end of that line.

When you press **F3** you are first asked to choose an ink colour from the palette of black, red, green and white. The names of the colours appear in the central region of the control area. You select a colour by repeatedly pressing any key (except **ESC** or **ENTER**) until the colour you want is highlighted. You should then press **ENTER** to record your choice. ARCHIVE initially suggests white ink.

Once you have selected an ink colour you will be offered a similar selection to be used for the paper. You should select the colour in the same manner as for ink. In this case ARCHIVE makes an initial suggestion of black paper. You can, if you wish, choose the same colour for both paper and ink but it does make the following text rather difficult to read!

After you have made a choice of paper and ink colour, you are returned to the main level of **sedit**.

If you want the colour changed to affect only part of a line of the screen, you should move the cursor to the start of the region and select the paper and ink colours that you want. You should then move the cursor to the end of the region and make a second selection of paper and ink colours, returning them to their original values.

The final option, selected by pressing **F4**, is to allow you reserve space in the screen for the display of the value of a variable. When you press **F4** you will be asked to type in the name of a variable. Normally this will be the name of a field in one of your files, but could be the name of any variable that you use. Press **ENTER** to mark the end of

the name. This will indicate to ARCHIVE that you want to display the value of that variable, starting at the position of the cursor in the display area

You must then reserve the space for the value. Press any key (except **ESC** or **ENTER**) once for each character space that you want to reserve. The reserved space will show on the screen as a row of dots.

When you have reserved sufficient space you should press **ENTER** again and ARCHIVE will go back to the main level of **sedit**.

If you try to reserve space in a region which overlaps any area already reserved for the display of some other variable, you are given the option of cancelling the new area or of allowing it to replace the old one

When you have completed the display screen design to your satisfaction you should press **F5** again to leave the **sedit** command.

As an example, suppose you want to label an area of the screen with the word "Name", and reserve a 15 character space following the label to display the value of the variable `fname$`. You should move the cursor to the place where you want the label to start and type it into the screen exactly as you want it to appear. Next you should press **F4** and type in the name (eg `fname$`) of the variable you want to display, ending it by pressing **ENTER**. Finally you should press any key (eg the space bar) 15 times and press **ENTER** again. That region of the screen will appear as:

Name:.....

7.3 ACTIVATING A SCREEN FORMAT

Once you have designed a screen format and have left **sedit** the screen format will be left in the *active* state. This means that the values of all the variables in the screen will be displayed automatically every time that you ARCHIVE returns to the keyboard interpreter after carrying out a command (or a program). You can also force the display of the values of the variables in an active screen from within a procedure with the **sprint** command. If there is no active screen you will find that **sprint** has no effect.

If a screen format is in the computer's memory but is not active, you can activate it with the **screen** command. This also displays the text of the screen format, but does not show the current values of the variables. A screen which you have previously designed and saved on a Microdrive cartridge is also left in an active state when you load it into the computer's memory with the **sload** command, described in the next section.

Any active screen is deactivated each time you use the **cls** command

7.4 SAVING AND LOADING SCREENS

You can save your screen design on a Microdrive cartridge by using the command:

ssave "filename"

where "filename" is the name of your choice. The display screen will be saved exactly as it appears.

You can reload the display screen at any future time by typing in the command

sload "filename"

When you load a screen format it is automatically displayed on the screen and is made active.

7.5 THE DISPLAY COMMAND

Once you have an active screen format for your display screen you can use all the display words (**first**, **last** and so on) described in Chapter 5. The current values of any variables in the screen format are displayed on return to the keyboard interpreter, exactly as when you use the **display** command.

Remember that the **display** command itself always uses its own format and will always replace any screen format of your own that is in the computer's memory with its own list of the fields of the current record of the current file. You must therefore **ssave** your screen format before you next use **display**. If you do not your screen format will be replaced by that used by **display** and you will not be able to get it back again without redesigning it with **sedit**.

The commands and functions of ARCHIVE together form a programming language which you can use to write programs to manipulate your files. You will find that ARCHIVE programs are simple to write, although the approach is different from writing programs in SuperBASIC. If you have written programs in BASIC before you will see one immediate difference - ARCHIVE programs do not need line numbers. Many of the commands are, however, very like those used in SuperBASIC so you do not have too many new commands to learn about.

8.1 INTRODUCTION

An ARCHIVE program is made up of one or more separate sections. Each section is known as a *procedure*. A procedure is simply a named section of program. You can then refer to the procedure by its name, as with the procedures which you can write and use in SuperBASIC. In ARCHIVE you can run a procedure by typing its name at the keyboard (and pressing **ENTER**). It behaves in the same way as a command - when you write a procedure you are effectively adding a new command to ARCHIVE.

8.2 CREATING A PROCEDURE

Procedures may be as simple or as complex as you want to make them. It is, however, good practice to use lots of short procedures rather than one long one. You will find that you will make fewer programming mistakes. It will also be much easier to find any mistakes that do slip through.

You must use the *program editor* whenever you want to write or change a procedure. This editor provides you with many powerful tools for adding, deleting or changing the text of procedures. It is described in detail in Chapter 9, but in this chapter we shall look briefly at some of the main features so that we can write a few short procedures. We shall assume that initially there are no procedures in the computer's memory.

Type:

```
edit ENTER
```

to enter the program editor. You will see that the control area changes to show that you should type in the name for a new procedure. You have been placed directly in the option to create a new procedure. Entering the editor will always lead you to this option if you have not yet defined or loaded any procedures.

The first thing to do, therefore, is to define the new procedure. Let us start with a very simple task; to make life easier by renaming the **display** command. (This, as you will remember from Chapter 4, shows the contents of the current record in the display area.) The idea is to give it a single letter name and so reduce the amount of typing necessary when using it. We shall give it the name 'd'.

8.3 WRITING A PROCEDURE

You should just type in the letter 'd', followed by **ENTER**. The sequence of key presses so far, therefore, is:

```
edit ENTER
d ENTER
```

Once you have named the new procedure you will be shown the full range of editing options. Their actions are described in the next chapter. The left hand side of the display area will contain the name of the procedure. The right hand side of the display area will show a listing of the procedure. After the steps described above it should show:

```
d          proc d
          endproc
```

You did not need to type in the 'proc' or 'endproc' which mark the beginning and the end of a procedure. ARCHIVE always inserts them automatically when you create a procedure. This is a complete and usable procedure but, since it contains no commands, it will do precisely nothing!

Once you have created a new procedure as described in the previous section you have to add the *body* - that is the sequence of actions that it is to perform. In terms of the current example this means that you must now insert the name of the **display** command into the procedure.

PROVISIONAL

After you have given a name to the new procedure the contents of the control area changes again to show the full range of options available in the **edit** command. As you can see, one of these options is to insert lines of text into a procedure. All you have to do is to press **F4** and then type the text of a line, ending it by pressing **ENTER**. Press **F4** and then type:

```
display ENTER
```

ARCHIVE inserts the new line into the procedure, below the highlighted line. If you have followed this example so far the display will contain:

```
d          proc d
          display
          endproc
```

You could add more lines of text - each line (you mark the end of each line by pressing **ENTER**) would be inserted below the highlighted line.

In this case, however, the procedure is complete so you can leave the **edit** command - by pressing **ESC**.

All you have to do to use the procedure is to type its name, followed by **ENTER**. This new procedure will act in exactly the same way as the **display** command (remember that you must have opened a file before it can be used - trying to use the new procedure without having first opened a file will produce an error message just as you would get with the **display** command).

Why not try to use this same method to give single-letter names to all the other file display commands; first, last, next, and so on?

The next time you use the **edit** command ARCHIVE will allow you to select from the full set of editing options - remember that you are only directed to the option to create a new procedure when there are no procedures in the computer's memory.

You may be puzzled to see that the option to create a new procedure is not one of those shown in the control area. The reason is that it is one of a number of sub-commands within **edit**. You can select one of these sub-commands in the normal way you use to select any command - press **F3** and then the first letter of the name. To create a new procedure you will have, therefore, to press **F3** and then the N key (for New procedure).

From this point on the process follows the same method as described earlier.

8.4 LISTING AND PRINTING

Whenever you call the **edit** command you will see that you are shown, at the left of the display area, a list of the names of all the defined procedures present in the computer's memory.

You can list any one of these procedures from within the edit command by pressing the **TABULATE** key (to move down the list) or the **SHIFT** and **TABULATE** keys together (to move up the list) until the particular procedure name is highlighted. The procedure is automatically listed at the right hand side of the screen. If the procedure is too long to fit in the display area you will be shown the first part of it. You can then scroll up and down through the procedure with the aid of the up and down cursor keys. When you have finished looking at the procedure listing you can leave the **edit** command by pressing **ESC**.

If you want a printed listing of your procedures you can use the **llist** command. All you have to do is to type in

```
llist ENTER
```

and all the procedures currently in the computer's memory will be listed on the printer.

8.5 SAVING AND LOADING PROCEDURES

If you want to keep the procedures that you have defined for future use you can do so by using the **save** command. This stores all defined procedures in a single named file on the Microdrive cartridge. If you want to save the new display procedures that you have just defined with a file name "view", you should type in

```
save "view"
```

PROVISIONAL

At any later time you can bring these procedures back into the computer's memory by typing:

```
load "view"
```

Renaming commonly-used commands with single-names is one of making life easier for yourself. An alternative would be to write a longer procedure to display the contents of the records in any of your data files. Try using the **edit** command to define the following procedure. Don't worry if you make a few mistakes while typing it in - you will learn how to correct them in the next chapter.

8.6 EXAMINING FILE RECORDS

```
proc vufile
  cls
  input "which file?:";file$
  look file$
  display
  let key$ = "z"
  while key$ "q"
    sprint
    let key$ = lower(getkey())
    if key$ = "f":first:endif
    if key$ = "l":last:endif
    if key$ = "n":next:endif
    if key$ = "b":back:endif
  endwhile
close
endproc
```

Remember that you leave **edit** by pressing **ESC**. You can use the procedure by typing:

```
vufile ENTER
```

It will first clear the display area and then prompt you to type in a file name. When you have entered the name of one of your data files the procedure will open that file in read-only mode and display its first record. It will then wait for you to press a key and will respond to the keys f, l, n, b or q. The first four of these will cause the appropriate display action (first, last, next or back) and pressing the q (quit) key will close the file and end the procedure.

If you find you have made any typing errors, so that the procedure does not work properly, you can correct them with the aid of the line and program editors described in the next chapter.

Since this is the first program of any great length that we have written a few comments might prove helpful. Firstly note how the example is indented to clarify the structure of the procedure. There is no need for you to type it like this, with all the indentations. They are added automatically as you write, list or print the procedure.

The main part of the procedure (waiting for a key to be pressed and performing the appropriate action) is enclosed between **while** and **endwhile**. These commands cause the section of procedure that they enclose to be performed repeatedly, while the condition following **while** is true (not-zero). This repetitive loop will only be left when the condition is false (in this case, when you press the q key). For correct operation every **while** command must have a matching **endwhile**.

The **if** command, used several times within this loop, also requires that each **if** has a matching **endif** to mark the end of the sequence of instructions to be executed if the condition is true. **If** and **endif** are, like **while** and **endwhile**, separate commands and can be used on different lines. We could, for example, have written the first of the if statements in this procedure as;

```
if key$ = "f"
  first
endif
```

You may include several lines of statements between **if** and **endif**; they will all be executed, provided the condition following if is true. In the **vufile** procedure these statements are sufficiently short that each can be written on a single line, using the colon to separate the individual statements.

PROVISIONAL

As you can see, an **sprint** command is used within the main loop of this procedure. The reason for its inclusion is that, although the display commands (**first**, **last** etc.) always move to the correct record, the data in the display area is not normally changed until control returns to the keyboard at the end of a procedure. The **sprint** command forces the new data to be written into the display area immediately, regardless of whether a procedure is being executed or not. If we had not included the **sprint** command no information would have been shown in the display area until you pressed the q key to leave a procedure. In that case all you would see would be the result of the last of any sequence of key presses that you had made. This only applies to the commands which affect the display of the whole of a record: anything that you put into the display area by using the **print** command, for example, will appear immediately.

There is more information about procedures, concerning the use of parameters and local variables, in Sections 10.3.5 and 11.3.

This chapter describes the line and program editors and how to use them. We shall include a few simple examples, but the best way to learn is by using them yourself.

9.1 INTRODUCTION

When you have read this chapter you could try writing a few simple programs of your own, or you could try modifying the procedures you typed in while working on the last chapter. If you want to use longer examples you could use the editor to type in all or part of the the programs in the following chapters.

A full line editor is available at all times that you are typing characters at the keyboard. With its aid you can modify any or all of the editable text in the input line.

9.2 THE LINE EDITOR

The editable text excludes, for example, the **proc** statement at the beginning of a procedure. In general, any text that appears as the result of pressing a single key can not be edited. You can edit any text that you have typed in full, before you press **ENTER** to pass the text to ARCHIVE.

At all times each character that you type will be inserted to the left of the cursor position, and the cursor will move one space to the right. Regardless of the position of the cursor, all the text in the input line is accepted as input when you press **ENTER**.

The line editor uses the four cursor keys, together with the **CTRL** and **SHIFT** keys.

Left and Right Cursor Keys The left and right cursor keys, used on their own, move the cursor by one character to the left or right in the input line.

If you press **SHIFT** and, while holding it down, press the left or right cursor key the input line cursor moves left or right by units of a word, that is to the next space or comma.

If you press **CTRL** and, while holding it down, press the left cursor key you will delete the character to the left of the cursor. Pressing **CTRL** together with the right cursor key deletes the character under the cursor. The following text closes up to fill the gap.

Up and Down Cursor Keys If you press the up cursor key the cursor moves to the beginning of the editable text in the input line; the down cursor key moves the cursor to the end of the text.

Holding down the **CTRL** key and pressing the up cursor key will delete all editable text to the left of the cursor. Pressing **CTRL** and the down cursor key deletes all text to the right, including the character under the cursor.

You can use the line editor on the text of a command you have just typed in, before you press **ENTER**. This allows you to correct any typing mistakes you make, without having to type in the whole line again.

Suppose you do not notice a mistake before you press **ENTER**. ARCHIVE will detect it when it tries to carry out your instructions and give you an error message. Even at this stage all is not lost. You can press **F5** which will put the last line of text you typed in back into the input line. You can then use the line editor to correct the mistake and press **ENTER** to try again.

You can also use the line editor from within the program editor to change a line of one of your procedures.

You enter the *main level* of the edit command from the main command level by typing:

edit ENTER

You can leave edit at any time by pressing **ESC**.

9.3 THE PROGRAM EDITOR

PROVISIONAL

When you enter the **edit** command the display area changes to show, on the left, a list of the names of any procedures that are in memory. They are always listed in alphabetic order. The first procedure in the list is shown in full on the right hand side of the display area. You are in the *main level* of the **edit** command.

You will notice that the name of this first procedure is highlighted, as is the first line of its listing. At all stages during the use of the editor highlighting marks the *current procedure* and the *current line* within it. This is the line that will be affected by any changes you make.

If there are no procedures in the computer's memory at the time you select the **edit** command, the display area will be blank and you are automatically given the opportunity to create a procedure (as described in the previous chapter). Otherwise the control area changes to show the list of the main options available to you. These are to:

- select a procedure
- select a line within a procedure
- select an editing command
- insert text into a procedure
- edit a line of a procedure

We shall examine each of these in more detail

9.3.1 Select a Procedure

You can select a different current procedure by pressing **TABULATE** to move down the list of procedures, or by pressing **SHIFT** and **TABULATE** together to move up the list. Each time you change the current procedure the listing at the right will change so that it always shows the current procedure.

9.3.2 Select a Line

You can use the up and down cursor keys to select a different current line within the current procedure. The current (selected) line is marked by highlighting. Insertions, for example, will be added immediately after the current line.

9.3.3 Editing Commands

There are four separate editing commands within the edit command itself. They are:

- Delete procedure
- New procedure
- Cut
- Paste

When you are at the main level of **edit** you can select one of them by pressing **F3** and then typing the first letter of its name. At the end of the action of each of these commands ARCHIVE will go back to the main level of **edit**.

Delete Procedure

This command deletes the current procedure from your program. You must first select the procedure you want to delete by using the **SHIFT** and **TABULATE** keys, as described earlier, to make it the current procedure. You then select the command by pressing **F3** and then the D key.

You must then press **ENTER** to confirm that you really do want to delete the procedure. If you change your mind at this stage you can, instead of pressing **ENTER**, press any other key to leave the command and go back to the main level of **edit** without deleting the procedure.

Be careful when you use this command since there is no way to restore a deleted procedure, except by typing it in again.

New Procedure

You will need to use this option whenever you want to start writing a new procedure. As was mentioned earlier you are automatically given this option if you select the **edit** command when there are no procedures in the computer's memory. Otherwise you select it pressing **F3** and then the N key.

As indicated by the prompt, all you have to do is to type in the name of the procedure you want to create. If you type in the name of an existing procedure you will not be allowed to create a second procedure with the same name. In this case you will be offered the option of editing the existing procedure of that name.

When you press **ENTER** at the end of the name the new procedure becomes the current one, listed at the right of the screen. You are presented with an empty procedure - that is, one containing only the **proc** and **endproc** statements - ready for you to add its body.

For example, if you select the **new procedure** command, by pressing (from within **edit**) **F3** and then **N**, and then type in the name **test**, you will find that the display area contains:

```
test      proc test
          endproc
```

Cut This command removes one or more lines of text from the current procedure. The text that is removed can be inserted in another position, or even in another procedure, by means of the **paste** command.

Before you select the command you should use the up and down cursor keys to make the current line be either the first or the last line of the section you want to remove. You can then select the command by pressing **F3** and then the **C** key.

If you then press **ENTER** the current line will be removed from the procedure. Alternatively you can use the up or the down cursor key to move the cursor to the other end of a section of text that you want to remove. The region of text that will be removed is marked by highlighting. When you have marked the text you want to remove you should press **ENTER**.

ARCHIVE will immediately delete the marked text. The text that is removed replaces any text removed by a previous use of **cut**. If you want to insert the text elsewhere you must therefore use the **paste** command before you use **cut** again.

Paste This command inserts the text removed by the last use of the **cut** command into the current procedure, below the current line. The text can be inserted in another position, or even in another procedure.

Before you select the command you should, if necessary, use the **SHIFT** and **TABULATE** keys to select the procedure in which you want to insert the text. You should also use the up and down cursor keys to make the current line be the line immediately above the position where you want to insert the text. You can then select the command by pressing **F3** and then the **P** key.

ARCHIVE immediately inserts the text, underneath the current line.

You can select the option to insert lines of text below the current line by pressing **F4**. Anything you type, up to the next time you press **ENTER**, is inserted as a new line of text. This new line then becomes the current line.

9.3.4 Inserting Text

ARCHIVE stays in the insert option so that you can type in several lines; you mark the end of each line by pressing **ENTER**. When you have finished inserting new text you should leave the option by pressing **ENTER** without first typing any text into the input line.

As an example we can add a couple of statements to the **test** procedure which we created earlier in this chapter. Remember that, from the main level of **edit**, we left the procedure with just its **proc** and **endproc** statements. Make sure that you are at the main level of the editor and that the screen shows:

```
test      proc test
          endproc
```

The highlighting marks the line including **proc**, so any inserted text will go under this line.

Now press **F4** and then type:

```
print "this is a test" ENTER
print "there are two statements" ENTER
ENTER
```

When you have finished the screen will look like:

```
test          proc test
              print "this is a test" ENTER
              print "it has two statements" ENTER
endproc
```

The highlighting marks the line containing the second print statement.

If you make a mistake you can correct it, provided you notice it before you have pressed **ENTER**, by using the line editor, described earlier. Remember that you can use this editor at any time that you have typed some text into the input line, before you press **ENTER**.

Once you have pressed **ENTER** the line of text is inserted into the procedure and you will have to use the line editing option, described in the next section, to make any corrections.

9.3.5 Edit a Line From the main level of **edit**, press **F5** to edit the current line. The contents of this line are copied into the input line and you can then edit the text with the line editor, described earlier. When you press **ENTER** ARCHIVE will replace the old line in the procedure with the contents of the input line.

You are not allowed to edit the **endproc** statement at the end of the procedure. You are also not allowed to edit the word **proc** in the first line of the procedure, but you may edit the rest of the contents of this line. You can, therefore, rename a procedure by using the line editor to delete the old name and replace it with a new one. The list of procedures at the left of the screen is rearranged automatically to keep the procedures in alphabetical order.

10.1 INTRODUCTION

This chapter is about writing programs in the ARCHIVE database language. In addition to explaining the main features of the language, it will describe the development of an actual working example. The example will be developed as we go along, and each new technique will be described as it is needed.

Suppose you are involved in running a club or society which charges a subscription and produces a newsletter. You will need to send a copy of each issue to every paid-up member. You will also need to send a reminder to each member when his or her subscription falls due.

This example allows you to construct a mailing list and will then print a set of address labels on request. The address label includes a reminder when a subscription is due. The example assumes that you send out six issues of the newsletter per year and that a person's subscription falls due when he or she has received six issues. It could easily be adapted to any situation where you regularly send out some form of circular letter to a number of people on a mailing list.

In this example we shall make as much use as possible of the existing facilities in the database language. We can, for example, use the **insert** and **alter** commands for all additions and changes to the file records. We shall, however, need to write special routines to print out the address labels.

10.2 BASIC DESIGN

We shall have to cater for the following set of requirements:

- 1) Add a new record to the file.
- 2) Delete a record.
- 3) Modify a record.
- 4) Record subscription payments.
- 5) Produce the address labels.
- 6) Leave the program.

We shall write a procedure to handle each of these tasks and link them together by another procedure which will allow you to select any of the options.

In this application it is quite clear what fields each record must contain. There will have to be the name and address plus one field to record the number of issues the person has received. We can create the necessary file immediately, as shown below.

```
create "mail"  
title$  
fname$  
surname$  
street$  
town$  
county$  
postcode$  
issues  
endcreate
```

We have used three string fields for the person's name; to hold the title (Dr, Mr, Mrs etc), the first name and the surname respectively. We could probably have managed with just a single field but, as you will no doubt discover, you can never have too many fields in a record - the normal problem is that you have too few!

There are four string fields for the address, nominally reserved for the street address, the town, county and postcode. You do not always have to use them in this way, but can treat them as four general fields to hold the address. Four fields should normally be quite sufficient.

There is only one numeric field, to hold the information about how many issues have been sent.

