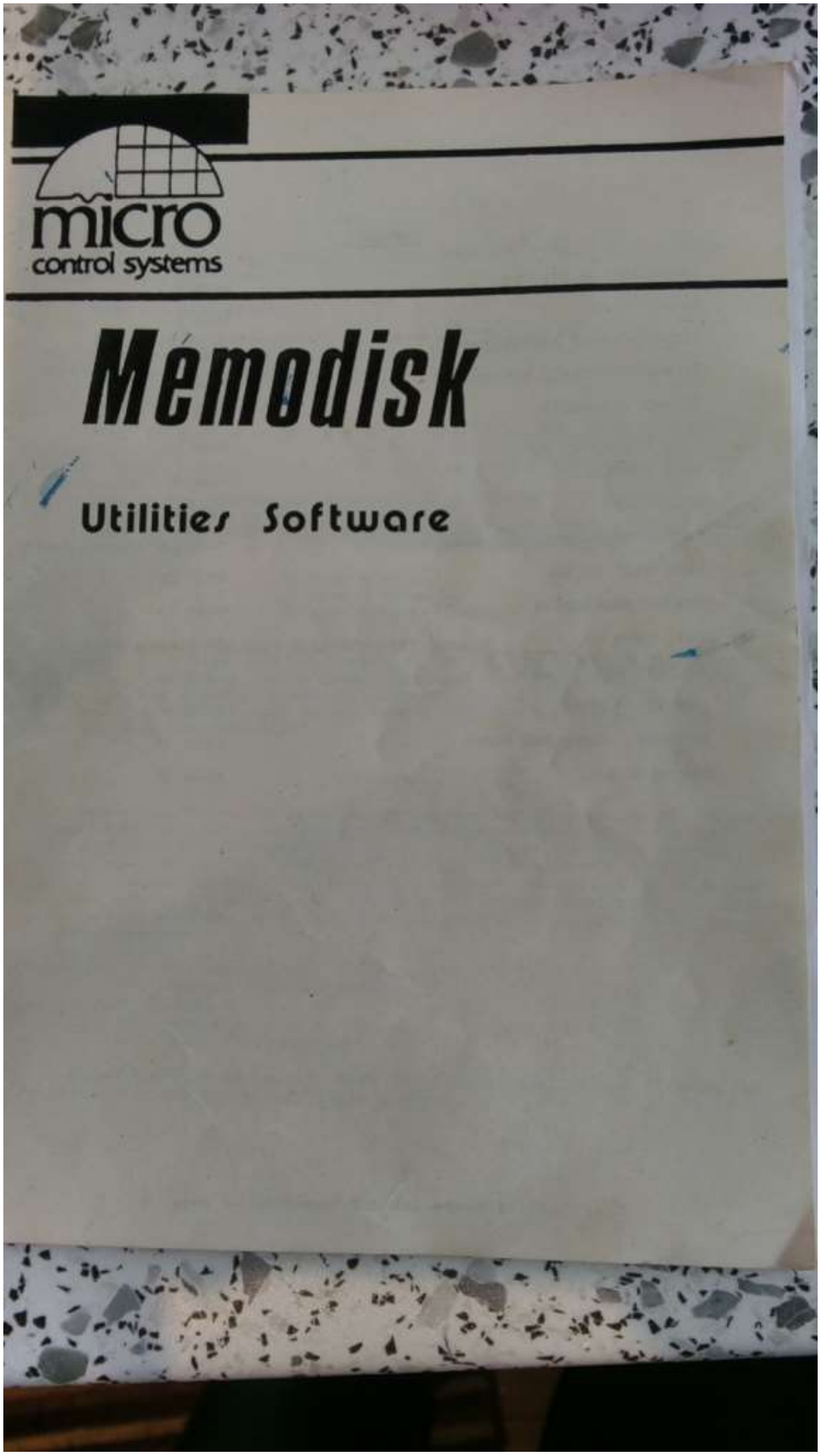
micro
control systems
Mémodisk
Utilities Software

CONTENTS

SECTION 1
INSTALLATION AND USE

Installing the ROM board.

The ROM cartridge plugs into the ROM port on the rear of the QL. Its protective cover may be removed by pushing in at one end so that the other end is forced outward, now simply pull it out.

THE ROM CARTRIDGE MUST NEVER BE REMOVED OR INSERTED WHILST THE POWER IS ON. CHECK THAT THE POWER IS TURNED OFF NOW.

Insert the cartridge with the chips facing downward into the slot as far as it will go. When you are sure that it is inserted correctly, turn on the power. After the memory test you should get the signon message of the Ramdisk device printed at the top of the screen.

The Ramdisk is now present in the system, you will need to refer to the section on formatting before it is useable. To load in the basic extensions, type RAM_EXT this will link in the extra procedures and functions and setup the KEY_SCAN job.

COPYING THE UTILITIES TO DISK.

Provided that you have the MCS disk interface installed the the contents of the ROM cartridge may be copied to disk, and this is done with the following line of basic:

```
SBYTES FLP1_MCS_TOOLS,49196,16340 (enter)
```

This will save a copy to disk under the file MCS_TOOLS (the name could be, of course, anything you prefer). This file may be copied to any number of disks, but it will only run if the MCS disk interface is present.