Now that we have the file, we can use it to test the various procedures as we write them. It is a good idea to test each procedure as far as possible as you go along. You

can then spot each mistake as it occurs and correct it immediately. If you leave all the testing to the end it will be much more complicated as several things may be going wrong at the same time. Keep things as simple as possible while you are still testing your procedures - try to make sure that each procedure works correctly before you move on to the next one. That way you will find that your final program will usually work as soon as you have written the last procedure.

10.3 THE MAIN TASKS

10.3.1 Insertion We can write the procedure to add a record very simply:

```
proc add
  sprint
  insert
endproc
```

This procedure uses commands described before and should be easy to follow. Remember that you must use **sprint** to force the display of the contents of the record since they are not normally displayed until control returns to the keyboard interpreter.

You can use this procedure immediately to add a few records to the file so that you can test the other procedures on a real file.

10.3.2 Deletions At some time you will want to remove the records of people who have not renewed their subscriptions. We shall write a procedure, **kill**, which allows you to scan through the file, examining the records of all people who have not renewed, and to decide whether each one should be deleted.

We shall use the field variable **issues** to hold the number of issues that a person is entitled to receive. All records for which the value of **issues** is zero are therefore candidates for deletion.

```
proc kill
  cls
  select issues = 0
  all
  sprint
  print at 10,0; "DELETE (y/n)? ";
  let ok$ = lower(getkey())
  print ok$
  if ok$ = "y"
    delete
    print "DELETED"
  endif
endall
endproc
```

Since a deleted record can not be recovered, the full contents of the record are displayed and you are asked to confirm that you really want to delete it. We use the `getkey()` function which waits for a key to be pressed and then returns the ASCII code of that key. Note that `lower()` converts the code to the lower case character so that you can type the letter in either upper or lower case.

10.3.3 Payments You will normally want to record subscription payments from a list of names and addresses of those people who have sent in their subscriptions. You will therefore need to locate the record of a particular person. The best approach is to write a separate procedure, **getrec**, to locate a particular record and then incorporate it in the **pay** procedure.

This procedure asks for a text string and then locates the first record in the file which contains that text. If you reply by just pressing **ENTER**, `n$` is set to the empty string (""), and no search is made. You should use this method to indicate that you have finished recording payments.

PROVISIONAL

```
proc getrec
  rem ***** locate a particular record *****
  let ok$ = "n"
  input "who? "; n$
  if n$ ""
    find n$
    while ok$ "y" and found()
      print title$ ; " "; fname$ (1); " "; surname$
      print street$
      print "OK (y/n)? ";
      let ok$ = lower(getkey())
      cls
      if ok$ "y"
        continue
      endif
    endwhile
  if not found()
    print n$ ; " not found"
  endif
endproc
```

The search uses the **find** command, so that the text is found in any string field. You can therefore identify a record by name or by address. Of course, the first record which matches may not be the one you want, so we have to be able to continue the search. This is the purpose of the **while endwhile** loop. This prints out the name and first line of the address, to identify the record, and asks you if that is the right record. If you do not respond by pressing the Y key, it continues the search. The loop ends either when you answer by pressing the Y key or when the text is not found in any of the remaining records. (Note that the function **found()** returns a non-zero value if the search is successful.)

Note that, since **ok\$** could initially be "y" (from a previous successful search) we must give it some other value at the beginning of the procedure, before entering the loop, to make sure that the loop will be used at least once.

We can now write the **pay** procedure:

```
proc pay
  cls
  let n$ = "x"
  while n$ ""
    getrec
    if ok$ = "y"
      let issues = issues + 6
      update
    endif
  endwhile
endproc
```

The loop in this procedure continues until **n\$** is an empty string. This allows you to record several payments without having to select the **pay** option for each one. When you have finished, just press **ENTER** in response to the "who?" prompt. If the value of **ok\$** is "y" after the call to **getrec** then the payment is recorded by marking it as valid for a further six issues.

Again we have to set the initial value of **n\$** to some appropriate value (anything except "") to make sure that the procedure is not affected by a previous operation.

10.3.4 Changes The procedure to allow you to change the contents of a record is now very easy. Again you must be able to select a particular record to change, so the general structure can be identical to `pay`.

```
proc change
  let n$ = "x"
  cls
  while n$ ""
    getrec
    if ok$ = "y"
      sprint
      alter
      cls
    endif
  endwhile
endproc
```

10.3.5 Parameters We shall now take a short break from the development of the program to describe the use of parameters with procedures. You can use a *parameter* to pass a value to a procedure, rather than using the value of a variable. Rather than giving a long description of the theory of parameters, we shall show you a few examples of how they can be used.

Try the following simple example. You add the parameter to the line containing the procedure name by using the line editor (press **F5**) option of the `edit` command.

```
proc test; a
  print 5*a
endproc
```

This defines a procedure called "test" which requires one parameter, "a". Notice that the parameter is separated from the name of the procedure by a semicolon.

Whenever you use the procedure you must always supply a value for the parameter. For example, you could type:

```
test; 3 ENTER
```

which will print the value 15 - the number (3) has been passed to the procedure as the value of the variable a.

You may specify any number of parameters for a procedure, provided you separate them by commas. For example:

```
proc trial; a,b,c
  print a * b * c
endproc
```

which you can call by:

```
trial; 3,4,5 ENTER
```

The values you supply do not have to be literal values, but could be variables, as shown below:

```
let x = 2 ENTER
let y = 5 ENTER
let z = 7 ENTER
trial; x,y,z ENTER
```

Note that the names of the variables do not have to be the same as the names used within the procedure. We can distinguish between the *formal parameters* (eg a,b,c) in the definition of the procedure, and the *actual parameters* which are the actual values that are passed to the procedure.

You can also pass the results of expressions:

```
trial; x*2,z/y,(z-y)*x ENTER
```

You are not restricted to using numeric variables but can also pass strings (or string expressions) as parameters, provided you specify string variables in the definition of the procedure. For example:

PROVISIONAL

```
proc try; a$
  print a$
endproc
let t$ = "message" ENTER
try t$ ENTER
```

The only requirement is that the number and types of parameters supplied must match the list of formal parameters in the definition of the procedure.

The reason for the brief interlude about procedures is that they give a neat way of writing the procedure to print an address label. For the purposes of testing we shall first write the procedure to show the addresses on the display and later convert it to send the output to the printer. We shall assume that the labels are eight lines of print-out in length. If this is not right for your printer and label combination you will have to change the number of lines of space in the procedure so that it matches your needs.

10.3.6 Address Labels

First we shall write a procedure that displays a single line, the contents of which are passed via a parameter.

```
proc doline; x$
  print x$
endproc
```

We can now use this procedure to display eight lines of text for the address label.

```
proc dolabel
  if issues
    if issues = 1
      doline; "REMINDER - Subscription Now Due"
    else
      doline; ""
    endif
    doline; ""
    doline; title$ + " " + fname$ (1) + ". " + surname$
    doline; street$
    doline; town$
    doline; county$
    doline; postcode$
    doline; ""
    let issues = issues - 1
    update
  endif
endproc
```

The procedure includes a reminder in the address label if the person is about to receive his or her last issue. Each time a label is printed, that person's issue count is reduced by one. If this number has reached zero then the label is not printed.

You can begin to see how useful parameters can be - without them this procedure would be much longer. Look how easy it is to combine the title, initial and surname for the first line of the address.

Perhaps you are wondering why we went to the trouble of defining **doline** when we could have just used **print** statements throughout **dolabel**. The reason is that, as we mentioned earlier, the routine in its present form shows the addresses on the display screen. We can convert it to send its output to the printer merely by changing one line in **doline**, instead of having to change every print statement in **dolabel**. All we need to do is change **doline** to read:

```
proc doline; x$
  lprint x$
endproc
```

Finally we can write the procedure to print all the address labels:

```
proc dispatch
  cls
  all
  dolabel
endall
endproc
```

10.3.7 Leaving the Program

The final option is to leave the program when you have finished. This procedure can be very simple - all it has to do is to make sure that the file is closed properly before returning control to the keyboard interpreter. We have also added a short sign-off message to make it clear that the program has ended.

```
proc bye
  close
  print "bye"
  stop
endproc
```

10.3.8 Putting it Together

We can now write a procedure which will allow you to select any one of the six options with a single key press. It is sufficiently simple that no explanation is necessary.

```
proc choose
  cls
  print
  print " Add Dispatch Pay Change Kill Quit"
  print "? ";
  let c$ =lower(getkey())
  print c$
  if c$ ="a": add : endif
  if c$ ="d": dispatch : endif
  if c$ ="p": pay : endif
  if c$ ="c": change : endif
  if c$ ="k": kill : endif
  if c$ ="q": bye : endif
endproc
```

10.3.9 Errors

It is quite likely that sooner or later you will make an error while using this or some other program. You may, for example, accidentally press the **ESC** key or you may type in some text when a number is expected. This type of mistake is detected by ARCHIVE and normally results in the display of an error message and a return from your program to the keyboard interpreter. This could be annoying, to say the least! Fortunately, ARCHIVE has a method by which you can handle all such errors from within the program.

You can use the **error** command to mark a procedure to be treated specially if any error is detected. Any error occurring in the marked procedure, or any procedure that it calls, results in an immediate, premature, return from the marked procedure.

The normal method of handling errors is switched off for the marked procedure and it is left to you to decide how to deal with it. You can find out the number of the last error that occurred by using the `errnum()` function. You can use it to read the error number more than once as the value is only cleared to zero by the next use of the **error** command. If no errors have occurred since the start of the program, or since the last time **error** was executed, then `errnum()` will return a value of zero.

This method, although not easy to understand at first, gives you a very powerful and flexible control of how to deal with errors. The following example shows a typical way of using **error**. It gives you an error-resistant method of inputting a number.

```
proc dotest
  input x
endproc

proc test
  let n = 1
  while n
    error dotest
    let n =errnum()
    if n
      print "You made error number " ;n ;", try again"
    endif
  endwhile
endproc
```

The first procedure simply waits for your input to the variable `x`. The second procedure handles any error during the execution of the input procedure. If any error occurs within **dotest** it will be terminated prematurely and the error number will be set. This number

PROVISIONAL

is then read by `errnum()` and, if it is non-zero, the error message is printed (this error message could, of course, be anything you like). Since these statements are enclosed in a **while endwhile** loop, any error will cause them to be executed again. The error number is cleared by **error**, ready for the next try. You can not leave **test** until you have typed in a valid number.

This example reports the number of the error that was detected. On most occasions you will not be concerned about which error occurred. The main use of `errnum()` is to differentiate between there being no error - `errnum()` returns zero - and there being a detected error of any type - `errnum()` returns a non-zero value.

All that remains to be done to complete our program is to write a start-up procedure which opens the file and calls **choose**. We must include **choose** in a loop so that you are offered the options again, each time you complete your previous selection.

10.3.10 The Final Program

You will see that the **while endwhile** loop in the following procedure will never end. Such a loop will only come to an end when the expression following **while** has a zero value. In the above procedure the expression (ie 1) is never zero, so the loop will continue indefinitely. The only way of leaving this loop is to choose the Quit option. The **stop** command in **bye** immediately returns control to the keyboard interpreter.

```
proc start
  cls
  open "newmail.dbf"
  while 1
    error choose
    let n = errnum()
    if n
      print "Mistake - Press any key to continue"
      let m = getkey()
    endif
  endwhile
endproc
```

Within this loop is a sequence of statements which handles any errors, using a similar method to that described in the previous section. If you make a mistake the program will not continue until you press a key. This allows you to look at what you have just done so that you can find out how you made the error.

The main procedure is named "start". This is so that you can use the **run** command when using the program. Suppose that, when we have written all the procedures of the program, we save them under the name "maillist". When you want to run the program you will need to load the procedures into the computer's memory and then execute the main procedure, which will call all the others. One way is to use the **load** command and then type in the name of the main procedure, for example:

```
load"maillist" ENTER
start ENTER
```

The **run** command will load a named program and then execute the procedure named "start" (if it exists). You can run the program exactly as in the previous example just by typing:

10.4 THE RUN COMMAND

```
run"maillist" ENTER
```

1901 1465

(

(

(

(

This chapter describes more of the techniques of writing programs in ARCHIVE. Again it uses a running example which is both longer and more complex than the example in the previous chapter. It shows some alternatives to the methods of Chapter 10 and extends the description of procedures, parameters and error-handling. In addition it explains how to use local variables in procedures.

The example that we have chosen is a cheque book reconciliation which you can use to keep an eye on your expenditure, or to check on your bank statements. None of the input and output procedures use the special commands, so that they can be exactly tailored to the application.

The idea is that you can type in the details of all your bank account debits and credits, either day by day or at the end of the month. You can then ask for a report of the current state of your account. In order to make it easy to use, we shall use a comprehensive set of prompts for your input. In addition it will be flexible enough to allow you to modify a previous month's account, and to ask for a report for any month about which you have records.

The example is provided on the ARCHIVE Microdrive cartridge. You can use it by typing in:

```
run "chequebook" ENTER
```

Before we get involved in the details of the program it is worth thinking about the general structure as implied by the brief specification in the previous paragraph. We shall use a basic *menu-driven* approach, where at each stage you are presented with a list of the options you have (rather like ARCHIVE itself). At the highest level we can identify three main options:

- 1) add an entry,
- 2) obtain a monthly report,
- 3) leave the program.

Each of these main options may have subsidiary menus, and you should be able to choose any option by pressing a single key, throughout the whole program.

We can now start writing the shell of the program. It must open the chequebook data file and display the initial menu. The program must return to this menu whenever a selected option has been completed. This implies that the menu selection should be placed in an endless loop

```
proc start
  rem ***** run the whole thing *****
  mode 0,8
  cls
  print at 7,10; "CHEQUE BOOK RECONCILIATION"
  open "cheque"
  pause
  let x$ = lower(getkey())
  while 1
    error menu
    if errnum()
      print "Mistake"
      pause
    endif
  endwhile
endproc
```

This procedure starts by clearing the display area and printing the program title at line 7, column 10. We have included the **mode** command to remove the display of the prompts, since none of the standard commands will be used. The procedure then opens a file called "cheque" and uses **pause**, which waits for you to press any key before continuing. The **pause** procedure is described later, in Section 11.4.

PROVISIONAL

You will note that we have made no decisions about the contents of the file records - the only decision about the file is that it is to be called "cheque". As yet we are not sure about what fields will be necessary. The actual structure of the file will be decided much later when we have found out what is needed.

We have, as in the example of the previous chapter, again used an endless **while endwhile** loop to keep the program going. We have also used the **error** command to make sure that any error causes a return to this loop, rather than giving an error message and leaving the program. The method is similar to that used in Section 10.3.10, but is a little shorter.

11.3 LOCAL VARIABLES

Most variables that appear in procedures are *global*. This means that they are defined for the whole of the program. They may be used or changed in any procedure, and not just the procedure in which they are first assigned a value.

The variables used as formal parameters in a procedure are *local variables* in that they are not defined outside the procedure in which they appear.

The following example may help to make the distinction clear.

```
proc demo; a,b$
  print x,y$
  print a,b$
endproc

let u = 3 ENTER
let v$ = "text" ENTER
```

```
demo; u,v$ ENTER
```

```
print u v$ ENTER
```

In this example *u* and *v\$* are normal (global) variables. They were assigned values outside the procedure "demo" but their values are defined inside the procedure as well as outside it. There is no problem with printing their values from inside or outside the procedure.

The variables *a* and *b\$*, however, are local to the procedure. The example shows that they are recognised inside "demo", but if you now try:

```
print a,b$ ENTER
```

you will see that their values are not defined outside the procedure.

All formal parameters are local variables, but you can also declare other variables to be local, as in the following example:

```
proc dumbo
  print "inside dumbo"
  print p,q,r
endproc

proc dummy
  local q,r
  let p = 2
  let q = 3
  let r = 4
  print "inside dummy"
  print p,q,r
  dumbo
endproc

dummy ENTER
```

This example shows that the values of *p*, *q* and *r* are all defined in "dummy", but "dumbo" does not know the values of *q* and *r*, which are local to "dummy". The values of local variables are not defined anywhere except in the procedure in which they are declared - not even in procedures called from the declaring procedure. The variable *p* is global and is recognised everywhere.

PROVISIONAL

If the program does not contain a procedure with the name "start" then the **run** command has the same effect as **load**. The menu procedure will need to display the list of available options together with a prompt message, and then wait for you to press a key. Displaying a prompt and waiting for a key to be pressed is one of the most commonly-needed actions, so it is worth writing a general-purpose procedure. The procedure must be able to display a wide range of messages. A simple way of allowing the procedure to print any message is to pass the message to the procedure in the form of a parameter.

We can now continue with our cheque book program, by defining a general-purpose prompt routine.

```
proc prompt; m$
  print m$ ; " ";
  let x$ = lower(getkey())
  print x$
endproc
```

The message to be displayed is passed to the procedure as a parameter in the local variable **m\$**. The function **getkey()** waits for a key to be pressed and returns the ASCII code for the key. In this procedure the ASCII code is converted to lower case by the function **lower()**, so that the result is independent of upper or lower case. Finally the resulting value is assigned to the variable **x\$**. This is a global variable, so that the key that was actually pressed is available to any other procedure in the program.

A useful routine, which has already been used and will be used in many places in the program is **pause**. It uses "prompt" to print a message and then simply waits until a key is pressed. Since you are not usually interested in knowing which key was actually pressed, it uses a local variable, **y\$**, to preserve the original contents of **x\$**. We can then use it to write the **menu** routine, to display the main menu of our program.

```
proc pause
  rem ***** wait for any key *****
  local y$
  let y$ = x$
  print
  prompt; "press any key to continue"
  let x$ = y$
endproc

proc menu
  rem ***** display the menu of options *****
  cls
  print tab 15; "OPTIONS"
  print tab 10; "a - add entry"
  print tab 10; "r - monthly report"
  print tab 10; "s - stop"
  let x$ = ""
  while not (x$ = "a" or x$ = "r" or x$ = "s")
    print
    prompt; "Choose (a/r/s):"
  endwhile
  if x$ = "a": entry
  else : if x$ = "r": report
        else : if x$ = "s": bye : endif
  endif
endif
endproc
```

After clearing the display area, **menu** lists the options and then uses **prompt** to wait for your selection. Note that **prompt** is used within a loop which continues until one of the valid keys is pressed. (Remember that **prompt** returns the key value in **x\$**.)

The last few lines, where the selected procedure is called, seem to be a bit more complex than you might expect. It would seem reasonable to use the following test.

```
if x$ = "a": entry: endif
if x$ = "r": report: endif
if x$ = "s": bye: endif
```

This contains a hidden danger. Suppose you had selected the first option and that this procedure changed the value of **x\$** to either "r" or "s". You would then find that, on

11.4 PROMPTS AND MENUS

PROVISIONAL

return from **entry**, one of the other two tests would succeed and another of the options would be called. This is one of the dangers of using a universal prompt routine, always returning the key that was pressed in the same global variable. You should always be aware of this type of interaction between different procedures in your programs. In the present program the various options have been chosen so that this problem can not occur.

Once a valid key has been pressed, the appropriate procedure is called:

“entry” for entering a credit or debit,
“report” for producing a monthly report, or
“bye” to leave the program.

The last of these procedures is easy to write:

```
proc bye
  rem ***** exit gracefully *****
  close
  print
  print "bye"
  mode 1,8
  stop
endproc
```

Don't forget that you have to close the file and use **mode** again, to restore the display of the prompts, before leaving the program.

So far we have not had to think too much about the actual application. The procedures so far could be used with any menu-driven program. Now, however, we must start considering the particular requirements of the cheque book program.

11.5 DISPLAYING RECORDS

11.5.1 Introduction

In this section we shall develop the routines necessary to display a record and a monthly report. Before we can decide how the display will work we have to make some decisions about the structure of a record.

11.5.2 The Record Structure

Although you could use the **display** command to show a file record, it is not particularly suitable for this application. We shall want to display the report for a whole month and this implies that each record should take up only one line. Furthermore, it would be useful if we could display the results in columns, with credits and debits separated into different columns.

The credit and debit records must therefore be treated separately, and we must first decide how to do this. One idea would be to use positive and negative values for credits and debits respectively.

We shall also need a third type of record to hold internal 'housekeeping' data. These records can be used to transfer the running balance from one month to the next. You will never see these records when you use the program.

If we use a separate field in each record to indicate its type, we can use it to distinguish between credit, debit and housekeeping records. It will also allow us to use other types of records if we find we need them.

We are now close to a specification of the form of a record in our program. It must have fields for the date (day, month, year), the amount and the record type. In addition we might want to include a cheque number for each cheque you draw and some space for a note of what the amount was for.

We choose to store the date in three separate fields of the record so that we have maximum flexibility in selecting and ordering the records of the file. Let us therefore make the following tentative definition of the fields of a record:

PROVISIONAL

Variable	Type	Purpose
day	Numeric)	
month	Numeric)	the date
year	Numeric)	
amount	Numeric	cash amount
chqno	Numeric	the cheque number (0 if no number)
detail\$	String	any details you want to record
type\$	String	the record type: c = credit d = debit b = balance to carry forward

Now that we have a good idea of the basic structure of the records we can write the routines to display them, and to add new ones.

11.5.3 The Display Procedures

We shall start by writing the procedures to display a single record, and then a whole series of records in date order. They make extensive use of the **tab** print item to put the various values in columns. The debit and credit figures are displayed in different columns by **showrec**. Note that this procedure regards all types of records, except type "d", are regarded as credits.

```
proc showrec
  rem ***** display a record *****
  print day ; "/" ; month ; "/" ; year ;
  if type$ = "d"
    print tab 22; amount ; tab 32; endbal ;
    if chqno 0
      print tab 44; chqno ;
    endif
  else
    print tab 12; amount ; tab 32; endbal ;
  endif
  print tab 55; detail$
endproc
```

The next procedure displays the body of the report, displaying all the records in date order. It also keeps a running balance and displays it at the end of the list.

```
proc doreport
  rem ***** show a set of records totals *****
  local n
  print "Date"; tab 12; "Credit";
  print tab 22; "Debit"; tab 32; "Balance";
  print tab 44; "Cheque no"; tab 55; "Details"
  print
  order day ; a ,type$ ; a
  let endbal = 0
  first
  let n = count()
  while n 0
    if type$ = "d"
      let endbal = endbal - amount
    else
      let endbal = endbal + amount
    endif
    showrec
  next
  let n = n - 1
  endwhile
  print
  print tab 20; endbal ; " Carried Forward"
endproc
```

Note that we do not use an **all endall** loop to display the records. This type of loop is very efficient way to scan through all the records of a file, but will not proceed in any particular order. Instead we use the less efficient loop that is highlighted in the procedure. It guarantees that the records will appear in their sorted order.

This procedure displays every record in the file. You must make sure that you have selected only those records that you want to see before calling it.

11.6 DATA ENTRY

11.6.1 INTRODUCTION

In our cheque book program we shall have to type in text and numbers when adding new details to the file. In addition we shall need to type in dates. These usually have a special format and could be treated either as numbers or as text.

First we shall look at some methods for entering text, numbers and dates. Obviously we shall make use of the **input** command, but we shall need to consider how to detect and deal with errors in the typed input.

11.6.2 Entering Text

Accepting text as typed input is quite simple. Any collection of characters is a valid text string (even if it does not make sense) and will not cause a system error. You will not normally need to take any special precautions when accepting text input. It will usually be sufficient to use a line such as the following, which asks you to type in your name:

```
input "Please type your name: ";name$
```

Note that a space is included as the last character of the prompt text; This small point makes a lot of difference to the appearance of your program when you use it.

You can input several items with one input statement. All you have to do is to include all the prompts and variable names, separated by semicolons.

```
input "Your first name! ";fname$;"Your surname? "sname$;
```

This last input statement also ends with a semicolon -- this stops the cursor moving to the following line after you have typed your input.

You may want to perform additional checks on the input text, for example to see if it is a particular number of characters long. You will find an example of this type of technique in the date entry routine given in Section 11.6.4.

11.6.3 Numbers

When you use the **input** command to enter text to a string variable the computer will accept anything that you type, without complaint. If, however, you try the same thing with input to a numeric variable you will get an error message if you type anything except a valid number. Assuming that you do not want to leave your program every time your finger slips while you are typing in a number, you must make sure that your program can cope with such errors.

One possibility is to make all input use string variables, even for inputting numbers. Your program can then convert the text string to a number, and ask you to try again if anything goes wrong. Suppose you want to input a value for the variable num. You could write a procedure such as the following one, which will only accept the input of a valid number with two places of decimals.

```
proc getval
  local ok,num$,test$
  let ok = 0
  while not ok
    input num$ :rem *** input string ***
    let num = val(num$) :rem *** convert to number ***
    let test$ = str(num,0,2) :rem *** & back to a string ***
    if test$ = num$ :rem *** then the number is valid ***
      let ok = 1
    else
      print "Try again"
    endif
  endwhile
endproc
```

PROVISIONAL

You can control the exact format of the number to be accepted by means of the `str()` function. In this case `str()` converts the number to decimal format, with two decimal places. The number is accepted only if the original string (`num$`) and the processed string (`test$`) match exactly.

You can not use this technique to input a number in any format, since it only accepts the format specified by the `str()` function. If you want to accept a number in any format you must use a different method. The most useful way is to make use of the **error** command, which was described in the last chapter. The following procedures, for example, will accept any valid number within a specified range. They even provide the display of any prompt message you want to appear.

```
proc getnum; m$ ,min ,max
  local wrong
  let wrong = 1
  while wrong
    print m$ ; "? ";
    error readnum
    let wrong = errnum()
    if not wrong
      if (num min ) or (num max )
        let wrong = 1
        print "Allowed range is "; min ; " to "; max
      endif
    endif
    if wrong
      print "Try again"
    endif
  endwhile
endproc

proc readnum
  input num
endproc
```

We shall use this form for numeric input in our example. Note that we now have two levels of error-handling. Any error during numeric input will use the error treatment in **getnum**, but any other error will result in a return to the main menu, using the overall error-handling in **start**.

In the cheque book program we shall certainly need to sort the records of the file into date order and to be able to select all the records for a particular month. These processes will be kept simple if we store the date as the day, the month and the year, in three separate variables.

11.6.4 Dates

As was mentioned in Section 11.6.1, dates are usually written in a special format. When you want to type in a date you must choose how you are going to deal with it. You could, for example, input the day, month and year as separate numeric values:

```
input "day? ";day
input "month? ";month
input "year? ";year
```

The following procedure accepts date in the format `dd/mm/yyyy`. It checks that all three parts are present, and that they are separated by `"/"` symbols. You are allowed to type in single-digit day or month figures with either one or two characters, eg you can use either `"3"` or `"03"`. You must, however, type the year with all four digits. The procedure does not otherwise check the date for validity so, for example, the date `31/2/1996` is accepted. You could add a check for a valid calendar date once you have used **getdate** to separate out the day, month and year.

PROVISIONAL

```
proc getdate
rem ***** accept date as DD/MM/YYYY *****
local a ,b ,c ,date$ ,separator
let a =0
while a = 4
  let b =0
  while b = 0
    let separator =0
    while separator = 0
      input "date: "; date$
      let c = 1
      let separator =instr(date$ ,"/")
      if separator =0
        print " use '/' to separate day and month"
      endif
    endwhile
    let day =val(date$ (c to separator - 1))
    let c =separator + 1
    let b =instr(date$ (c to ),"/")
    if b =0
      print " use '/' to separate month and year"
    endif
  endwhile
  let separator =separator + b
  let month =val(date$ (c to separator - 1))
  let year =val(date$ (separator + 1 to ))
  let a =len(date$ (separator + 1 to ))
  if a = 4
    print " year should be four figures - eg 1984"
  endif
endwhile
endproc
```

The day, month and year values are left in the variables day, month and year respectively. the values of the variables a, b, c, date\$ and separator are not needed outside the procedure and so can be declared as local. This does mean that the procedure, which is somewhat longer than most, can not easily be split into smaller procedures.

11.6.5 Adding New Records

Now that we have the structure of a record we can write the procedures to accept new data.

```
proc entry
rem ***** add an entry *****
let type$ =""
while type$ = "e"
  cls
  let type$ =""
  while not (type$ = "c" or type$ = "d" or type$ = "e")
    prompt; "debit, credit or end (d/c/e):"
    let type$ =x$
  endwhile
  if type$ = "e"
    let x$ =""
    while not (x$ = "a" or x$ = "r")
      getfids
      prompt; "accept or reject data (a/r):"
    endwhile
    if x$ = "a"
      append
    else
      print "data rejected"
      pause
    endif
  endif
endwhile
endproc
```

PROVISIONAL

This procedure allows you the option of cancelling your input to the new record, in case you make a mistake while typing it in. The actual task of requesting the input is done by `getflds` which is listed below.

```
proc getflds
  rem ***** input field variable values *****
  if type$ = "d"
    ischq
  endif
  getdate
  getnum; "amount",0,1000000
  let amount = num
  input "other details: "; detail$
endproc
```

This procedure uses the error-protected numeric input (`getnum`) that was described in Section 11.6.3.

We have included a further procedure, `ischq`, which asks for you to type in a cheque number only if the debit is a cheque. You may want to include other types of debit – standing orders, for example. For these debits the cheque number is set to zero so that will not display them.

```
proc ischq
  rem **** get cheque no if necessary *****
  let chqno = 0
  let x$ = " "
  while not (x$ = "c" or x$ = "o")
    prompt; "cheque or other (c/o):"
  endwhile
  if x$ = "c"
    getnum; "cheque number",0,999999
  endif
  let chqno = num
endproc
```

The `doreport` procedure of Section 11.5.3 will display all the records in the file in the format that we want. The `report` option of the main menu must select the appropriate records to be displayed. We shall write the procedure so that you can select a particular month, or just display the report for the last month in the file.

11.7 GENERATING A MONTHLY REPORT

We can write the `report` procedure to accept all the input information, leaving the main work of selecting the records to another procedure. We have again used the error-resistant `getnum` to accept input of the month and year. For the year, we have arbitrarily restricted the range to lie between 1950 and 2100. You can, of course, use the editor to change this to match your own needs.

```
proc report
  rem ***** display a monthly report *****
  while not (x$ = "1" or x$ = "p")
    prompt; "latest, or a particular month (l/p):"
  endwhile
  if x$ = "p"
    getnum; "which month",1,12
    let rmonth = num
    getnum; "which year",1950,2100
    let ryear = num
  endif
  monthrep
endproc
```

The following procedure actually produces the report for one month's data. We now have to start considering how we are going to use the additional records, first mentioned in Section 11.5.2, which are to hold the running balance from month to month. These are to be of type "b" and, in general, there will be one of these records per month. When we first ask for the report for a month we shall have to create one of these records to carry the balance to the following month. Each subsequent time we ask for the report for this month we shall not have to create a new record, but will have to check if there have been any changes and modify it if necessary. We shall therefore have to know if the record exists in order to know what to do.

PROVISIONAL

So that we do not have to think of too many things at once, we shall use several further procedures, as shown below.

```
proc monthrep
  rem ***** produce the report *****
  getrecs
  let c1 = count(): rem *** including closing balance(s) ***
  if c1
    select month = maxmth and year = maxyr
    endif
  let c2 = count(): rem *** without closing balance(s) ***
  if c2
    rephead
    doreport
    reset
    dobalancerec
  else
    print "NO RECORDS FOR THIS MONTH"
    reset
  endif
  pause
endproc
```

In this procedure we assume that **getrecs** selects the records we want, including any following balance records. (Remember that if we have not asked for the last month in the file there will be several balance records, one for each following month.) We also assume that the month and year in question are contained in the variables **maxmth** and **maxyr** respectively.

The variables **c1** and **c2** are used to hold the number of records in the selected file, with and without the balance records respectively. Note that the **select** command to remove the balance records can not be:

```
select type$ "b"
```

since we need to keep the one that holds the balance brought forward from the previous month.

Provided the value of **c2** is non-zero we can go ahead with the report. We shall use **title** to display the report title and then we can leave the rest of the display to **doreport**, which we have already written. The task of sorting out what to do about the balance records is left for later, when we decide how to write **dobalancerec**.

The next task is to make the selection of the records, bearing in mind that it must be able to select the last month automatically or any particular month, together with any following balance records. Both options can use the same basic method. The main task can be just to select the records of the last month in the file. If we want the report for a particular month we can first discard all unwanted records and then select the last month.

```
proc getrecs
  rem ***** select all records for the month *****
  cls
  print "selecting the records"
  if x$ = "p"
    select type$ = "b" or (month = rmonth and year = ryear)
    let maxmth = rmonth
    let maxyr = ryear
  else
    rem *** then find the last month in this year ***
    let maxyr = 0
  all
    if year = maxyr and type$ = "b"
      let maxyr = year
    endif
  endall
  select year = maxyr
  let maxmth = 0
  all
    if month = maxmth and type$ = "b"
      let maxmth = month
    endif
  endall
```

PROVISIONAL

```
        endall
      endif
      select 100*year + month = 100*maxyr + maxmth
    endproc
```

Note that the final selection is based on the value of the expression $100 * \text{year} + \text{month}$. This is guaranteed to increase with the (monthly) date of the record, ie January 1985 will give a larger value than for December 1984.

The next procedure just displays the header for the monthly report.

```
proc rephead
  rem ***** display report header *****
  cls
  print tab 20; "Report for "; month : "/" : year
  print
  print
endproc
```

We are now left with the task of dealing with the balance records. There are two main cases;

- 1) The first time a report for a particular month is requested there will be no balance record, so one will have to be created.
- 2) On subsequent occasions the record will exist and it (together with any following balance records) must be updated if there have been any changes to the contents of the report.

The overall task of selecting one of these two cases is done by the following procedure.

```
proc dobalancerec
  rem ***** create a closing balance record *****
  rem ***** or update it if it exists *****
  if c1 = c2 : rem *** no closing balance exists ***
    addbal
  else
    modbal
  endif
  reset
endproc
```

The first option – adding a new record – is quite straightforward. All we have to do is to append a new record, of type "b" and containing the closing balance from the report, to be the first record for the following month.

```
proc addbal
  rem ***** create a new balance record *****
  let day = 1
  let year = maxyr
  let month = maxmth + 1
  if month = 13
    let month = 1
    let year = year + 1
  endif
  let type$ = "b"
  let amount = endbal
  let detail$ = "Brought Forward"
  let chqno = 0
  append
endproc
```

The task of modifying an existing balance record is slightly more complicated. Remember that you may have changed the records for any month, and that any such change will affect the balance records for all subsequent months. The `modbal` procedure locates the record containing the closing balance for the month and checks if it needs modifying. If it does then the correction is applied to all following balance records.

```

proc modbal
  rem ***** update existing closing balance(s) *****
  order year ; a ,month ; a
  getbalancerec
  let difference = endbal - amount
  if difference
    while found()
      let amount = amount + difference
      update
      continue
    endwhile
  endif
endproc

```

The records are sorted into month and year date order and the main work is done by **getbalancerec**, which searches for the first balance record following the reported month. this is the search that is continued by **modbal**, until no more balance records are found.

```

proc getbalancerec
  rem ***** locate the closing balance record(s) *****
  if maxmth = 12
    let maxmth = 0
    let maxyr = maxyr + 1
  endif
  search type$ = "b" and 100*year + month 100*maxyr + maxmth
endproc

```

11.8 CHANGING A RECORD

This example does not include any means of correcting an individual record that has been typed in wrongly. You should not need such a facility very often since the data entry procedures allow you to check the record before accepting it into the file. You could add this facility as a fourth option in the menu – the modular design allows you to do this without affecting any of the other options. Alternatively, you could make such modifications directly on the file, using the **alter** command.

Remember that if you make any change (adding, deleting or altering a record) to an earlier month, it will change the reports for that month and all following months. Any reports that you have previously produced will then be incorrect and you will have to produce them again.

12.1 INTRODUCTION

This chapter extends the explanation of how to use the ARCHIVE programming language by describing how to work with two or more open files. When you have more than one file open at the same time you must be able to identify which file you want to use for any particular operation. You must give each file a unique logical file name when you open or create it and then refer to it by that name in all commands that refer to the file.

Our first example will show you how to add, delete or rename fields within an existing file.

12.2 CHANGING THE RECORDS OF A FILE

Suppose that you want to make some changes to the card index file, described in Chapter 3. This file has fields to hold a person's name, address, home and work telephone numbers, company, birthday and some general notes. If you want to construct a card index solely for personal use you may want to delete the fields containing business information. To cater for friends who live abroad you may want to add a field to contain the country. If you are discarding the work telephone number you can also rename the home telephone number field as just "tel\$".

The most convenient way of changing the file is to create a second file containing the fields you want and then to copy the required records from the old file to the new one. Let us call the new file "friends". The following procedure will do the rest of the work.

```

proc start
  create "friends" as "f"
    fname$
    surname$
    street$
    town$
    county$
    country$
    postcode$
    tel$
  endcreate
  look "index" as "i"
  all ";"
    print at 0,0;i.fname$;" ";i.surname$
    let f.fname$ = i.fname$
    let f.surname$ = i.surname$
    let f.street$ = i.street$
    let f.town$ = i.town$
    let f.county$ = i.county$
    let f.postcode = i.postcode
    let f.note$ = i.note$
    let f.tel$ = i.hometel$
    let f.country$ = " "
    append "f"
  endall
  close "f"
  close "i"
  print
  print "DONE"
endproc

```

You can see, from the previous example, that you can use the same name for a field in both files -- they can be distinguished by including the logical file name. If you do not include the logical file name then it will be assumed that the *current file* is to be used. The last file to be opened automatically becomes the logical file. In this example the current file will be "index" (with logical file name "i") so we could make use of this by writing the procedure as:

12.3 THE CURRENT FILE

```

proc start
  create "friends" as "f"
  fname$
  surname$
  street$
  town$
  county$
  country$
  postcode$
  tel$
  endcreate
  look "index" as "i"
  all
  print at 0,0;fname$;" ";surname$
  let f.fname$ = fname$
  let f.surname$ = surname$
  let f.street$ = street$
  let f.town$ = town$
  let f.county$ = county$
  let f.postcode = postcode
  let f.note$ = note$
  let f.tel$ = hometel$
  let f.country$ = " "
  append "f"
  endall
close "f"
close
print
print "DONE"
endproc

```

If you do not include the logical file name in any case where it is optional, ARCHIVE will assume that the command refers to the current file. It is usually safer to include the logical file name explicitly, to avoid any possibility of confusion.

You can, at any time, specify the current file by means of the **use** command. If you included the command:

```
use "f"
```

in the above example, then "friends" would be the current file until you changed it again, either by opening another file or by means of the **use** command.

12.4 STOCK CONTROL

12.4.1 Introduction

In a stock control system you will need to:

- 1) Find information on a particular stock item.
- 2) Obtain a report on the current stock levels of all items.
- 3) Record sales and modify the stock records accordingly.
- 4) Order new supplies, to maintain adequate stock levels.
- 5) Record deliveries of stock.

You will obviously need a file to hold the details of all items held in stock and it is convenient to have a second file to hold details of all your suppliers. You will need to be able to access either file from the other — for example you may want to know all the possible suppliers of a particular item, or to find out what items are supplied by a particular company.

In order to keep the application as simple as possible we shall not use the menu-driven approach of the examples in the previous two chapters. We shall write it as a series of separate commands which can be used — like the standard commands — by typing their names.

Since the procedures will be strongly dependent on the file structure we use, we must first give some thought to their appearance.

12.4.2 The Stock File

The stock file must contain full details of the stock situation for each item. The following list explains all the fields we shall use.

PROVISIONAL

Field Name	Use	Example
code\$	The internal stock code	A101
description\$	Item description	Widget, large
qty	Number in stock	500
sellpr	Selling price	1.25
reorderlev	Reorder when stock level falls below this value	200
buyqty	How many to order	400
supplier\$	Internal supplier code(s)	a,b,c

We can create the file by:

```
create "stock" as "sto"
  code$
  description$
  qty
  reorderlev
  sellpr
  buyqty
  supplier$
endcreate
```

In addition to the items supplied we shall need to include the company's name, address and telephone number. It will be useful also to include the name of a contact person in the company. We shall use the following fields:

12.4.3 The Supplier File

Field Name	Use	Example
coname\$	The company's name	Large Widgets plc
street\$	First line of address	
town\$	Second line of address	
county\$	Third line of address	
postcode\$	Last line of address	
contact\$	Name of a contact	
tel\$	Telephone number	021-356 1234 ext. 212
code\$	Your code for the company a	
item\$	List of your stock codes for the items supplied	A101,B236,C659
scode\$	List of the supplier's codes	11-30X,55147Z,33-28Q
price\$	List of the supplier's prices	0.85,1.37,5.22
delay\$	List of delivery times in days	10,28,14

The last four fields contain lists of items, separated by commas. There must be the same number of items in each list and the order of the items should correspond. For example, the item which you refer to as B236 has a supplier's code 55-47Z, the cost to you is 1.37 and the delivery time is twenty eight days.

We can create the file by:

```
create supplier" as "sup"
  coname$
  street$
  town$
  county$
  postcode$
  contact$
  tel$
```

```
code$
item$
scode$
price$
delay$
endcreate
```

12.4.4 Enquiries You will find that the most frequently-needed facility is to find information about a particular stock item, in response to customer enquiries. You will need to find the information as quickly as possible, so we must use the **locate** command. Remember that the command will not work unless the file has been sorted on the field(s) used with **locate**. In this case the stock file must have been sorted by means of:

```
use "sto"
order code$;a
```

The procedure can be quite simple:

```
proc query
  input "Stock code? " ; stcode$
  use "sto"
  locate stcode$
  if found()
    display
    sprint
  else
    print stcode$ ; " does not exist"
  endif
endproc
```

We could use the **find** command to search for the name of the item. In a large file this would be much slower since it examines all string fields in each record of the file, until a match is found.

12.4.5 Stock Report We can also write a simple procedure to produce a general stock report.

```
proc report
  print chr(12) ; rem *** form-feed for a new page ***
  print tab 2 ; "ITEM" ; tab 11 ; CODE ;
  print tab 15 ; "QUANTITY" ; tab 20 ; "PRICE" ;
  print tab 28 ; "STOCK VALUE"
  print
  let total = 0
  use "sto"
  all
    print description$ ( to 10); " "; code$ ; tab 15; qty ;
    print tab 20; sellpr ; tab 30; sellpr *qty
    let total = total + sellpr *qty
  endall
  print
  print "Total stock value = "; total
endproc
```

12.4.6 Recording Sales All we need to do to record a sale is to subtract the number of items sold from the relevant stock record. It is advisable to include some form of confirmation that we are dealing with the right stock item and that the stock is sufficient to meet the order.

```
proc sale
  query
  print "how many? " ; num
  if num = sto.qty
    print num ; " * " ; sto.code$ ; "(" ; sto.description$ ;
  ),
  print "Order value:- " ; num*sto.sellpr
  print "Confirm ((y/n))" ;
  if lower(getkey())="y"
    let sto.qty = sto.qty - num
    update
  endif
  else
    print "Not enough stock"
  endif
endproc
```

PROVISION

The following procedure allows you to record the delivery of stock. Again it requests confirmation of the details you type in before accepting them and updating the relevant stock record.

12.4.7 Recording Incoming Stock

```
proc delivery
  query
  input "Number of items? " ; num
  print num ; " * " ; sto.code$ ; "(" ; sto.description$ ;
  "("
  print "Confirm (y/n) " ;
  if lower(getkey())="y"
    print "Accepted"
    let sto.qty = sto.qty + num
  endif
endproc
```

So far our procedures have only referred to the stock file. When we want to order more stock we shall have to refer to the supplier file for the name and address of the company, the price, and so on.

12.4.8 Ordering New Stock

Our first task will be to identify those companies that sell a particular item:

```
proc supplier
  use "sup"
  select instr(sto.supplier$ ,sup.code$ )
endproc
```

This procedure simply selects the records from the supplier file whose codes are included in the string of codes in the supplier\$ field of the current stock record.

We shall now want to find the details of the price, delivery time and manufacturer's reference code from the lists in the supplier records. What we need is a table-lookup procedure which will locate a specific item in one list and give you the corresponding item from one of the other lists.

```
proc lookup; seek$ ,within$ ,from$
  local index ,num ,offset
  rem *** find the character count to the start of ***
  rem *** seek$, in within$ ***
  let offset = instr(within$ ,seek$ ) - 1
  rem *** now count the commas up to that point ***
  let num = 0
  let index = 0
  while num < offset
    let num = num + instr(within$ (num + 1 to ),",")
    let index = index + 1
  endwhile
  rem *** step over that many commas in from$ ***
  let offset = 0
  while index < 0
    let offset = offset + instr(from$ (offset + 1 to ),",")
    let index = index - 1
  endwhile
  rem *** take the rest of the string ***
  let result$ = from$ (offset + 1 to )
  rem *** check for any comma ***
  let offset = instr(result$ ,",")
  if offset
    rem *** throw away the comma and all ***
    rem *** following text ***
    let result$ = result$ ( to offset - 1 )
  endif
endproc
```

This procedure takes three parameters:

- seek\$ – the text to be located,
- within\$ – the text string to be searched for seek\$
- from – the text from which the corresponding item is to be extracted.

PROVISIONAL

and leaves the found text in the string variable result\$. If, for example, we define the following three string variables:

```
let a$ = "B236"  
let b$ = "A101,B236,C659"  
let c$ = "10,28,14"
```

and then call lookup as:

```
lookup; a$,b$,c$
```

we shall find that result\$ has the value "28". You can recover the numeric value by using val(result\$).

Let us use this procedure to find the supplier with the cheapest price.

```
proc cheap  
  use "sup"  
  lookup; sto.code$,sup.item$ ,sup.price$  
  let lowest = val(result$)  
  let name$ = sup.coname$  
  all "sup"  
    lookup; sto.code$,sup.item$ ,sup.price$  
    if val(result$) < lowest  
      let lowest = val(result$)  
      let name$ = sup.coname$  
    endif  
  endall  
  locate name$  
endproc
```

And how about the one with the fastest delivery?

```
proc fast  
  use "sup"  
  lookup; sto.code$,sup.item$ ,sup.delay$  
  let lowest = val(result$)  
  let name$ = sup.coname$  
  all "sup"  
    lookup; sto.code$,sup.item$ ,sup.delay$  
    if val(result$) < lowest  
      let lowest = val(result$)  
      let name$ = sup.coname$  
    endif  
  endall  
  locate name$  
endproc
```

We can now write the procedure to order an item.

```
proc doorder  
  query  
  supplier  
  print "fast or cheap (f/c)";  
  let reply$ = lower(getkey())  
  if reply$ = "f"  
    fast  
  else  
    cheap  
  endif  
  doform  
  reset  
endproc
```

The procedure **doform** produces the actual order form. You should modify it to your own requirements. We shall use a simple version which shows the order details on the display.

PROVISIONAL

```
proc doform
  print
  print sup.coname$
  print sup.street$
  print sup.town$
  print sup.county$
  print sup.postcode$
  print
  print "Please supply " ; sto.buyqty ;
  print " * part number " ;
  lookup; sto.code$,sup.item$,sup.scode$ ;
  print result$
  lookup; sto.code$,sup.item$,sup.price$
  let pri = val(result$)
  print " at " ; pri ; " each."
  print
  print "Total value: " ; sto.buyqty * pri
endproc
```

Finally we can write a short procedure to run the application. It must open both files with the correct logical file names, clear the display and show you the additional commands that you have.

```
proc start
  open "stock.dbf" as "sto"
  open "suppliers.dbf" as "sup"
  clear
endproc
```

The `clear` procedure simply clears the screen and shows a list of the extra commands available:

```
proc clear
  cls
  print "QUERY REPORT DELIVERY DOORDER SALE"
endproc
```

...

...

...

...

...

...

...

...

...

...

...

...

...

...

)

)

)

)

The five function keys are used in ARCHIVE for the following purposes:

Key	Use
F1	Help
F2	turn the prompts on and off
F3	call a command menu edit command - call command menu sedit command - select colour
F4	edit command - insert text sedit command - reserve space
F5	edit last line of input edit command - edit current line sedit command - leave sedit

Pressing **F1** displays a Help screen, containing information relevant to your current action and your possible options. You can ask for further information on any of the topics listed at the bottom of the display by typing in its name. Just pressing **ENTER** goes back one level in Help, until you reach the level at which you entered Help, when you will be returned to the exact point from which you left. You can return immediately from any level of Help by pressing **ESC**.

13.2 HELP

You can turn off the display of the control area and the prompts that it contains by pressing **F2**. This allows you to see more of your work on the display. You can restore the display of the control area by pressing **F2** again - each press changes the state of the control area display between on and off.

13.3 THE PROMPTS

Variable names may be up to thirteen characters in length, and must not start with a digit (0 TO 9). They may contain any combination of characters, except that '\$' and '.' have special meanings.

13.4 VARIABLES

If a variable name ends with a '\$' it is a string variable. Strings may be up to 256 characters in length. If the name does not end with a '\$' the variable is numeric. A variable name may refer to the contents of a record in a file and is then known as a field variable. Field variables are normally assumed to refer to the current file but may be made to refer to another open file by including a logical file name, separated by a '.' from the variable's name. Such a field variable is written as:

logical-file-name.field name

e.g. main.surname\$

If a variable name includes a dot then it must refer to a field in an open file. If there is no dot an attempt is made to match the name to an existing variable in the following sequence:

- 1) a field of the current file
- 2) a local variable (a parameter in the current procedure, if any)
- 3) a global variable

An error message is given if no match is found.

13.5 FILES There are two aspects of files in ARCHIVE. The first concerns the structure of the files you use to hold your information. The second is the way in which Microdrive files are named. The method of naming files applies to all files, including procedure files, screen format files and to the ARCHIVE program itself.

**13.5.1 ARCHIVE
Data Files**

13.5.1.1 A Field A *field* is the space reserved to hold either a string or a number.

In ARCHIVE, each field is identified by a field variable name, as described in Section 12.4. Whether a particular field can hold a string or a number is dependent on the name given to the field at the time it was created - string fields have a name ending with a \$.

13.5.1.2 A Record A *record* is a collection of fields, whose contents are related in some way. The fields of a record might, for example, be used to hold the name, the address and the telephone number of a particular person.

In ARCHIVE the records are of *variable length* so that each record only takes up as much room as is necessary to hold the information contained in its fields.

13.5.1.3 A File A *file* is made up from a number of related records. To continue the example of Section 12.5.2, a file could consist of a collection of name, address and telephone number records for many different people.

A file is the basic unit that you can save to or load from a Microdrive cartridge.

Each file has a name to identify it. In ARCHIVE you give a name to the file when it is created, but you can change the name at any time.

13.5.1.4 Opening and Closing Files When you want to read from or write to a file you must first *open* it. Generally speaking, opening a file transfers a copy of the file from the Microdrive cartridge into memory although, in the case of a long file, it is possible that only part of the file will be present in memory at any one time.

You can open a file in *read only* mode which, as its name suggests, means that you can not change its contents. You also have the option of opening a file in *update* mode, when you are allowed both to read and to change its contents.

Every time you open a file, ARCHIVE reserves space for the field variables needed by a record of the file. The field variables always contain the values of the current record within the file.

When you close a file any changes that you have made are copied into the file stored on the Microdrive cartridge and then the copy held in memory is discarded. Closing a file is the only way of ensuring that the copy on the Microdrive cartridge contains your latest version.

When you leave ARCHIVE by means of the **quit** command all open files are closed automatically.

13.5.1.5 Logical File Names Each open file has an associated *logical file name*, given to it when the file is opened. If you do not specify a logical file name when you open the file, it is given the default name "main".

The logical file name is used to identify a particular file when you are using several files at once.

13.5.2 Microdrive Files

13.5.2.1 File Names A full file name consists of three sections, separated by underscores. The three components are:

- | | |
|------------------------------------|---------|
| an optional drive specifier | eg MDV1 |
| a file name of up to x characters | eg FRED |
| an optional three-letter extension | eg DBF |

A full file name for an ARCHIVE file could therefore be:

MDV2__FRED__DBF

PROVISIONAL

If you do not include a drive specifier in a file name then ARCHIVE assumes that you are referring to the current drive, that is, the drive that was last used. The one exception is when you are loading ARCHIVE itself from SuperBASIC, as described in Section 2.1. In this case you must include the drive specifier in the file name.

You do not normally need to specify an extension since ARCHIVE supplies a default extension for every file access. The **look**, **open**, **close** and **create** commands work on data files with an assumed extension of `__DBF`. The **load** and **save** commands supply a default extension of `__DBL` to the program files. The default extension for Import and Export files is `__EXP`, and when you Print to a file the default extension is `__LIS`. Screen format files, are loaded and saved by the **sload** and **ssave** commands, which assume an extension of `__DBS`.

If you include an extension in any file name you type in then it will be used in preference to the default extension normally provided by ARCHIVE.

Every time that an ARCHIVE command asks you to type in a file name you have the option of pressing the ? key to obtain a list of the names of files on the current drive. The file name `"*__*"` (file name and extension) will appear in the input line and, if you accept this by pressing **ENTER**, you will be given a list of all files on the current drive.

In this context the `"*"` character is a *wild card* which stands for any sequence of characters. You may also use the character `"?"` to represent any single character in a file name.

13.5.2.2 Wild Cards

You have the option of using the line editor to modify the suggested file name, in order to obtain a list of the names of a particular group of files.

If, for example, you edit the file name to read `"*__TST"` and then press **ENTER** you will be given a list of the names of all files with an extension of `__TST`. Changing the file name to `"X*__*"` would result in a listing of all files, with any extension, whose names begin with X.

You could use the single character wild card `"?"` as, for example,

```
MYFILE?__*
```

which would result in a listing of all files with names such as:

```
MYFILE1 MYFILE2 MYFILE3
```

and so on, with any extension.

Note that this facility is only available when you are requesting a list of file names before typing in a file name for any of the file-based commands (files, load, save, print and so on).

Any file (of any type - eg a data file, a program file or a screen format file) whose name starts with a dollar sign (\$) is a *memory file*. Such files are always stored in the computer's memory and never on a Microdrive cartridge. Otherwise they work in exactly the same way as any other file of the same type.

13.5.3 Memory Files

You may want to use a memory file for information that you are frequently loading, such as a designed screen format. Each time you use the **display** command ARCHIVE will replace your design with its own layout and you will have to use **sload** to recover your own.

If you **ssave** your design as a memory file - for example:

```
ssave "$scr"
```

it will be saved in memory instead of on a Microdrive cartridge. It will then load almost instantaneously each time you load it with:

```
sload "$scr"
```

Memory files do, of course, use part of the computer's memory for their storage so you should not save long files as memory files.

Also, remember that they are never transferred to a Microdrive cartridge and will be destroyed when you leave ARCHIVE. If you want to keep a permanent copy you should

use the **backup** command to copy the memory file to a new file whose name does not start with a dollar sign. For example:

```
backup "$scr" ENTER "scr" ENTER
```

The file named "scr" is saved on the current Microdrive cartridge.

If, on a later occasion, you want to load the file as a memory file you can again use the **backup** command:

```
backup "scr" ENTER "$scr" ENTER
```

The file "\$scr" is created as a memory file, but must still be loaded with (since it is a screen format) the **sload** command.

13.6 FORMULAE

A formula is any allowed combination of functions, cell references, labels and arithmetic operators. Examples are:

```
A1*B3
month(col()-1)
if(instr(B6,"is"),1,0)
rept("=",len(G23))+":"
```

13.7 ARITHMETIC

The arithmetic operations in ARCHIVE follow the same rules as for the arithmetic in SuperBASIC. The valid range for numbers is from $-2.9E-39$ to $+1.7E+38$. All calculations are accurate to 17 significant digits but only a maximum of 16 significant digits may be displayed.

The following arithmetic operations are provided:

+	Addition (on numbers), or concatenation (on strings)	
-	Subtraction	
*	Multiplication	
/	Division	
^	Raising to a power	
=	Equal	
>	Greater than	Both operands must be of the same type. The result is a number, 1 if the comparison is true and 0 if it is not.
<	Less than	
>=	Less than or equal to	
<=	Greater than or equal to	
<>	Not equal to	

Functions and operations have the following priorities:

Operation	Priority
Subscripting and slicing	12
All functions	11
^	10
Unary minus (ie, minus just used to negate something)	9
*,/	8
+,- (minus used to subtract one number from another)	6
=, >, <, <=, >=, <>	5
not	4
and	3
or	2

In addition, string slicing is provided, again in a form similar to that of SuperBASIC. The slicing operations provided are:

- (n) select the nth character.
- (n to m) select all characters from the nth to mth character inclusive.
- (n to) select from character n to end.
- (to m) select from the beginning to the mth character.

13.7.1 THE SCREEN

This section will document the screen format, as implied by the **SLOAD**, **SSAVE**, **SPRINT** and **SCREEN** commands.

13.8 SYNTAX

Each command in Section 12.8 is accompanied by a statement of its *syntax* (the exact structure of the command). This section describes the notation used to express the syntax.

PROVISIONAL

Symbol	Meaning
< >	syntactic entity
	optional item
()	any number of
	or
<var>	variable name, either string or numeric
<exp>	expression, either string or numeric
<n.exp>	numeric expression
<s.exp>	string expression
<lfn>	logical file name
<fnm>	physical file name
<prn>	procedure name
<s.lit>	string literal
<ptm>	print item

13.8.1 SYNTAX CONVENTIONS

A print item is one of four possibilities: at, tab, ink or paper. A full description of a print item is, in our syntax notation,

`at <n.exp> , <n.exp> | tab <n.exp> | ink <n.exp> | paper <n.exp>`

As an example, consider the syntax of the **order** command. In our notation it appears as:

`order <var>;a|d (, <var>;a|d)`

Order therefore needs to be followed by at least one variable name, separated by a semicolon from a letter which must be either 'a' or 'd'. In addition you may optionally include any number of pairs of a variable name and a letter, provided each pair is separated by commas. Clearly, the syntax notation provides a much more compact description.

Note that the syntax notation does not tell you the meaning or purpose of the symbols - you will have to read the rest of the description for each command. The syntax only gives you a formal description of the number and kind of items that go to make up a valid command.

The following commands are available

13.8.2 Syntactic Entities

13.9 THE COMMANDS

ALL Syntax: `all | <lfn> :...:endall`

It scans through logically present records of the file in the fastest possible time. This scan will not, in general, be in any particular sequence. The optional logical file name will force it to refer to a specified open file. If the logical file is not given it will scan on the current file.

ALTER Syntax: `alter`

Displays a list of the field variable names in the current file, together with their values in the current record. You can change the contents of any one or more fields. First select the field to change by pressing **TABULATE** until the cursor is at the correct field. You can then use the line editor to modify the value. Press **TABULATE** to mark the end of your changes to the field and move on to the next. (**SHIFT** and **TABULATE** moves back to the previous field.)

When you have made all the changes you want you should press **ENTER** to replace the old record with the new one. If the file is ordered the new version of the record is inserted in sequence, otherwise the insertion takes place at an unspecified position.

APPEND Syntax: `append | <lfn> |`

Adds a record to the specified file, or to to the current file if the logical file name is not given. The fields of the record are given the current values of the field variables. If the file is ordered the insertion is in sequence, otherwise the insertion takes place at an unspecified position.

BACK Syntax: `back | <lfn> |`

Moves backwards one record in the specified file, or in the current file if the logical file name is not given.

- BACKUP** Syntax: backup <fnm>
Makes a copy of the specified file. You should make copies of all your files, to protect against accidental damage or erasure.
- CAT** Syntax: cat
Displays a list of all files on the current Microdrive cartridge.
- CLOSE** Syntax: close | <lfm> |
Closes the specified file, or the current file if no logical file name is specified.
- CLS** Syntax: cls
Clears the display area and switches off any display screen. See screen, load, sprint.
- CONTINUE** Syntax: continue
Continues the previous search or find, from the record following the current record in the current file.
- CREATE** Syntax:
create <fnm> | as<lfm> | :<v!r> (:<var>) :endcreate
It creates a named open file whose records contain the fields given by the list of variables specified in the command. You have the option of specifying a logical file name - if you do not the file is created with the logical file name "main".
- DELETE** Syntax: delete | <lfm> |
Deletes the current record from the specified file, or from the current file if no logical file name is given.
Use this command with care since you can not recover the deleted record.
- DISPLAY** Syntax: display
Shows the logical file name of the current file and a list of the field names and the values of the field variables for the current record. If the file is sorted it also shows the sort fields and their sort priority.
The command replaces any existing user-defined screen format with this list, which becomes the active screen format.
- EDIT** Syntax: edit
Calls the procedure editor to create a new procedure or to edit an existing procedure.
- ENDALL** See ALL.
- ENDCREATE** See CREATE.
- ERASE** Syntax: erase <fnm>
Erases the specified file from the Microdrive cartridge.
Use this command with care since you can not recover the erased file.
- ERROR** Syntax: error <pnm> <exp> ; <exp>
Marks a procedure for the purposes of error-handling. Any error which occurs during the execution of this procedure, or any other procedure which it calls, causes a premature return from the marked procedure. The procedure can determine the nature of the error by using the errnum() function to read the error number. This error number is cleared each time that **error** is executed.

PROVISIONAL

EXPORT

Syntax: export <fnm>

Saves the specified ARCHIVE file on a Microdrive cartridge in a form suitable for use by the other packages.

FIND

Syntax: find <s.exp>

Rewinds the file to the beginning and searches for the first record containing a match to the specified string in any string field. The match is independent of upper or lower case text.

You can continue the search with the CONTINUE command, and determine whether the search was successful by examining the value returned by the found() function.

FIRST

Syntax: first [<lfm>]

Sets the file pointer to the first record of the specified file, or the current file if no logical file name is specified.

IF

Syntax: if <n.exp> : ... | : else : ... | : endif

1) Without the optional ELSE. If the expression is non-zero the following statements are executed. If the expression is zero execution transfers to the statement following ENDIF.

2) With the optional ELSE. If the numeric expression is non-zero the statements between IF and ELSE are executed. Otherwise the statements between ELSE and ENDIF are executed. In either case execution continues with the statements following ENDIF.

IMPORT

Syntax: import <fnm> create <fnm> | as <lfm> |

Reads a file produced by one of the other packages and produces a file suitable for use with ARCHIVE.

INK

Syntax: ink <n.exp>

Sets the foreground colour for all following text to the colour specified by the value of the expression. The colours are:

0 and 1 black
2 and 3 red
4 and 5 green
6 and 7 white

If the expression evaluates to more than 7, the value taken is the remainder after division by 8, i.e. ink 9 is equivalent to ink 1, both setting the print colour to black. If ink is used within a print command it will only change the print colour for the duration of that command.

INPUT

Syntax:

input | <var> | <s.lit> | <ptm> | (; <var> | <s.lit> | <ptm>) || ; |

Requests input from the keyboard to one or more variables. Each variable in an input list may be preceded by a initial string which will be displayed as a prompt for the input. All input items must be separated from each other by semicolons. If the list has a final semicolon the cursor will not move to a new line after the input.

The list of input items may include the cursor-positioning items

at line,column
tab column

The first of these positions the cursor at the specified line and column position, and tab moves the cursor to the specified column within the current line. If the cursor is already to the right of the specified column, tab will have no effect.

These two items may not be used outside an input or a print command.

PROVISIONAL

You may also use ink and paper as input items. If used within an input command they will only affect the ink and paper colours to the end of the input, when the colours will return to their original settings.

- INSERT** Syntax: insert
- Displays a list of the field variable names in the current file and requests you to type in a value for each field variable. You should end the input to each variable by pressing **TABULATE** which also moves on to the next field. (**SHIFT** and **TABULATE** moves back to the previous field.)
- When you have given values to all the field variables you should press **ENTER** to insert the new record into the current file. If the file is ordered the record is inserted in sequence, otherwise the insertion takes place at an unspecified position.
- LAST** Syntax: last | <lfn> |
- Sets the file pointer to the last record of the specified file, or the current file if you do not specify a logical file name.
- LET** Syntax: let <var> = <exp>
- Used to assign a value to a variable (as in SuperBASIC).
- LLIST** Syntax: llist
- Lists all the procedures currently in memory on a printer.
- LOCAL** Syntax: local <var> | (, <var>) |
- Within a procedure, forces the following list of variables to be local variables. These variables exist only within the procedure in which they are declared and are undefined in any other procedure. Their values are destroyed on exit from the procedure.
- LOCATE** Syntax: locate <exp> | (, <exp>) |
- Finds the first record whose field contents match the expression(s). The record is located much more quickly than if you used **find**, but the file must first have been sorted. Each expression must explicitly refer to the contents of a particular sort field. In the case of a string field the match is case-dependent.
- If you have ordered the file on more than one field you can specify several fields to locate (one for each sort field) separated by commas. The order of the items in the list must correspond to the order in the preceding order command. For example,
- ```
order animal$; a , colour$; a
locate "Elephant" , "grey"
```
- will find the first record in which the fields animal\$ and colour\$ contain the text "Elephant" and "grey" respectively
- LOAD**        Syntax: load <fnm>
- Loads the specified procedure file from a Microdrive cartridge into memory.
- LOOK**        Syntax: look fnm as lfn
- Opens the named file for read access only. If the logical file name is not specified it is given the default value "main".
- LPRINT**      Syntax: lprint exp | ptm ( ; exp | ptm ) ;
- Displays the values of the following list of items on a printer, in the same way as for **PRINT**.

# PROVISIONAL

- MODE** Syntax: mode <var>, <var>
- Changes the form of the display. The first variable may have a value of 0 or 1. A value of 0 joins the control, display and work areas into a single region. A value of 1 separates them back into three distinct areas.
- The second variable may have a value of 4, 6 or 8 and switches the display between showing 40, 64 or 80 characters per line.
- The initial setting, when you load ARCHIVE, is equivalent to:
- mode 1,8**
- NEW** Syntax: new
- Closes all files and deletes all procedures, ready for a fresh start. Any open files are NOT updated when they are closed.
- NEXT** Syntax: next | <lfn> |
- Moves the file pointer to the next record in the specified file, or in the current file if you do not specify a logical file name.
- OPEN** Syntax: open <fnm> | as <lfn> |
- Opens the specified file for both reading and writing. The file is given a logical file name "main" if you do not specify one.
- ORDER** Syntax: order <var>;a|d| (, <var>;a|d) |
- Orders the records of the file according to the contents of the specified fields. The first field specified in the list is the primary sort field. Records which have equal contents of their primary sort field are further sorted according to the contents of the next field in the list (if it is specified) and so on. For each specified field an ordering direction must be given. This must be either "a" or "d" to specify ascending or descending order respectively.
- PAPER** Syntax: paper <n.exp>
- Sets the background colour for all following text to the colour specified by the value of the expression. The colours are:
- 0 and 1 black**  
**2 and 3 red**  
**4 and 5 green**  
**6 and 7 white**
- If the expression evaluates to more than 7, the value taken is the remainder after division by 8, i.e. paper 11 is equivalent to paper 3, both setting the print colour to red.
- If paper is used within a print command it will only change the background colour for the duration of that command.
- POSITION** Syntax: position <n.exp>
- Makes the record whose record number is given by the expression the current record.
- PRINT** Syntax: print | <exp> | <ptm> | (;<exp> | <ptm>) | ; |
- Displays the values of the following list of items, which must be separated by semicolons. If the list has a final semicolon the cursor will not move to a new line after the display.
- QUIT** Syntax: quit
- Closes all files and returns to SuperBASIC.

# PROVISIONAL

|               |                                                                                                                                                                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>REM</b>    | Syntax: rem<br><br>When used within a procedure, it marks the rest of the line as containing a comment. Any following text on that line is ignored when the procedure is executed.                                                                                                                                  |
| <b>RESET</b>  | Syntax: reset<br><br>This command restores all the records in the current file which were removed by an earlier use of select.                                                                                                                                                                                      |
| <b>RETURN</b> | Syntax: return<br><br>Used within a procedure to cause an immediate termination of the procedure by returning to the calling procedure.                                                                                                                                                                             |
| <b>RUN</b>    | Syntax: run <fnn><br><br>Loads the specified procedure file into memory and starts execution of the procedure called "start".                                                                                                                                                                                       |
| <b>SAVE</b>   | Syntax: save <fnn><br><br>Saves all procedures currently in memory as a named file on a Microdrive cartridge.                                                                                                                                                                                                       |
| <b>SCREEN</b> | Syntax: screen<br><br>Displays the formatted screen layout previously SLOADed. It does nothing if there is no screen layout present. It does not display any of the variables in the screen.                                                                                                                        |
| <b>SEARCH</b> | Syntax: search <n.exp><br><br>Searches the current file from the beginning until a record is found in which the specified expression is true. This record becomes the current record.                                                                                                                               |
| <b>SEdit</b>  | Calls the screen editor, to enable you to define a new screen layout. See Chapter 7.                                                                                                                                                                                                                                |
| <b>SELECT</b> | Syntax: select <n.exp><br><br>Scans the whole file selecting only those records for which the specified expression is true. The file then behaves as if only the selected records are present.<br><br>You can restore all the discarded records with the reset command.                                             |
| <b>SINPUT</b> | Syntax: sinput <var>  (,<var>) <br><br>Waits for input to one or more variables in the following list. All the variables in the list must be currently displayed in an active screen format.                                                                                                                        |
| <b>SLOAD</b>  | Syntax: sload <fnn><br><br>Loads a previously defined and saved display screen format. It also displays this screen layout and activates the display of any variables within the screen. The displayed values are then updated automatically whenever control returns from a procedure to the keyboard interpreter. |
| <b>SPRINT</b> | Syntax: sprint<br><br>Forces a display of the fields of the current record. There must be an active screen format (the screen format is made active by a previous use of screen, sload or display). If there is no active screen format the command will have no effect.                                            |
| <b>SSAVE</b>  | Syntax: ssave <fnn><br><br>Saves, as a named file on a Microdrive cartridge, the current display area as a defined screen format. It saves the text of the screen and a list of the variables in the display, together with their positions.                                                                        |

# PROVISIONAL

- STOP** Syntax: stop  
Terminates the execution of all procedures and returns control to the keyboard interpreter.
- TRACE** Syntax: trace  
Switches the trace mode on and off. In trace mode each line of the program is displayed in the work area of the screen, as it is executed.
- UPDATE** Syntax: update [ <lfn> ]  
Replaces the current record in the specified file (or the current file if no logical file name is given) with a record containing the current values of the field variables.
- USE** Syntax: use [ <lfn> ]  
Makes the specified file the current file.
- WHILE** Syntax: while <n.exp> : ... : endwhile  
Repeatedly executes the statements between WHILE and ENDWHILE for as long as the value of the expression is non-zero (true).

You can think of a function as a kind of recipe which converts one or more initial values, known as the function's arguments, into a different value, which is said to be the value that is returned by the function.

The functions provided by ARCHIVE may take three, two, one or no arguments. The arguments for a function are placed in brackets after its name. You must not leave a space between the name and the opening bracket, but spaces are allowed between items within the brackets. If a function takes more than one argument, the arguments are separated by commas. All functions must be followed by the brackets, even if they take no arguments. The presence of the brackets is a useful reminder that you are referring to a function. They allow you to distinguish between a variable and a function, even if they have the same name.

## 13.10 FUNCTIONS

In the descriptions of the functions;

**n.exp** is a numeric expression,  
**s.exp** is a text string expression,  
**lfn** is a file identifier.

The following functions are provided.

- ABS** Returns the absolute value of the argument.  
( n.exp )
- CHR** This function returns the ASCII character whose code is n. Characters with ASCII codes less than 32 have no effect on the screen, but may be sent to the printer if they are preceded by an ASCII null, ie, chr(0). For example, chr(0) + chr(13) passes the ASCII character for a carriage return to a printer. This is useful if your printer needs control code sequences to produce special effects - refer to your printer manual for any special codes that it needs.  
( n.exp )
- You can, for example, send an "A" to the screen with: print chr(65)
- CODE** This returns the ASCII value of the first character found in the specified text.  
( s.exp )
- COUNT** Returns the count of the number of records in the current file.  
( lfn )
- DATE()** Returns today's date as a text string in the form:  
"DD/MM/YYYY"
- You must first have set the system clock, as described in the technical manual.
- EOF( lfn )** Returns a value indicating the position of the record pointer in the current file, or the specified file if a file identifier is given. The value returned is one if you have attempted to read past the end of the file, otherwise it is zero.

- ERRNUM()** Returns the number of the last error which occurred (an error number of zero indicates no errors). The error number is the same as that displayed together with the error message when ARCHIVE reports a detected error.
- FIELDT** ( n.exp , lfn )  
Returns the type of the specified field in the current record of the specified file (or the current file if no logical file name is given).
- FIELDV** ( n.exp , lfn )  
Returns the value of the specified field in the current record of the specified file (or the current file if no logical file name is given).
- FOUND()** Returns one if a record is found by use of SEARCH or FIND, otherwise returns zero.
- GETKEY()** Waits for a key to be pressed and returns the ASCII code of the key.
- INKEY()** Returns the ASCII code of any key pressed at the time the function is called - it does not wait for a keypress but will return zero if no key is pressed.
- INSTR** ( s.exp1 , s.exp2 )  
This finds the first occurrence of s.exp2 within s.exp1 and returns the position of the first character of s.exp2 in s.exp1 . It will return a value of zero if no match is found. The match is case-dependent.  
**instr("January","Jan") returns 1**  
**instr("January","an") returns 2**  
**instr("January","AN") returns 0**
- INT( n.exp )** Returns the integer value of the number, by truncating at the decimal point. The truncation always operates towards zero. Thus;  
**int(3.7) returns 3**  
**int(-4.8) returns -4**
- LEN( s.exp )** Returns the number of characters in the specified text.
- LOWER** ( s.exp )  
Converts the specified text to lower case.
- MEMORY()** Returns the number of unused bytes of memory remaining.
- MONTH** ( n.exp )  
Returns, as text, the name of a month.  
For example month(3) returns the text "March".  
If an argument larger than 12 is used, it is replaced by the remainder after division by 12 so that, for example, month(13) and month(1) will both give the result "January".
- NUMFLD** ( lfn )  
Returns the number of fields in the records of the specified file (or the current file if you do not give a logical file name).
- RECNUM** ( lfn )  
Returns the number (counting from zero at the first record) of the current record of the specified file (or the current file if you do not give a logical name).
- REPT** ( s.exp , n.exp )  
This function returns a string consisting of a number of copies of the given text, which may be up to 255 characters in length. For example, print rept(" ",5) will print five asterisks print rept("abc",3) prints three repetitions of "abc".

# PROVISIONAL

|                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SGN</b><br>( n.exp )                    | Returns +1, -1 or 0, depending on whether the argument is positive, negative or zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>STR</b><br>( n.exp1 , n.exp2 , n.exp3 ) | Converts a number, n.exp1 , to the equivalent text string. The second parameter indicates the form of the converted string as follows;<br><ul style="list-style-type: none"><li>0 decimal (floating point)</li><li>1 exponential, or scientific, notation</li><li>2 integer.</li><li>3 general format</li><li>4 monetary format</li><li>5 percentage</li></ul> The third parameter indicates the number of figures after the decimal point in the converted string. It should always be specified, although its value is ignored for integer, general and monetary formats. |
| <b>SQR</b><br>( n.exp )                    | Returns the square root of the argument, which must not be negative.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>TIME()</b>                              | Returns, as text, the time of day in the format HH:MM:SS. You must first have set the system clock, as described in the technical manual.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>UPPER</b><br>( s.exp )                  | Converts the specified string to upper case.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>VAL( s.exp )</b>                        | Val converts the text to its equivalent numeric value. It will only convert text composed of valid numeric characters and the conversion will stop at the first character that can not be interpreted as a digit. For example, val("1.1ABC") will return the value 1.1, and val("ABC") will return 0.0                                                                                                                                                                                                                                                                      |

A procedure is a named section of program, starting with a procedure declaration of the form:

```
proc pnm
or
proc pnm ; var (, var)
```

and ending with

```
endproc
```

It may be referred to by name from any other program or procedure, including itself. It acts as though its code had been inserted at the point from which it is called.

In ARCHIVE, the proc and endproc commands can not be used directly from the keyboard, but are added automatically when you use the procedure editor to create a procedure.

The line editor is always available to modify the contents of the input line.

| Key(s)               | Action                            |
|----------------------|-----------------------------------|
| Left cursor          | Move one character to the left    |
| Right cursor         | Move one character to the right   |
| Up cursor            | Move one word to the left         |
| Down cursor          | Move one word to the right        |
| CTRL + Left cursor   | Delete one character to the left  |
| CTRL + Right cursor  | Delete one character to the right |
| CTRL + Up cursor     | Delete all text to the left       |
| CTRL + Down cursor   | Delete all text to the right      |
| SHIFT + Left cursor  | Move left by one word             |
| SHIFT + Right cursor | Move right by one word            |

The program editor is entered by means of the Edit command.

If there are no procedures present in memory you will immediately be offered the option of creating a new procedure, as described later.

Otherwise you are given a list of all the procedures in memory at the left hand side of the display area. The first procedure is highlighted, and is listed in full on the right

## 13.11 PROCEDURES

## 13.12 THE LINE EDITOR

## 13.13 THE PROGRAM EDITOR

of the display. The first line of the procedure is highlighted. This highlighting marks the current procedure and the current line of the procedure.

You then have five options which are to:

- Select a procedure
- Select a line in the current procedure
- Call an editing command
- Insert text in the current procedure
- Edit a line of text in the current procedure

### 13.13.1 Select Procedure

Press **TABULATE** to move down the list of procedures. Press **SHIFT** and **TABULATE** to move up the list.

The listing on the right of the screen always shows the current procedure.

### 13.13.2 Select Line

Press the down cursor key to move to a lower line and the up cursor key to move to an earlier line in the current procedure.

### 13.13.3 The Editing Commands

Press **F3** for the menu of editing commands. There are four commands, selected by pressing the key corresponding to the first letter.

- Delete procedure - delete the current procedure.  
Press **ENTER** to delete the procedure highlighted on the left of the display.  
Press any other key to leave the command without deleting the procedure.
- New procedure - creates a new procedure  
Type in the name of the new procedure and press **ENTER**. If a procedure of that name already exists you will not create a new procedure but will be offered the opportunity to edit the named procedure.
- Cut - removes text from the current procedure and transfers it to the paste buffer. Use the up or down cursor keys to make the first (or last) line of the region to be removed the current line before calling this command. Then use the up or down cursor keys to mark the region of text to be removed. Press **ENTER** to remove the text into the paste buffer. The new text replaces the old contents of the paste buffer.
- Paste - insert text from the paste buffer, below the current line of the current procedure

### 13.13.4 Insert Text

Press **F4** to select the option to insert one or more new lines of text below the current line of the current procedure. Then type the line of text and press **ENTER**. You can leave this option by pressing **ENTER** without any preceding text.

### 13.13.5 Edit Text

Press **F5** to select the option to edit the current line of the current procedure. The line of text is copied into the input line, with the cursor at the start of the editable text in the line. You can then use the line editor, described in Section 9.2, to modify the text. Press **ENTER** to replace the old line of text and return to the main level of the Edit command.

## 13.14 ERRORS

When ARCHIVE detects an error in a command typed at the keyboard or in a procedure it displays an error number and a short error message. Examples of errors that would be detected are:

- attempting to divide by zero
- if not matched with an **endif**
- supplying a procedure with the wrong number of parameters

If you use the **error** command in your programs ARCHIVE will not report any error that it detects in any procedure marked with **error**. You are left to deal with any such error in any way that you want. You can find which error has occurred by examining the value returned by `errnum()`. This number is the same as the number which ARCHIVE would have given, together with the error message.

The list of error numbers and error messages is not available at the time of writing.

Handwritten text, likely bleed-through from the reverse side of the page. The text is mostly illegible due to fading and bleed-through.





**sinclair**

**QL**

**QL Easel**



EASEL is a business graphics package with a difference. It is so easy to use that you will probably find that you hardly ever need to refer to this manual.

Firstly it is *fully interactive*, which means that you see the results of everything you do immediately. From the moment you start you can just type in a series of numbers and see them displayed as a graph, as you type them in. You never need to worry about building up tables of values; EASEL takes care of that kind of thing for you, and keeps them where they should be - out of sight.

You can add *text* to the graph just as simply as you enter data and, once it is there, you can edit it or move it around (easily, of course!) until you are satisfied with the result.

EASEL has a *pyramidal* structure in that its facilities are organised in a series of levels. The top level, which is immediately available when you start, allows you to do the most commonly-needed operations, such as entering data or text. Beneath the surface are many sophisticated commands, each of which has a whole series of sub-levels. The full power of EASEL becomes apparent as you become more familiar with it and dig more deeply into the pyramid.

Despite this power, EASEL still remains simple to use at all levels. You do not need to remember lots of numbers and commands, since you are guided through each process by a carefully-designed sequence of *prompts* which explain what you can do at each stage. In particular, EASEL has a *design by example* facility which allows you to select or design anything from a single line or bar to a whole graph, simply by choosing from a set of pictures. With this facility you need never be in any doubt as to what the final appearance of your graph will be.

You should now go on to Chapter 2 which explains how to load EASEL and start using it. Don't just read this manual: try everything out as you go along. Please experiment as much as you like - you can not harm the computer in any way. The more things you try out the faster you will discover how simple and powerful EASEL is to use.

PROVISIONS

CHAPTER 1  
ABOUT  
BASE

C

C

C

C

This chapter describes how to load EASEL and how to use the basic options, immediately available when you start. By the end of the chapter you should be able to produce useful graphs and charts, using EASEL's pre-defined displays.

When you switch on the computer it will only respond to commands in SuperBASIC. You will have to load EASEL from its Microdrive cartridge. You will normally do so by inserting the EASEL cartridge in drive 1 (the left hand drive) and then typing:

**LGO MDV1\_EASEL ENTER**

After a few seconds the screen will show the message:

**EASEL - Copyright Psion Ltd 1983  
Press any key to start**

You should then press any key on the keyboard to start EASEL.

Before going any further, try typing in a few numbers - for example, 3, 5, -2, and so on - to see how easy EASEL is to use (don't forget to press **ENTER** after each number). See how your graph is displayed straight away, without your having to do anything else. Notice that the scale of the graph adjusts itself automatically to suit the numbers you type in.

Try pressing a few other keys, and see what effect they have on the display. Don't worry about doing anything wrong - you can't do any harm. When you have realised that there is no way you can cause any damage you will be in a better position to find out what EASEL can do.

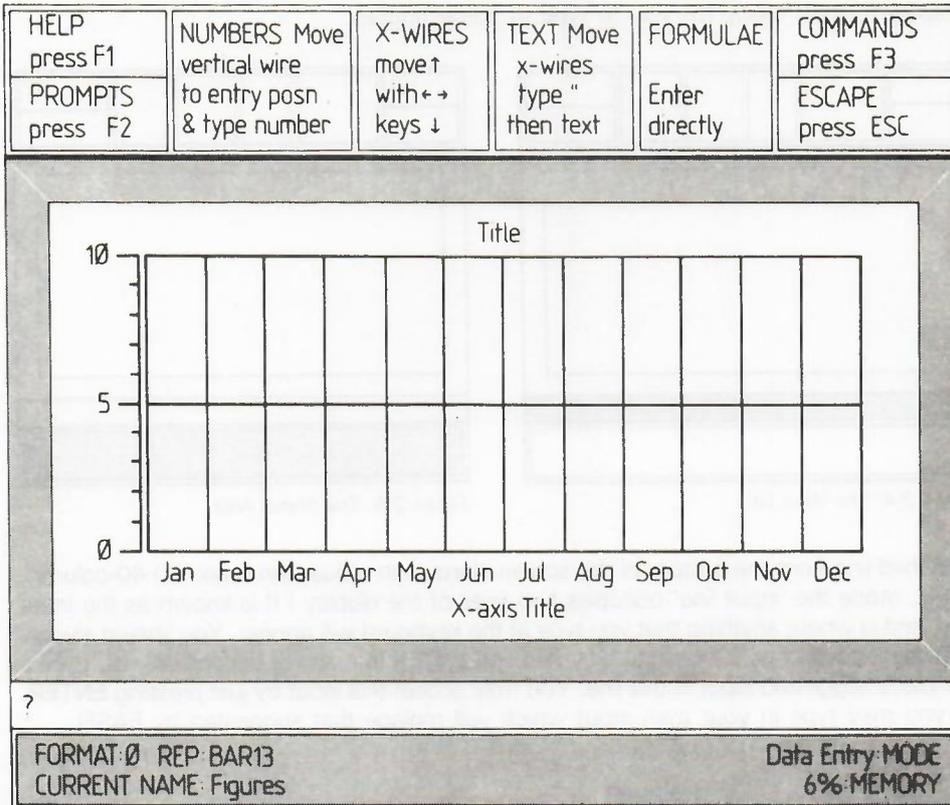


Figure 2.1: The Main Display.

When you have loaded EASEL the display should look like that shown in Figure 2.1. The display is divided into four main areas, known as the control area, the display area, the input line and the status area.

### 2.1 INTRODUCTION

### 2.2 LOADING EASEL

### 2.3 APPEARANCE

## 2.3.1 The Control Area

The control area occupies the top four lines of the display and shows you your options. Its contents will change from time to time, depending on what you are doing, and it also confirms your choices. Initially it shows that you have eight main choices which are to:

- press the Help key,
- turn off the prompts
- type in a number,
- move the cross wires,
- type in text,
- type in a formula,
- use a command,
- press the ESC key.

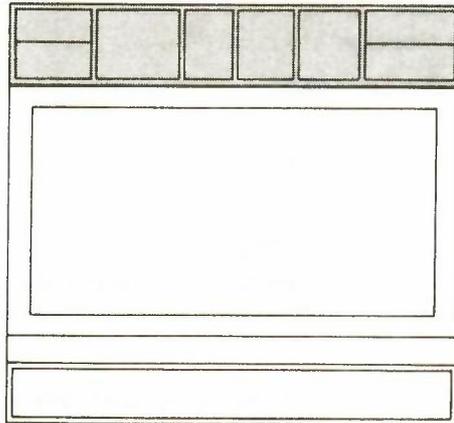


Figure 2.2: The Control Area.

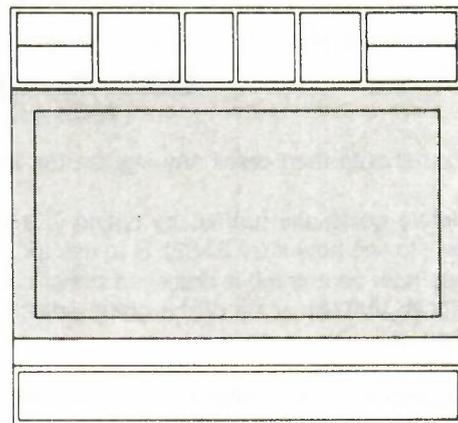


Figure 2.3: The Display Area.

## 2.3.2 The Display Area

The display area is, as its name suggests, where all graphs produced by EASEL are displayed. Before you add any data this area shows an empty grid (dependent on the display format) ready for you to type in some figures.

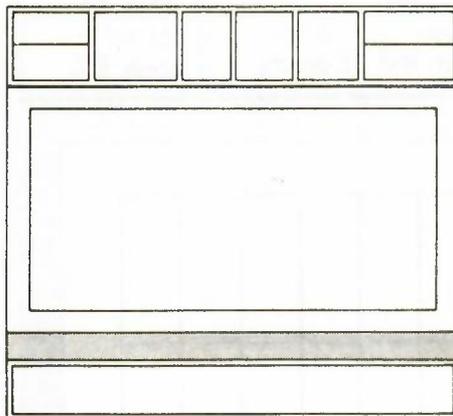


Figure 2.4: The Input Line.

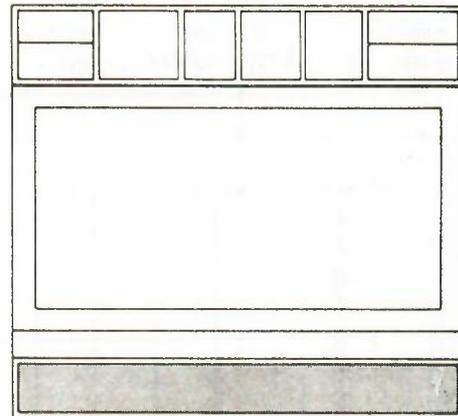


Figure 2.5: The Status Area.

## 2.3.3 The Input Line

The third line from the bottom of the screen starts with a question mark. (In 40-column, ie TV, mode the 'input line' occupies two lines of the display.) It is known as the input line, and is where anything that you type at the keyboard will appear. You should always indicate the end of your typed input by pressing **ENTER**. In many of the commands, EASEL will place suggested input in this line. You may accept this input by just pressing **ENTER**, or you may type in your own input which will replace that suggested by EASEL.

## 2.3.4 The Status Area

The status area uses the bottom two lines of the screen and tell you about the current state of the display. The *format* tells you how the values you type in will be displayed. There are eight different display formats (numbered 0 to 7) to choose from, pre-defined to give an assortment of bar, line and pie chart displays. (In Chapter 6 you will find out how to change them to suit your own purposes.) When you have just loaded EASEL, the format is set to give you a bar graph display (format 0).

# PROVISIONAL

You are also told the name of the *current figures*. This is the set of figures that is changed when you type in numbers. In addition you are told the *style* (as a bar, line or pie number) which will be used to represent them. Don't worry 2-4 if some of this information does not mean very much at first - you will find it useful when you have used EASEL for a short time.

The status area also tells you the current *mode* of operation; this is initially set to *data entry*, ready for you to type in some numbers. The final piece of information in the status area is the amount of memory that is currently being used, as a percentage of the total. The status area is also used to display *error messages*, should you make a mistake. These error messages tell you what has gone wrong and, if necessary, how to put it right.

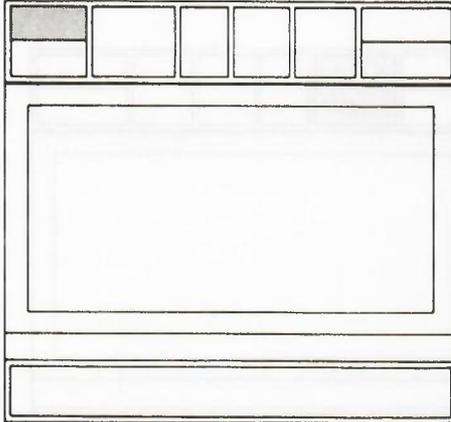


Figure 2.6: Help.

The first option, displayed at the top left of the control area, shows that you may ask for Help by pressing function key (F1). Regardless of any other changes in the control area display, the Help option will always be shown (unless you have switched off the prompts - see Section 2.5). This indicates that the Help facility is always available, no matter what you are doing.

*Try pressing the Help key now.* When you do, the current display will disappear, to be replaced by one giving brief details of your options. At the bottom of this display there is a list of topics. You can ask for further information about any one of these topics by typing in its name and pressing **ENTER**. You do not need to type the whole of a name; you need only to type in the first few letters - enough to distinguish it from any of the other names in the list.

When, after typing in one of these names, you press **ENTER** you will find that further, more detailed, information is shown about the command you have selected, and another list of sub-topics will be shown. You may then select one of these sub-topics by typing in the first few letters of its name, as described before. You may continue this process until no further information is available.

At any stage you may return to the previous screen by simply pressing **ENTER**. Repeatedly pressing **ENTER** will eventually take you back to the main display, with the control, display and work areas. At this point you will have left the Help facility and will have been returned to the exact state before you pressed the Help key. A faster way to return from Help is to press **ESC**. This will return you from any point with Help, back to the state from which it was first called.

*Try using the Help facility to examine some of the pathways through the information.* Don't worry if you do not understand some of the information that is shown - it will make sense when you actually need to use it. All you need to do at the moment is to become familiar with the way in which the Help facility is used. When you have finished, press **ESC** to return to the main display.

It must be emphasized that Help is always available, at any time. Whenever you are not sure what you should be doing, just press the Help key even if you are, for example, in the middle of typing in numbers or text as part of a command. You will not always start at the same point in Help, but will be presented with the information most relevant to what you were doing when you pressed the Help key.

When you have found the information you need and leave Help (by use of the **ESC** or **ENTER** keys) you will always be returned to the exact point from which you started, as though there had been no interruption. Use the Help key as often as you like - it is there to assist you and will usually be the quickest and simplest way of solving your problems.

## 2.4 HELP

## 2.5 THE PROMPTS

In addition to showing your options, the control area highlights your choice and, when necessary, suggests what you should do. These aids to using EASEL are known as prompt messages or just *prompts*.

*Try pressing function key 2 (F2).* The display will be redrawn without the control area, leaving more room for your graph. You can bring back the control area by pressing F2 again. EASEL works in exactly the same way, whether the prompts in the control area are displayed or not - you are free to choose either option. You will probably find it most useful to build up your graphs with the prompts in the control area visible, and then turn them off to look at the finished result.

You can restore the control area display at any time by pressing F2 again.

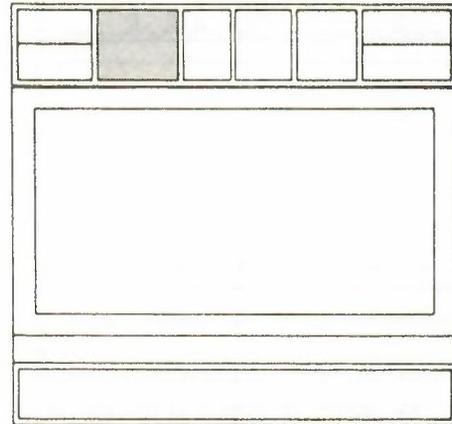


Figure 2.7: Prompts.

## 2.6 THE CROSSWIRES

You can indicate any point in the display area by using the crosswires, shown as a vertical and a horizontal line superimposed over the display. Initially, when you are in data entry mode, the vertical crosswire is visible, positioned on the cell (a position in the graph or chart where a value can be displayed) which will show the next number you type in. As you type in numbers the crosswire moves from cell to cell automatically. The **TABULATE** key moves the vertical crosswire from one cell to the next on the right, for data entry (pressing **SHIFT** and **TABULATE** moves to the cell on the left). You can also move the vertical crosswire to any particular cell by pressing the left or the right cursor key until it is where you want it. These keys move the crosswire smoothly across the screen, rather than in cell-sized steps.

If the crosswires are not visible you can display them by pressing the cursor keys; press either the left or the right cursor key to display the vertical crosswire, and either the up or down cursor key to show the horizontal crosswire. Once both crosswires are visible you can move them around the display area by means of the cursor keys. If you press one of these keys and release it immediately the crosswire will move a short distance in the appropriate direction, but if you hold the key down the crosswire will move more rapidly across the display area. Note that the crosswires move freely, to any point in the display area, in contrast to the behaviour of the vertical crosswire in the data entry mode.

*Try using the cursor keys to move the crosswires around the screen.*

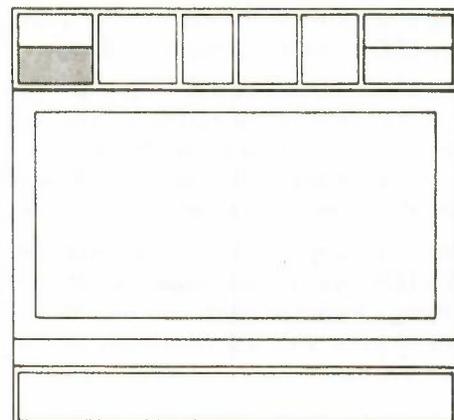


Figure 2.8: The Crosswires.

# PROVISIONAL

If you type in a number (and then press **ENTER**) it will be displayed immediately on the graph, at the current position of the vertical crosswire. Move the vertical crosswire to a point near the centre of the screen, type in a number and press **ENTER**. You will see the value displayed on the graph immediately and the vertical crosswire will move one cell to the right, ready for the next number to be entered. If you now try pressing either **TABULATE**, or **SHIFT** and **TABULATE** together, you will find that each press of the key makes the vertical crosswire move left or right by one cell.

## 2.7 NUMBERS

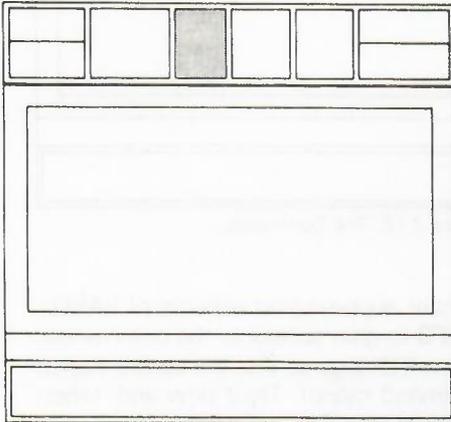


Figure 2.9: Entering Numbers.

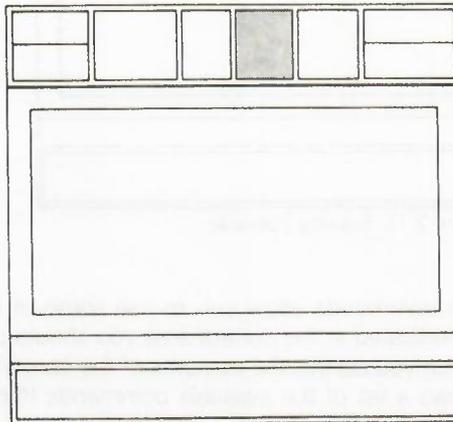


Figure 2.10: Entering Text.

When you have displayed a few values on the graph you can try adding some text. You can do this by typing a double or single quotation mark ( " ' ) as the first character to the input line.

## 2.8 TEXT

The crosswires will appear (if they were not already visible) and any following text that you type in will appear in the display area starting at the intersection of the crosswires. Don't worry if the text is not in the exact position that you want; it is very easy to move the text to another position. When you have typed in your text, press **ENTER**. You can now move your text to the exact position you want using the cursor keys. The crosswires will move across the screen, carrying the text with them. When the text is in the position you want, you should press **ENTER** to both drop the text and switch off the crosswire display.

*At this stage you might like to try adding a few messages at a number of different places in the display to see how it works.*

Remember that the text which labels the columns of the graph is treated specially and can only be altered using the Labels option of the Edit command.

You will normally use a formula to produce a new set of data from existing sets, and this will be described in Chapter 3. As a simple illustration of the use of formula, however, we can change the current set of the data (which, as you can see from the status area, has the name "figures"). EASEL interprets any keyboard input that does not start with a numeric digit or quotation marks as a formulae. An example of the formulae would be;

## 2.9 FORMULAE

**figures = figures \* 2**

try typing this in to see what effect it has. As you can see the new graph that is displayed looks just the same as the old one except that the vertical scale has been doubled. If you want to return to the original scale of the graph you can type in a further formula:

**figures = figures / 2**

A formula always starts with the name of a set of figures. This name could be, as in the previous two examples, the name of an existing set or it could be a new name. In either case the contents of that data set is defined by the expression to the right of the equals sign in the formula. It is important to realise that the formula will affect all the values in the set, rather than just one value.

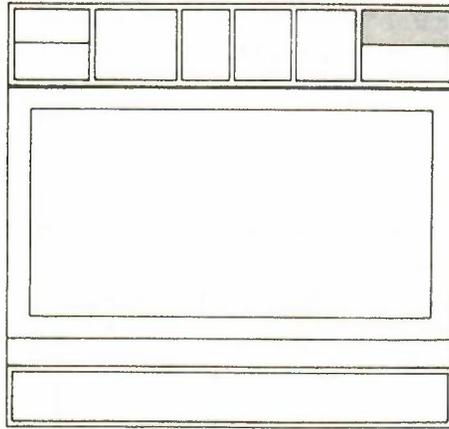


Figure 2.11: Entering Formulae.

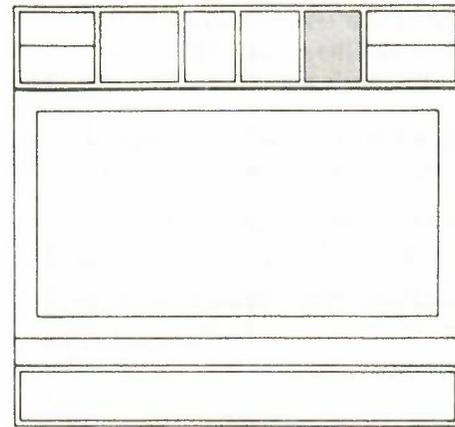


Figure 2.12: The Commands.

## 2.10 THE COMMANDS

The commands allow you to use some of the more sophisticated aspects of EASEL. As indicated in the control area you should press **F3** to gain access to the commands. When you do this the contents of the control area will change so that the central region shows a list of the available commands (the command menu). Try it now and, when you have the display of the command menu, as illustrated in Figure 2.13, press the Help key (**F1**).

The display will be replaced by a brief description of the commands and a list of topics for further information, as described in Section 2.4. You can follow this list to find out in more detail what each command does, pressing **ENTER** to go back one level, or pressing **ESC** to go back to the command menu.

When the command menu is displayed you can select any one command simply by typing its first letter on the keyboard. As an example you can activate the Quit command by pressing the Q key. This command causes you to leave EASEL and return to SuperBASIC, with an option to stay in EASEL (in case you called the command by mistake). If you decide you really do want to leave EASEL, you will be first asked if you want to save any files.

The remaining commands are described in later chapters.

|                     |                       |                |                |                  |                      |
|---------------------|-----------------------|----------------|----------------|------------------|----------------------|
| HELP<br>press F1    | COMMANDS<br>Highlight | Change<br>Load | Delete<br>Move | Files<br>Newdata | COMMANDS<br>press F3 |
| PROMPTS<br>press F2 | Olddata               | Print          | Quit           | Rename           | ESCAPE<br>press ESC  |
|                     | Same                  | View           |                |                  |                      |

Title

X-axis Title

Command?

FORMAT: REP: BAR1  
CURRENT NAME: Sales
View Data: MODE  
23%: MEMORY

Figure 2.13: The Command Menu.

# PROVISIONAL

You can generally use the **ESC** key to cancel the current action, or to leave a particular sequence of operations.

We have already seen how you can use the **ESC** key to leave Help, from any level (Section 2.4) and to leave the command menu (Section 2.10).

You can also use **ESC** to cancel input to the input line. Try typing in a number and, rather than pressing **ENTER**, press **ESC**. The entry is cancelled, as you would expect.

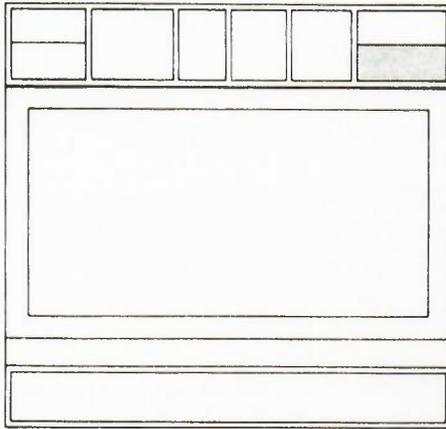


Figure 2.14: Escape

This key is used to delete numbers, text, axis labels and the key box (when you have more than one set of figures - see Chapter 3). You will find descriptions of how to delete text and the key box in Chapter 4.

You can delete a number from a graph by positioning the vertical crosswire on the cell containing the number to be deleted and pressing **F4**. The crosswires will move to the next cell to the right automatically, so that it is easy to delete several numbers with a small number of keypresses.

You can insert a new cell after the one marked by the vertical crosswire (in the case of line graphs and bar charts) or the highlighted box (for pie charts) by pressing **F5**. A gap is opened up, ready for you to type in the new number. The new value will not have a label, but you can add one at a later stage.

A further feature, available at all times that you are typing characters at the keyboard, is a full line editor. With its aid you can modify any or all of the editable text in the input line.

The editable text excludes, for example, the **command** prompt in the input line when you are using a command. In general, any text that appears as the result of pressing a single key can not be edited since it has already been interpreted and acted upon. You can edit any text that you have typed in full, before you press **ENTER** to pass the text to EASEL.

At all times each character that you type will be inserted to the left of the input line cursor position, and the cursor will move one space to the right. Regardless of the position of the cursor, all the text in the input line is accepted as input when you press **ENTER**.

The line editor uses the four cursor keys, together with the **CTRL** and **SHIFT** keys.

The left and right cursor keys, used on their own, move the input line cursor by one character to the left or right.

If you press **SHIFT** and, while holding it down, press the left or right cursor key the input line cursor moves left or right by units of a word, that is to the next space or comma.

If you press **CTRL** and, while holding it down, press the left cursor key you will delete the character to the left of the cursor. Pressing **CTRL** together with the right cursor key deletes the character under the cursor. The following text closes up to fill the gap.

If you press the up cursor key the cursor moves to the beginning of the editable text in the input line; the down cursor key moves the cursor to the end of the text.

## 2.11 ESCAPE

## 2.12 FUNCTION KEY 4

## 2.13 INSERTING A CELL

## 2.14 THE LINE EDITOR

### Left and Right Cursor Keys

### Up and Down Cursor Keys

Holding down the **CTRL** key and pressing the up cursor key will delete all editable text to the left of the cursor. Pressing **CTRL** and the down cursor key deletes all text to the right, including the character under the cursor.

## CHAPTER 3 MULTIPLE DATA SETS

### 3.1 INTRODUCTION

So far we have only described how to create and display a single set of figures. On many occasions you may want to display two or more sets of data on the same graph, for example to compare the sales figures for two successive years. This chapter describes the techniques you can use to produce, modify and display graphs containing several sets of figures.

Although you may have defined several sets of figures, you can only modify one set at a time. The set that you can add to or change is known as the *current figures*, and its name is shown in the status area. Its display is not necessarily shown on the screen. It will, however, always be displayed when you are actually making any changes.

### 3.2 THE CURRENT FIGURES

There are two methods that you can use to produce second and subsequent sets of figures and these are by using the Newdata command, or by using a formula. These two methods are described in this and the following section. Suppose you have created a set of figures called "sales" containing monthly sales figures and now want to include a display of the monthly costs. You can do this by pressing **F3** (to get the command menu and then pressing the N key which calls the Newdata command. You are then asked for the name to be given to the new figures and should, for example, type in

### 3.3 THE NEWDATA COMMAND

**costs ENTER** (no quotation marks are needed)

you are immediately given a new, blank graph (assuming you are in a bar or a line format) with the vertical crosswire set on the first column, ready for typing in the new set of numbers. The status area shows that the current figures are the new set, with name "costs". All you have to do is type in the new numbers which are immediately displayed on the graph as normal.

If you want to create a third set of figures, you can use the Newdata command again, exactly as has been described, giving each set of figures a different name. You can create as many sets as you like, the only limit is the amount of computer memory that you have available.

The Newdata command is useful for defining completely new sets of figures, but on many occasions you may want to produce a new set that is related in some way to one or more existing sets. You may, for example, have already entered sets of figures for sales and costs, and then want to include a set of figures which represent the resulting profits. Rather than calculate the profit figures yourself and then enter them by means of the Newdata command, you can use a formula and let EASEL do the work for you. All you have to do is type in the formula which describes the new set of figures that you want, for example

### 3.4 USING A FORMULA

**profits = sales - costs**

This will create a new set of figures with the name "profits" and each value being the difference between the corresponding values of the sales and the costs figures. The "profits" graph will be displayed immediately, and becomes the current figures set. You could easily produce a result like that of Figure 3.1.

When you use a formula in this way you will normally have an equal number of values in each of the sets of figures referred to in the formula. This, however, is not essential; EASEL will calculate and display all possible values, even if the sets of figures contain different numbers of values. You can also use a formula without having to refer to existing sets of figures. You could, for example, write a formula such as

**wave = 10 \* sin(count/2)**

This formula creates and displays a new set of figures with the name "wave", whose values are calculated using the sin() function. In this formula we have also used "count". This provides a value which is the number of the cell of the graph, counting from the left hand side. The left-most column gives a value of 1, the next column is a value of 2, and so on. To see how this works, type in the formula:

**a = count**

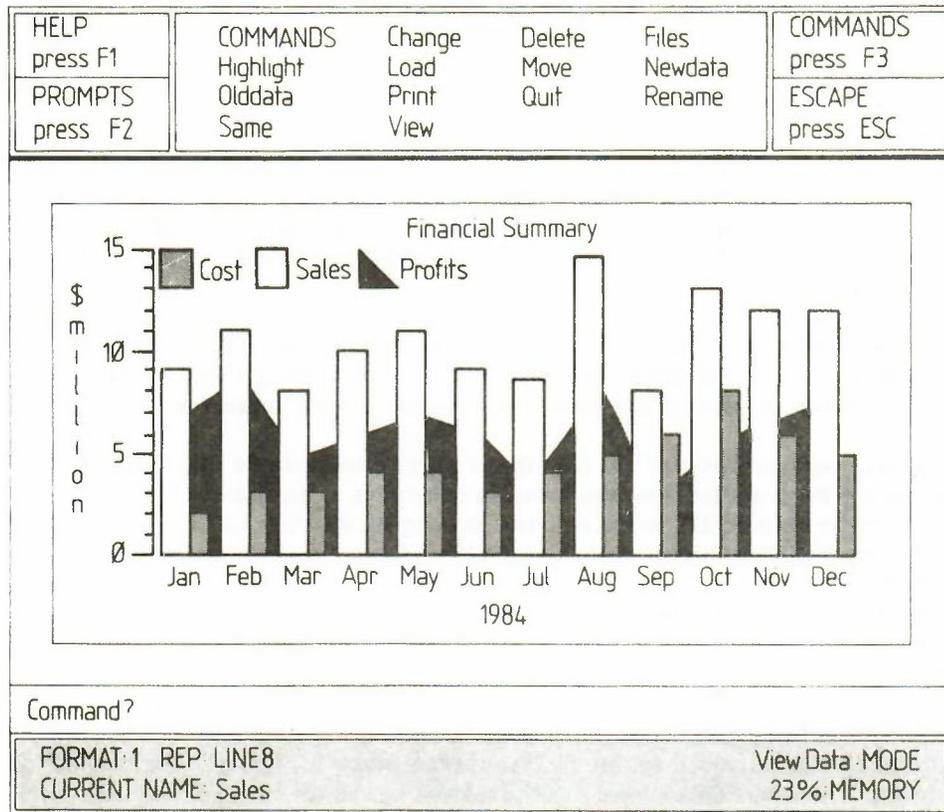


Figure 3.1: Profit Calculation

and look at the graph that is drawn. When you use count in a formula, the number of values in the set of figures is made equal to the number of columns currently being displayed on the graph.

### 3.5 THE OLDDATA COMMAND

When you use the Newdata command the sets of figures that you create becomes the current set. Remember that this is the set that can be added to or changed by typing in numbers. If you want to make some changes to an existing set of figures that is not the current set, you can do so by using the Olddata command. When you call this command you are asked to type in the name of a set of figures. In this case the name you type in must be the name of an existing set, and that set becomes the current figures.

Suppose that you have two sets of figures called "costs" and "sales", and that "sales" is the current set of figures. If you want to change or add to the "costs" figures you should select it by the Olddata command. The costs figures will then be displayed on the graph and you can modify the data by typing in numbers, as described in Chapter 2.

### 3.6 VIEWING THE DATA

You can see the effect of displaying all of your figures on a single graph by means of the View command. Try calling this command (by pressing F4 and then the V key). As you see, EASEL suggests that all the sets of figures should be displayed on the graph and you can accept this suggestion by pressing ENTER. EASEL then suggests the display format to be used and again you can accept this suggestion by pressing ENTER. A graph is displayed immediately, containing all the data that you have defined together with a key box which shows the name of each set of figures and the way that it is displayed (the key is not shown if you only have one set of figures on the graph). You can move the key if it is not in a convenient position and this is described in Chapter 4.

If you have defined a large number of sets of figures the graph will be very crowded and have very little impact. In general it is a good idea to display only a small number of sets of figures on any one graph to make the best visual impact. This does not mean that you should only define a small number of sets of figures, since the View command allows you to select which sets of figures that you want displayed. A way that you can do this is by not accepting the "all figures" suggestion given in this command. Instead of just pressing return at this point, you can type in a list of the names of those sets of figures which you want to be displayed, separating the items in the list by commas. When you have typed in all the names of the sets of figures that you want to be displayed you should then press ENTER.

# PROVISIONAL

You may also select a different format for the display instead of accepting the suggestion made by EASEL. Instead of just pressing **ENTER** to accept the suggested format you can type in a number between 0 and 7. EASEL is provided with eight pre-defined formats, numbered 0 to 7, providing various styles of bar charts, lines or pie diagrams. Try using the View command to display three or four sets of figures in a number of the different formats available. Chapter 6 will describe how you can design your own graph format, or style of bar, line or pie chart. Figure 3.2 is an example of what you can produce.

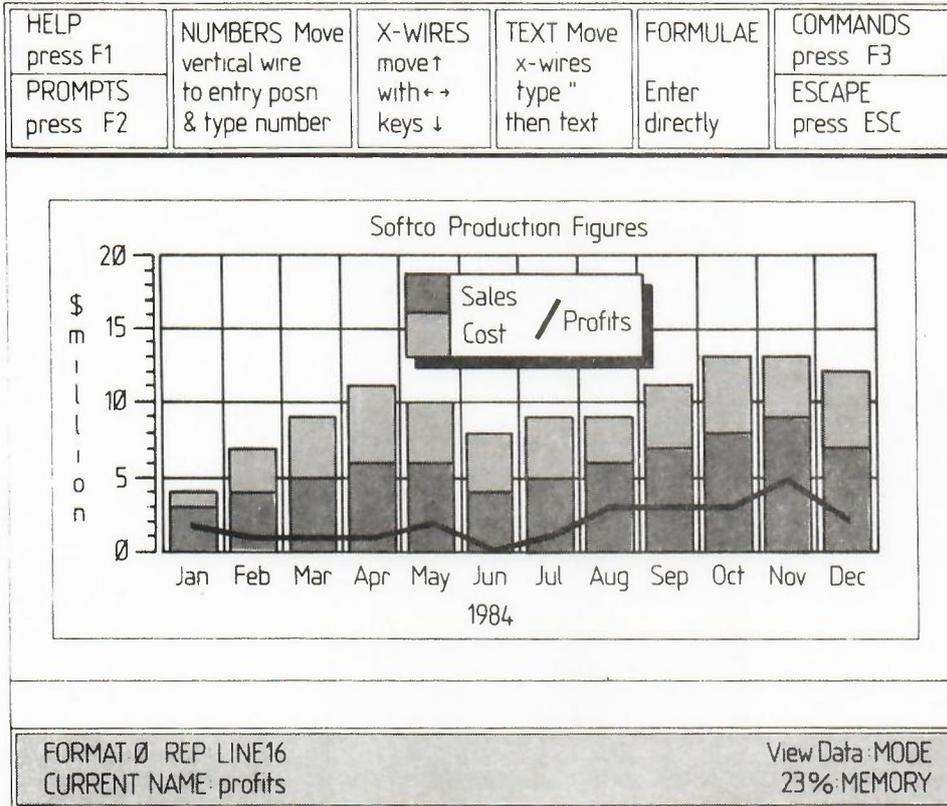


Figure 3.2: Lines And Stacked Bars



10/1/19

10/1/19

## CHAPTER 4 USING TEXT

### 4.1 INTRODUCTION

Section 2.8 describes how you can add text at any point in the display by typing in the text, starting with quotation marks, and then moving it with the aid of the cursor keys. In this chapter we shall describe how you can edit existing text, or move it to another point in the display. In addition we shall describe the changes you can make to the axis labels, the graph title and the key.

If you want to make changes to a piece of existing text you can use the Edit command, and then select the Text option by pressing the T key. As indicated in the control area, you should then use the cursor keys to move the intersection of the crosswires close to the text which you want to change. It is not necessary to position the crosswires exactly on the start of the text. When you have positioned the crosswires you should press any key and the crosswires will attach themselves to the nearest text. A copy of the text will also appear in the input line.

You then have two choices: you may delete the text by pressing F4, or modify the text with the aid of the line editor (see section 2.14). If you choose to delete the text this will also end the command.

When you are completely satisfied with the wording of the text you should press **ENTER**. You are then given the opportunity to reposition the text using the cursor keys. Press **ENTER** when you are satisfied with the position.

A further text option is to change the colour in which text is printed. You do this by means of the Change command which is described in Chapter 6. Any new text is displayed in the colour set by the last change of text colour. This includes any text that you edit after a change of colour. A convenient way of changing the colour of an existing piece of text is first to change the text colour and then to use the Edit command on the existing text, without actually changing its position.

All graph displays in EASEL can use three titles; a general title, an x-axis (horizontal axis) and a y-axis (vertical axis) title. When you load EASEL these titles are shown with the text "TITLE", "X-axis Title" and "Y-axis Title" respectively.

EASEL treats a graph title in a similar way to any other text. There are a few slight differences; the initial colour of the title is not necessarily the same as for other text (it has a separate default colour set by the display format) and there is a special "edit title" prompt which appears in the input line when you attach the crosswires to a title. Other than this, you can write a title, modify or move it exactly as described in the previous section. You can position the title anywhere in the display area, including the surrounding border.

The cells of the graph are provided with labels which are initially set to show the months from January to December. EASEL treats these labels specially, and you have to use the Labels option of the Edit command to change them. When you do so the crosswires will attach themselves to the nearest label. This label will then be displayed in full (normally only the first few characters are shown, depending on the width of the cells) and the text is also copied into the input line. You can then delete the label by using F4, or edit it as described in Section 4.2. Press **ENTER** to finish editing the label.

You cannot move an axis label; if, for example, you attach the crosswires to an x-axis label and press either **TABULATE** or **SHIFT** and **TABULATE** together the label will not move.

One of the options in the Edit command is to move the key. If you select this option the crosswires will attach themselves to the key immediately and you are then offered the option of either deleting the key by pressing F4, or moving the key by means of the cursor keys. If you choose the move option the cross wires will pull a box equal in size to the key around the display area. When you finally press **ENTER** the graph will be redrawn with the key in its new position.

You may at some time want to restore the display of a key which you had deleted earlier. You can do this by using the Edit command and selecting the Key option. The crosswires will attach themselves to the position of the (invisible) key and you may, if you want, move the key to a new location. Whether you move the key or not, pressing **ENTER** will cause the graph to be redrawn, including a display of the key.

### 4.2 EDITING AND MOVING TEXT

### 4.3 GRAPH TITLES

### 4.4 THE AXIS LABELS

### 4.5 THE KEY

The only change that you can make to the contents of the key box is to change the colour of the text that it includes. This text is always drawn in the colour last set by using the Change command (see Chapter 6). The symbols shown in the key box will, of course, always match the symbols which you use to display the graphs.

## CHAPTER 5 LINE GRAPHS AND PIE CHARTS

### 5.1 INTRODUCTION

Apart from a brief mention of the line graph and pie chart formats in Chapter 3, we have so far described everything in terms of bar charts. As you may have seen when you experimented with the different display formats, the sets of figures can also be represented by line graphs or by pie charts. This allows you to display a given set of the figures in many different ways so that you can choose the method most appropriate for your needs. EASEL has been designed so that the values themselves are kept completely separate from the way in which they are displayed, making it very simple for you to make such changes.

Several of the pre-defined formats provided with EASEL use lines to display some of the sets of figures. Each value may be marked by a symbol and the values are joined by lines of various thicknesses and colours. You can also use "filled" lines where the space between the line and the zero level is completely filled with colour. You may find this form of display useful for showing "critical values", such as a break even level, as a background to your graph. Since bars and lines are both displayed on the same type of grid, you can mix bars and lines in any combination. Titles, axis labels, general text and the key box all behave in exactly the same way for both bars and lines.

If you select a format which uses a line graph for your current figures, you can enter your data in exactly the same way as described for bar charts.

As you type in the numbers the graph is drawn using a thin white line, or a filled line, depending on the format you have chosen. You are completely free to move the vertical crosswire backwards and forwards to change the value in any cell, and the lines will be redrawn whenever you make a change.

When you have finished entering your values you can use the View command to see your graph with your chosen colour and style of line.

Although a pie chart is very different in appearance from a bar chart or a line graph, EASEL allows you to create them in exactly the same way.

### 5.2 LINE GRAPHS

### 5.3 DIRECT ENTRY FOR LINES

### 5.4 PIE CHARTS

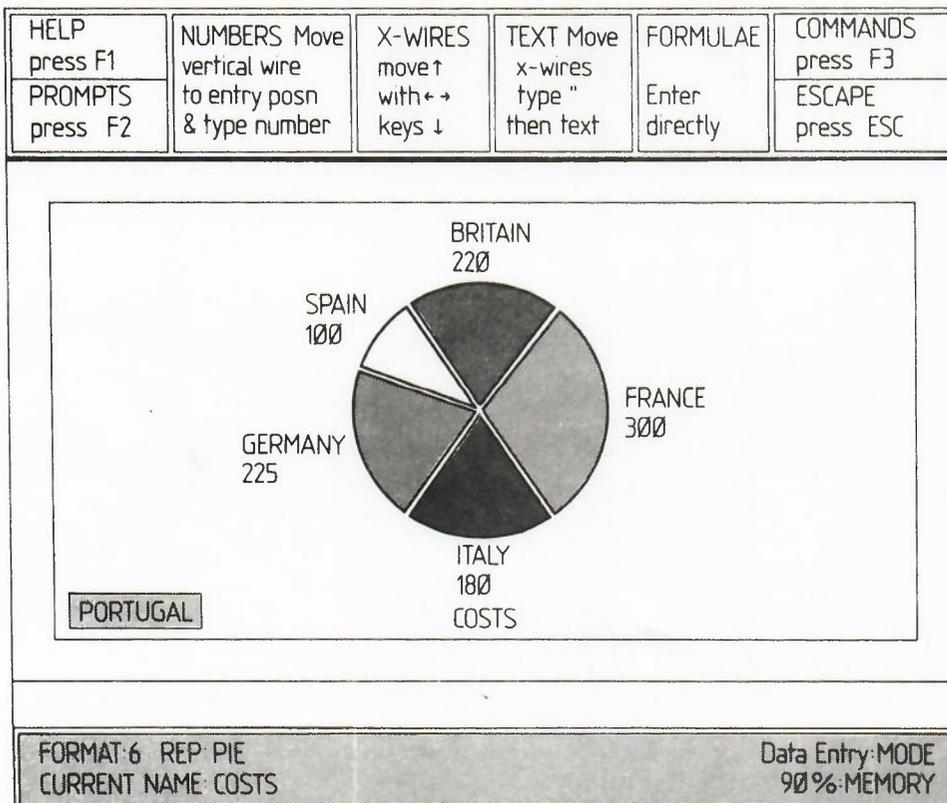


Figure 5.1: A Pie Chart.

To illustrate entry into a pie chart, use the Newdata command to create a new set of figures called, for example, "costs". Now use the Change command to display "costs" (as yet there is nothing to see) with format 7, which is a pie chart format. Now try typing in a few numbers, just as if you were entering values into a bar chart. Figure 5.1 shows a pie chart with some numbers entered.

The first number you type in will produce a complete circle of colour in the display area. This is simply the display of a pie chart which contains only one value. Now use the **TABULATE** key to move the the box cursor to another label. When you type in a second number the circle will be divided into two sections, showing the relative proportions of the two values. Note how each section is labelled, together with its value.

During data entry into a pie chart the next cell to receive data can not be shown by a crosswire position, as in bar charts and line graphs. If the cell is being displayed on the pie chart, its label will be highlighted, otherwise it is indicated by a special highlighted display box at the bottom left of the display area. In Figure 5.1 this is the label "PORTUGAL".

Pie charts are treated exactly as bar charts and line graphs. You can add, delete, and move text and titles exactly as described for lines and bars in the earlier chapters.

## CHAPTER 6 DESIGNING A DISPLAY

### 6.1 INTRODUCTION

By now you will be able to create graphic displays of your sets of figures, using bar charts, line graphs and pie diagrams. The appearance of these graphs has so far been limited to use only the pre-defined types of display provided with EASEL. In this chapter you will learn how to modify the appearance of a display to match your own requirements.

You have great flexibility in changing the display, ranging from changing the appearance of a single value to make it stand out in the display, to designing a completely new display format with a new border, graph paper, axis markings and type of representation for all the figures in the graph.

### 6.2 SELECTING A REPRESENTATION

There are two main routes to redesigning the display; design by example, and selection by style number.

In many cases you may not know the number of the bar that you want to use or you might want to try out the effect of using several different formats to find the one you like best. In such a case you can take advantage of the ability to create a design by example. This is by far the easiest way of changing the design of any feature.

#### 6.2.1 Design by Example

Wherever you have the option to choose a numbered style feature you can, instead of typing in a number, just press **ENTER**. Try this method of selecting a bar by typing in

#### F3 Change to BAR? ENTER

The display changes to show examples of all the available bar styles, together with their associated numbers. This is shown in Figure 6.1. The first bar in the display is surrounded by a box which indicates that this is the bar that will be selected for use if you press **ENTER**. If you select a different bar you should use the left and right cursor keys to move this box from bar to bar until it is positioned on the one you want. When you press **ENTER** the bar you have chosen will be used in the display of the current set of figures.

|                     |                                                                                                                                             |                      |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| HELP<br>press F1    | CHOOSE OBJECT Make the section<br>by moving the box cursor with the<br>cursor keys and press RETURN. For a<br>new design select the last(?) | COMMANDS<br>press F3 |
| PROMPTS<br>press F2 |                                                                                                                                             | ESCAPE<br>press ESC  |

|                   |    |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------------------|----|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| Default Bars      |    |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 0                 | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| User-Defined Bars |    |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 16                | 17 | ? |   |   |   |   |   |   |   |    |    |    |    |    |    |

|                       |                 |
|-----------------------|-----------------|
| FORMAT: 0 REP: BAR13  | View Data: MODE |
| CURRENT NAME: Figures | 11%:MEMORY      |

Figure 6.1: Selecting A Bar.

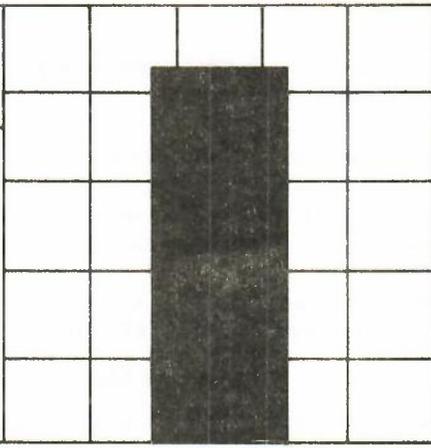
# PROVISIONAL

You will notice that there is one bar present which shows a question mark in place of its number. You can select this bar if you want to make your own design, rather than using one of the pre-defined styles.

To see how this works, position the selection box on this bar and then press **ENTER**. The design by example continues by presenting you with a blank bar design and a list of options, illustrated in Figure 6.2.

|                     |                                                                                                                                                                                   |                      |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| HELP<br>press F1    | DESIGN Use $\uparrow$ keys to select design option and press return. Use $\leftarrow$ $\rightarrow$ keys to select colours then press return.<br>Continued change until satisfied | COMMANDS<br>press F3 |
| PROMPTS<br>press F2 |                                                                                                                                                                                   | ESCAPE<br>press ESC  |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; width: 30px; height: 30px; background-color: white;"></div> <div style="border: 1px solid black; width: 30px; height: 30px; background-color: gray;"></div> <div style="border: 1px solid black; width: 30px; height: 30px; background-color: black;"></div> </div> <p>Choose bar fill colour</p> <p>Choose bar border colour</p> <p>Choose border thickness</p> <p>Satisfied with this design</p> |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

|                             |                |
|-----------------------------|----------------|
| FORMAT $\emptyset$ REP BARS | View Data MODE |
| CURRENT NAME: Figures       | 11% MEMORY     |

Figure 6.2: Designing A Bar.

The first option, which is highlighted, is to choose the bar colour from the palette shown across the top of the display. You have the choice of either moving to another option, by means of the up and down cursor keys, or of selecting the currently highlighted option by pressing **ENTER**. If you accept the bar fill option a box is drawn around the first colour in the palette and the specimen bar is filled with that colour. You can move from colour to colour by pressing the left or right cursor keys and make your selection by pressing **ENTER** when the bar is filled with the colour you want.

The next option in the list, to select a border colour for the bar is then highlighted automatically. Again you can either select this option by pressing **ENTER**, or move on to one of the other options, with the up and down cursor keys. If you select this option you can choose the border colour for the bar in the same way as you chose the main fill colour. Again you should press return when you are satisfied with the result.

The third option is to select the width of the border and, as with all the options, you can select it by pressing **ENTER** when the option is highlighted. In this case you are asked to type in a number to represent the width of the border as a percentage of half the width of the bar.

You are finally given the option of deciding whether you are satisfied with the design as shown. If you are you should press **ENTER**, when the new design will be added to the list of bar designs and it will automatically be used for display of the current set of figures. If you are not satisfied with the design you can go back to one of the other options, using the up and down cursor keys, and try a new combination. At any time before you accept the design you can terminate the command by pressing **ESC**. This will cause you to leave the command without creating a new bar design.

# PROVISIONAL

If you select the Line option of the Change command you will be offered the same design by example facility, where you can select one of a number of pre-defined lines or design a new one to your own specifications. In this case you are able to select the line colour, thickness, and whether or not the line should be filled with colour down to the zero level. In addition you are offered the option of choosing the shape and colour of the symbol to be attached to the line.

The second method, which is useful when you are constantly using the same form of display for many sets of figures, is to specify the format and the style of bar and so on by number. EASEL is provided with 8 main display formats (numbered 0 to 7) and you can use this number to specify which format should be used each time you use the View command. In addition to using different styles of border, background and bar colour, these formats give you a range of display styles.

In addition, each style of bar, line and pie chart has a number which is shown in the status area when you are displaying a single set of figures. This makes the choice of style very simple, provided you know the number of the style that you want. Suppose, for example, that you are displaying a single set of figures with bar style 13 and you know that you want to use bar style 8. You can make this modification very simply by means of the Change command, by typing in the following:

### F3 Change to BAR8 RETURN

*Remember that you only need to type in the first character of each word and EASEL fills in the rest of the command for you.*

You can redesign the entire appearance of any or all of the eight different formats provided with EASEL. Changing a format is one of the options in the Change command, so you would start by typing:

### F3 Change to Format ? ENTER

As with all the Change options, you can choose your design either by typing in the style number of each feature, or use the design by example facility. When EASEL asks you for the format number you can type in a number between 0 and 7, or just press **ENTER** to select design by example, which is much more fun.

Assuming that you choose design by example, EASEL displays all eight formats on the screen, together with some specimen data so that you can see the exact appearance of each. You can select any one of the formats by typing in its number pressing **ENTER**.

You would normally select the one that most closely resembles the appearance you want, and then modify it with some of the other options of the Change command.

Once you have selected the format you can then modify any of the features described below.

You can choose the background colour and the style of graph paper markings.

You can choose from several styles of display for the scales marking the axes of bar charts and line graphs.

## 6.2.2 Selection by Style Number

## 6.3 DESIGNING A FORMAT

### 6.3.1 Graph Paper

### 6.3.2 Axis



When you have created a graph you may want to save it for later use or modification. You can do this by means of the Save command. This command saves all the data for your graph as a named file on a Microdrive cartridge.

You will be asked to type in a name for the file. Note that this will be the name for the whole graph, so do not confuse it with the name of a set of figures. You may type in a question mark (followed by **ENTER**) to get a list of files already saved on the Microdrive. See section 8.3.2 for the ways of listing all or just a sub-set of the files.

EASEL will save all the information, including the graph format and the colours and positions of any text labels, titles and keys that you have added.

You use the Load command to recover a graph that you have previously saved (with the Save command) on a Microdrive cartridge.

You are asked to type in the file name that you gave the graph when you saved it. If you can not remember the name you can type a question mark (followed by **ENTER**) when EASEL will show you a list of the files present on the current Microdrive cartridge. As with the Save command, you can select whether you want to see the names of all the files or just a sub-set. See Section 8.3.2 for details of how to do this.

This command allows you to delete, rename or copy an EASEL file previously saved on a Microdrive cartridge. It also allows you to transfer data between EASEL and the other programs in the Psion 4D package.

This option of the Files command allows you to make a copy of an EASEL file. You can also use it to make a copy of the EASEL program itself. You would be wise to make at least one copy of EASEL on another cartridge, just in case of accidents.

You can copy EASEL from the cartridge in drive 1 to a blank, formatted cartridge in drive 2 in the following way:

```
F3 F B MDV1__EASEL ENTER MDV2__EASEL ENTER
```

With this option of the Files command you can delete an entire file from a Microdrive cartridge. EASEL asks you to type in the name of the file to be deleted.

**Always remember that you can not recover a file which you have deleted, so make sure you really do not want to use that graph again before deleting it.**

These two options allow you to transfer data between EASEL and the spreadsheet and database programs.

You use the Export option to send the data from your graph to the other programs. It sends all the sets of figures, together with their names and the axis labels, but ignores the graph and axis titles and any key or other labels that you have added.

If you export data to the spreadsheet, the names of the sets of figures are interpreted as cell labels. They label each set of figures, the values of which appear in successive cells of a row (or column). The axis labels are interpreted as labels for the columns (or rows) of values.

In the case of exporting to the database, the names of the sets of figures become text fields in database records, with one set of figures in each record. The axis labels are taken as the field names for the values in each set of figures.

You should only import a file that has been exported by either the spreadsheet or the database. The imported figures are displayed in the format which you select by the options of the Change command.

When you import sets of figures Easel gives them names which correspond to text exported with the data from the original application.

### 7.1 SAVING FILES

### 7.2 LOADING FILES

### 7.3 THE FILES COMMAND

#### 7.3.1 Backup

#### 7.3.2 Delete

#### 7.3.3 IMPORT AND EXPORT

# PROVISIONAL

A file exported from the spreadsheet in row order, for example, will produce a set of figures from the data in each row. The label at the beginning of each spreadsheet row becomes the name of each set of figures. The spreadsheet column labels are used as the graph axis labels.

When you import a file from the database, the data from each record becomes a new set of figures. The contents of the first text field in each record are used as the name for that set of figures. The field names of the numeric fields are used as the axis labels.

## 7.4 PRINTING

If you have a printer you can make printed copies of your graphs. The Print command simply makes a printed copy of the graph currently shown on the screen.

If you want to print a graph that you have previously saved on a Microdrive cartridge you must therefore first load it. When the appearance of the graph is exactly as you want it you simply type:

**F3 P ENTER**

The exact appearance of the printed graph will depend on the make of printer that you are using.

## 7.5 PHOTOGRAPHY

The simplest way of obtaining a permanent copy of one of your graphs is to take a photograph of the screen. You must, however, take a little care if you want to obtain good results.

One of the most common causes of a poor photograph of a television screen is using too short an exposure time. The picture is made up of 625 separate lines, displayed one after another. It takes a 25th of a second to display all the lines in the picture and if you use an exposure time of about this length, or shorter, the picture will be unevenly lit. It is best to use an exposure time of around a quarter of a second - this means that you must support the camera on a tripod. An average colour film (for prints or transparencies) with a speed of, say, 100 ASA will need an aperture of around f3.5.

Try to take the photograph in a darkened room, to avoid reflections of the surroundings from the surface of the screen. It is surprising how strongly such reflections show up on the photograph, even if you do not notice any when you look through the camera viewfinder.

## 7.6 THE MICRODRIVES

You can put Microdrive cartridges in both drives and use either drive, provided that you include the drive specifier in the file name. (See Section 8.3 for a full description of Microdrive file names.)

You could, for example, load a file called "PICTURE\_\_GRF" from drive 2 by using the Load command as:

**F3 L MDV2\_\_PICTURE\_\_GRF ENTER**

You do not need to include the extension. EASEL will assume that it is \_\_GRF unless you type in something different. You could therefore just use:

**F3 L MDV2\_\_PICTURE ENTER**

Remember that this will make drive 2 the default drive so that you could now load another file, called "PLOT\_\_GRF" from drive 2 by:

**F3 L PLOT ENTER**

Drive 2 will remain the default drive until you specify a different drive specifier in a file name.

The arithmetic operations in EASEL follow the same rules as for the arithmetic in SuperBASIC. The valid range for numbers is from  $-2.9E-39$  to  $+1.7E+38$ . All calculations are accurate to 17 significant digits but only a maximum of 16 significant digits may be displayed.

The following arithmetic operations are provided:

- + Addition
- Subtraction
- \* Multiplication
- / Division
- ^ Raising to a power

Functions and operations have the following priorities

| Operation                                             | Priority |
|-------------------------------------------------------|----------|
| All functions                                         | 9        |
| ^                                                     | 8        |
| Unary minus (ie, minus just used to negate something) | 7        |
| *, /                                                  | 6        |
| +, - (minus used to subtract one number from another) | 5        |
| not                                                   | 4        |
| and                                                   | 3        |
| or                                                    | 2        |

The five function keys are used as follows:

- F1 call the Help facility
- F2 remove/restore control area
- F3 call the commands
- F4 delete:
  - text
  - labels
  - numbers
  - the key
  - user-defined objects

Note: user-defined objects are bars, lines, graph paper and axes.

- F5 insert a cell

The line editor is always available to modify the contents of the input line.

| Key(s)               | Action                            |
|----------------------|-----------------------------------|
| Left cursor          | Move one character to the left    |
| Right cursor         | Move one character to the right   |
| Up cursor            | Move one word to the left         |
| Down cursor          | Move one word to the right        |
| CTRL + Left cursor   | Delete one character to the left  |
| CTRL + Right cursor  | Delete one character to the right |
| CTRL + Up cursor     | Delete all text to the left       |
| CTRL + Down cursor   | Delete all text to the right      |
| SHIFT + Left cursor  | Move left by one word             |
| SHIFT + Right cursor | Move right by one word            |

### 8.1 ARITHMETIC

### 8.2 THE FUNCTION KEYS

### 8.3 THE LINE EDITOR

### 8.4 FILES

A full file name consists of three sections, separated by underscores. The three components are:

- an optional drive specifier eg MDV1
- a file name of up to x characters eg PLOT
- an optional three-letter extension eg GRF

A full file name for an EASEL file could therefore be:

**MDV2\_PLOT\_GRF**

#### 8.4.1 File Names

# PROVISIONAL

If you do not include a drive specifier in a file name then EASEL assumes that you are referring to the current drive, that is, the drive that was last used. The one exception is when you are loading EASEL itself from SuperBASIC, as described in Section 2.1. In this case you must include the drive specifier in the file name.

You do not normally need to specify an extension since EASEL supplies a default extension for every file access. The Load and Save commands supply a default extension of `__GRF`. The default extension for Import and Export files is `__EXP`, and when you Print to a file the default extension is `__LIS`.

If you include an extension in any file name you type in then it will be used in preference to the default extension normally provided by EASEL.

## 8.4.2 Wild Cards

Every time that an EASEL command asks you to type in a file name you have the option of pressing the ? key to obtain a list of the names of files on the current drive. The file name `"*_**"` (file name and extension) will appear in the input line and, if you accept this by pressing **ENTER**, you will be given a list of all files on the current drive.

In this context the `"**"` character is a *wild card* which stands for any sequence of characters. You may also use the character `"?"` to represent any single character in a file name.

You have the option of using the line editor to modify the suggested file name, in order to obtain a list of the names of a particular group of files.

If, for example, you edit the file name to read `"*_TST"` and then press **ENTER** you will be given a list of the names of all files with an extension of `__TST`. Changing the file name to `"*X*_**"` would result in a listing of all files, with any extension, whose names begin with X.

You could use the single character wild card as, for example:

`MYFILE?_*`

which would result in a listing of all files with names such as:

`MYFILE1 MYFILE2 MYFILE3`

and so on, with any extension.

**Note** that this facility is only available when you are requesting a list of file names before typing in a file name for any of the file-based commands (Files, Load and Save).

## 8.5 THE COMMANDS

The commands give access to the deeper levels of EASEL and allow you to use many of the more advanced facilities. The following commands are provided.

### CHANGE

The Change command allows you to modify the appearance of any feature of the graph.

You are offered the following options:

**Format** - to redefine the appearance of the entire graph. You may choose a defined format by its number, or by example. Selecting design by example allows you to change any feature, including the background colour, style of graph paper grid, the graph scales, colour and width of the border, plus any combination of the following options.

**Bar** - to select or define the style of bar used to represent the current set of figures. You may choose a previously-defined bar by its number, or by example. The design by example option allows you to select a bar or to design a new one. You can choose the bar fill colour and the border colour and thickness.

**Line** - to select or define the style of line used to represent the current set of figures. You may choose a previously-defined line by its number, or by example. The design by example option allows you to select a line or to design a new one. You can choose the line colour and thickness, and the style of mark used for each point on the line, or select a "filled line" where the space between the line and the zero level on the graph is colour-filled.

**Pie** - to select or define the style of pie chart used to represent the current set of figures. You may choose a previously-defined pie chart by its number, or by example. The design by example option allows you to select a pie chart or to design a new one. You can choose the colour used for each sector.

**Graph\_\_paper** - to select both the background colour and the style and colour of the grid markings.

**Axis** - to select the axis markings. You can alter the style and colour of the axes and the colour used for the numbers labeling the y-axis.

**Text\_\_colour** - to select the colour used for both the text and its background. You can select a transparent background so that the underlying graph will show through. Any existing text will retain its original colour, but new text will appear in the selected style, until you change it again. The text in a key box is always drawn in the current text colour.

The Defaults command allows you to select a number of features, such as whether you use a 40 character (suitable for a domestic television) or 64 or 80 character (for a monitor) display. You can select an item by pressing the key corresponding to its first letter in the list of options shown.

## DEFAULTS

After each selection you are allowed to make further selections in the same way. When you have finished you should press the X key to return to the display of your graph.

The Edit command allows you to modify or move text, labels and the key.

## EDIT

You are asked to choose between the following three options:

**Text** - the crosswires lock on to the nearest piece of text and you can use the line editor to change the wording. When you are satisfied with the text you should press ENTER. You are then offered the option of moving the text to a new position with the cursor keys. Press ENTER when you are satisfied with the position.

**Labels** - the crosswires lock on to the nearest axis label and you can then edit the text of the label as in the Text option. When you press ENTER at the end of your editing you are not offered the option of moving the label; labels can not be moved.

**Key** - you are immediately offered the option of moving the key box with the cursor keys. When the outline of the key box is in the position you want you should press ENTER. The key box is then redrawn in its new position.

This command allows you to modify EASEL files, previously saved on a Microdrive cartridge, or to transfer data files to another of the Psion 4D programs.

## FILES

The options ask you to type in the names of files. Each time you are asked for a file name you can press ? for a list of all files on the current cartridge. You can then accept the suggestion of \*\_\_\* to display all the files (by pressing ENTER) or you can use the line editor to change either the file name or its extension, to list any particular subset. This is explained more fully in Section 6.7.

At the conclusion of any option you are left in the Files command menu, ready to use another Files option. You can return to the main display by pressing ESC.

You are offered the following options:

**Files B** used to make a backup copy of an EASEL file. You are asked for the name of the file to be copied. Making copies of your files is strongly recommended, to protect yourself against accidental loss of, or damage to, the cartridge, and against making a mistake which causes your application to be corrupted or deleted.

**Files D** deletes a named file from a Microdrive cartridge. Note that this command is **NOT** reversible and should therefore be used with **GREAT CARE**.

**Files E** exports a named file. The file is saved in a form suitable for being read by the database or the spreadsheet. Note that you should not send files to the word processor by means of the Export option. Such files need additional formatting information and you should therefore use the Print command for this purpose.

All the current sets of figures are written into the file. Each set of figures retains its name and (axis) labels. All other text, including titles and user-defined labels, is ignored. If you do not specify a file name extension for an exported file, EASEL will supply an extension of **\_\_EXP**.

**Files I** imports a named file. It allows EASEL to read files exported by any of the other programs in the Psion 4D package.

If you do not specify a file name extension for an imported file, EASEL will assume an extension of **\_\_EXP**.

**Files R** renames a file. You are asked to type in the original file name, followed by the new name you want to give to the file.

If you do not specify a file name extension for the original file EASEL will assume an extension of **\_\_GRF**. If you do not specify a file name extension for the new file EASEL will assume that it should be the same as for the old file.

## HIGHLIGHT

This command allows you to use a special symbol to represent a particular number in a set of figures. The value to be highlighted is the one at the current position of the intersection of the crosswires.

In the case of a bar graph you are allowed to choose a bar style to be used for the value, either by its style number or by example, as in the Change command.

On a pie diagram you indicate which sector is to be highlighted by moving the box cursor to its label. The highlighted sector is shown as being slightly detached from the remainder of the pie.

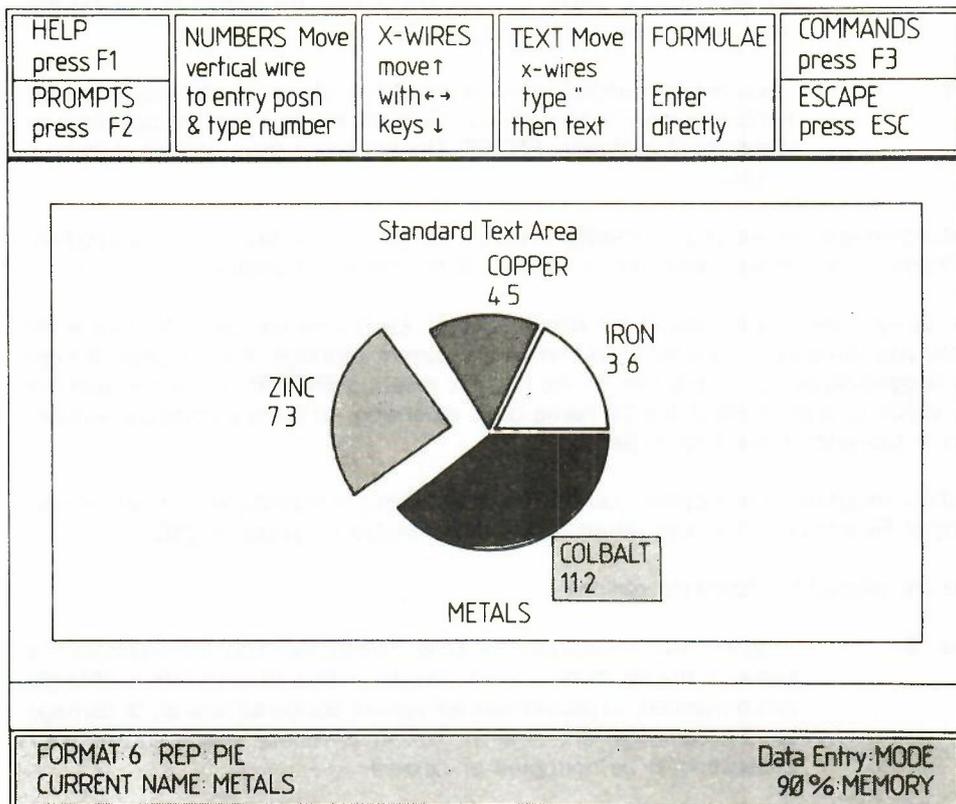


Figure 8.1: A Highlighted Pie Sector.

Deletes one or more sets of figures from the graph and destroys the data. When you select this command you are asked to type in a list of the names of the figures you want to delete, separated by commas and ending with **ENTER**. If you just press **ENTER**, EASEL will delete the current figures.

**KILL**

Loads a previously saved graph from a Microdrive cartridge. All the Design options are loaded with the data so that the graph of the loaded data has exactly the same appearance as it had when it was saved.

**LOAD**

This command allows you to create a new set of figures, which becomes the "current figures". You are asked to type in the name of the new set (no quotation marks are needed). When you press **ENTER** you are returned in data entry mode, ready to type in some values.

**NEWDATA**

The Olddata command allows you to make an existing set of figures the "current figures". You are asked to type in the name of the old set (no quotation marks are needed). When you press **ENTER** you are returned in data entry mode, ready to change or add to the values.

**OLDDATA**

Prints the graph that is currently displayed on the screen. The exact appearance of the printed graph will depend on the make of printer that you use.

**PRINT**

You use this command to leave EASEL and return to SuperBASIC. You are offered the options of:

**QUIT**

**Return - to EASEL**  
**Exit - to SuperBASIC.**

If you choose this option you are asked if you want to save your files before exiting.

This command allows you to rename an existing set of figures. You are asked to type in the old name and then the new name. You should type in both names without surrounding quotation marks.

**RENAME**

Saves all the sets of figures currently in the computer's memory on a Microdrive cartridge. You are asked to type in a name under which the figures will be saved. You can load the graph back into the computer at a later date by using the Load command.

**SAVE**

All the properties of the graph, eg the bar colours and style of axes, are saved with the figures.

You use this command to redisplay your graph, showing all, or a selected few, of your sets of figures. EASEL suggests that all sets of figures are to be displayed and you can either accept this suggestion, by pressing **ENTER**, or type in a list of the names of those sets that you want to be displayed. You should separate the names in the list by commas and end the list by pressing **ENTER**.

**VIEW**

You are then offered a suggested format number for the display. You can accept the suggested format, by pressing **ENTER**, or type in your own choice of format number, followed by **ENTER**.

You can think of a function as a kind of recipe which converts a number of initial values, known as the function's *arguments*, into a different value, which is said to be the value that is *returned* by the function.

## 8.6 FUNCTIONS

The functions provided by EASEL take one or no arguments. The argument for a function is placed in brackets after its name. You must not leave a space between the name and the opening bracket, but spaces are allowed within the brackets. All function names must be followed by the brackets, even if they take no arguments. The presence of the brackets is a useful reminder that you are referring to a function. They allow you to distinguish between the name of a set of figures and a function, even if they have the same name.

The following functions are provided.

**ABS(n)** Returns the absolute value, that is the numerical value irrespective of its sign, of the argument. For example, `abs(5)` and `abs(-5)` both return the value 5.

# PROVISIONAL

|        |                                                                                                                                                                                                                    |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATN(n) | Returns the angle, in radians, whose tangent is n.                                                                                                                                                                 |
| COS(n) | Returns the cosine of the given (radian) angle.                                                                                                                                                                    |
| EXP(n) | Returns the value of e (approximately 2.718) raised to the power n. The returned value will be in error if n lies outside the range from -87 to +88, since the result will then exceed the numeric range of EASEL. |
| INT(n) | Returns the integer value of the number, by truncating at the decimal point. The truncation always operates towards zero. Thus;<br><b>int(3.7) returns 3</b><br><b>int(-4.8) returns -4</b>                        |
| LN(n)  | Returns the natural, or base e, logarithm of n. An error results if n is negative or zero, since logarithms are not defined in this range.                                                                         |
| PI()   | Returns the value of the mathematical constant pi.                                                                                                                                                                 |
| SGN(n) | Returns +1, -1, or 0, depending on whether the argument is positive, negative or zero.                                                                                                                             |
| SIN(n) | Returns the value of the sine of the specified (radian) angle.                                                                                                                                                     |
| SQR(n) | Returns the square root of the number n, which must not be negative.                                                                                                                                               |
| TAN(n) | Returns the tangent of the specified (radian) angle.                                                                                                                                                               |



**PROVISIONAL**

**sinclair**

**QL**

**QL Information**





# RETURNS FORM

MR/MRS/MISS \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

If this address is different from the one to which we sent your QL, please indicate your old address:-

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Please quote your acknowledgement number \_\_\_\_\_

QLUB membership number (if any) \_\_\_\_\_

Please describe the fault which has developed:-

Approximately how many hours a week have you been using your QL?

\_\_\_\_\_ hours per week

How old is your QL? \_\_\_\_\_ months

# RETURNS PROCEDURES

## 1. Guarantee

Your Sinclair QL is covered by a 12-month comprehensive guarantee valid in the UK only and effective from date of despatch. If it should develop a fault within this 12-month period then it will be repaired or replaced free of charge provided 1) it has been put to normal use and 2) it has not been opened or modified by anyone other than Sinclair Research Limited or their agents. Sometimes however, it will not immediately be clear whether the fault lies with the QL itself, or with one of the QL applications software packages. Please follow the instructions below if you have a problem.

## 2. If you have a problem

Check whether it is the QL computer or one of the QL applications software packages (QL Abacus, QL Quill, QL Easel and QL Archive) which is at fault, as follows:

If none of the four software cartridges will load, then it is almost certainly your QL which is at fault, and if it is still under guarantee you should follow the instructions under (3) below.

OR

If the software packages run successfully but all display a similar fault while running, then it is almost certainly your QL which is at fault, and if it is still under guarantee you should follow the instructions under (3) below.

If one of the cartridges fails to load, check all the others

OR

If one or more of the software packages seems to have a fault, check all the others

If any one of the four packages works correctly then it is the individual cartridge(s) which are at fault, and you should follow the instructions under (4) below.

## 3. Returning your QL within the 12-month guarantee period

Please complete the "Returns Form" at the end of this section and send it with your QL, the leads, the power supply unit as well as all four software cartridges (QL Abacus, QL Quill, QL Easel and QL Archive) in their polystyrene box to Sinclair Research Limited, QL Department, Computer Division, Stanhope Road, Camberley, SURREY GU15 3BR. Please obtain and keep "proof of posting". PLEASE DO NOT RETURN YOUR QL MANUAL.

Within a week of receiving your QL package we will return an equivalent fully-functioning package to the address you indicated on your "Returns Form". In some cases you will receive your original computer repaired. In other cases we may send you a replacement QL. This will depend on the nature of the fault which your computer has developed.

Please note that your 12-month guarantee extends from the date you received your first QL. It does not extend for 12 months from when you receive any subsequent repaired or replacement QL although this latter computer would also be covered under the *original* 12-month guarantee.

## 4. Returning one of your QL Applications Software Packages

Please complete the "Returns Form" at the end of this section and send it with the faulty cartridge(s) to Sinclair Research Limited, Computer Division, Stanhope Road, Camberley, SURREY GU15 3BR. Please do not return all four cartridges - only those which are faulty.

We will replace the faulty cartridge(s) within one week from receipt.

## 5. If you have problems using your QL Applications Software Packages

1. Refer to your manual.
2. Consider joining the QLUB (the QL Users Bureau) to which you may subscribe at any time irrespective of when you purchased your QL. You will find full details of what the QLUB can offer you and an order form at the back of this manual.
3. Refer to books published about the QL.
4. Please do not phone Sinclair Research unless this is your last resort. Our customer services department does not have detailed technical information.

# Sinclair QL Preservation Project (SQPP)



On January 12<sup>th</sup> 1984 Sir Clive Sinclair presented the Sinclair QL Professional Computer in a typical Sinclair-extravaganza type launch event at the Intercontinental Hotel, Hyde Park Corner, London. This was exactly 12 days earlier than Steve Jobs presented the Apple Macintosh.

The QL is a very good example of an innovative, stylish, powerful and overall underestimated product and ecosystem. On one hand it failed in the market but on the other hand it influenced many developments which ended in many of today's computing devices.



Check out the website <http://sinclairql.net/> – The semi-official website related to the Sinclair QL Professional Computer. **QL forever!**

Urs König (aka QLvsJAGUAR)

[http://sinclairql.net/about\\_urs.html](http://sinclairql.net/about_urs.html)

<https://www.youtube.com/QLvsJAGUAR>

<https://plus.google.com/+QLvsJAGUAR>

# Sinclair QL Preservation Project (SQPP)



On January 12<sup>th</sup> 1984 Sir Clive Sinclair presented the Sinclair QL Professional Computer in a typical Sinclair-extravaganza type launch event at the Intercontinental Hotel, Hyde Park Corner, London. This was exactly 12 days earlier than Steve Jobs presented the Apple Macintosh.

The QL is a very good example of an innovative, stylish, powerful and overall underestimated product and ecosystem. On one hand it failed in the market but on the other hand it influenced many developments which ended in many of today's computing devices.

## Document details

Topic: Sinclair QL User Guide - Launch Edition

Notes: This is the Launch Day Edition of the Sinclair QL User Guide. It dates 12th January, 1984. This specific one came with a pre-production QL (S/N D02-001071, ISS 4 PCB).

Number of pages (including SQPP cover and back pages): 366

Scanned: 2018-06-13 on a HP M477 fdw

Check out the website <http://sinclairql.net/> – The semi-official website related to the Sinclair QL Professional Computer. **QL forever!**

Urs König (aka QLvsJAGUAR)

[http://sinclairql.net/about\\_urs.html](http://sinclairql.net/about_urs.html)

<https://www.youtube.com/QLvsJAGUAR>

<https://plus.google.com/+QLvsJAGUAR>