RUNNING THE UTILITIES FROM DISK

To run the utilities from disk the following line of basic may be typed in directly or inserted as part of a boot file.

```
a=RESPR(16340):LBYTES FLP1_MCS_TOOLS,a:CALL a
```

The filename will be the name under which you saved the code. RAM_EXT will then initialise the extensions.

SECTION 2
RAMDISK USE.

The ramdisk is a new directory device in your QL. It operates exactly the same as disks or microdrives but with greater speed. Any data written to ramdisk files is stored in the QL's memory, this allows very fast access of data. There is of course a drawback - anything in the ramdisk when the power is turned off or the computer reset is lost.

Ramdisks are very useful when an amount of data needs to be accessed regularly and when permanent storage is not of importance.

To use the Ramdisk it must first be formatted, this differs from other devices that have to be formatted in that instead of specifying the medium name, the number of blocks required must be entered. On the ramdisk one block is worth 32K of storage space (64 traditional sectors) so the line :

```
FORMAT  RAM1_1
```

will set up a 32K ramdisk with 64 sectors. If no size is specified at all then the default size is 2 (64K or 128 sectors). The maximum size of ramdisk allowed is 8 (256K or 512 sectors).

If the memory used by the Ramdisk is required to be released then the ramdisk should be formatted with a size of 0.

FORMATTING THE RAMDISK WILL DESTROY ALL ITS PREVIOUS CONTENTS.

Although the Ramdisk can only be formatted with RAM1 it will respond to RAM1 - RAM8, the information being the same on all device numbers.

RAMDISK SUPERBASIC EXTENSIONS

RAM_OVER procedure

This procedure forces any open_new type commands to overwrite the file specified if it exists. For example:

```
[line no] RAM_OVER
[line no] SAVE RAM1_BOOT
```

If the file BOOT already exists on ram1_ then it will be overwritten with the new file.
RAM_OVER is valid for all the following commands and in the case of copy affects only the destination file.

OPEN_NEW, SAVE, SBYTES, SEXEC and COPY

RAM_OVER will remain active until reset or the command RAM_PROT is Issued.

RAM_PROT procedure

This procedure is the opposite of RAM_OVER and is used to turn off the overwrite mode.

RAM_USE procedure

This procedure, when called, will allow the name of the ramdisk to be changed. The new name can be any three letters but will most usually be used to rename the device to flp, this enables software configured to run on floppy disks to run on ramdisk.

[line no] RAM_USE 'FLP' or RAM_USE flp

RAM_EXT procedure.

This procedure will install the Superbasic extensions, The NDW device and the KEY_SCAN job.

[line no] RAM_EXT

SECTION 3
MICRODRIVE SUPERBASIC EXTENSIONS

MDV_USE

This command is fundamentely the same as RAM_USE except that it allows the microdrives to be renamed to another device name. Consider the following program.

```
10 FORMAT raml_4
20 WCOPY mdv1_ TO raml_
30 mdv_use MIC
40 ram_use MDV
```

After running this program all the contents of the Microdrive that was in MDV1_ will have been copied to ramdisk. The Ramdisk now responds the the name MDV and will run any software as if it were on microdrive. To save data back to microdrive then the name MIC must be used.

MDV_OVER
MDV_PROT

As RAM_OVER/PROT except that they apply to MDV.

SECTION 4
SCREEN UTILITIES

NON-DESTRUCTIVE WINDOWS.

A new device driver is contained in the utilities. It is called NDW and behaves exactly the same as the CON device except for one major difference - it saves the screen area which it corrupts. It is used exactly the same as CON or SCR and any programs written to output to these devices may be easily changed (basic or machine code.)

When the channel is opened, NDW examines the parameters regarding the size and position of the window and saves the corresponding screen memory in the QL's common heap.

Upon closure of the channel the screen underneath the window is restored to preserve the display.

The use of NDW does of course take up QL memory and it is worth remembering that a full size window can use up to 32K of memory, therefore try to restrict the size of the window to a reasonable size or dont open too many windows at once!. If an attempt to open a window is made with insufficient memory space available then the window is simply opened as CON.

example:

```
100 OPEN #3,NDW_178x60a160x90
110 PAPER #3,7:CLS #3:INK #3,0
120 PRINT #3,'HELLO WORLD'
130 PAUSE 150
140 CLOSE #3
```

A couple of points to note are: (i) because the screen image is taken when the window is opened any subsequent moving of the window does not move the image and (ii) windows (where overlapping) must be closed in the correct sequence (the one on top first).

CAPS

This procedure can be used to turn the caps lock on or off. Useful in programs where all user input must be in one letter case. The procedure requires one parameter this being 1 for caps lock on and 0 for caps lock off.

```
SYNTAX  [line]  CAPS 0
or              CAPS 1
```

CURSEN
CURDIS

Can be used to turn on or off a cursor in a specified channel. Will default to channel 1 if none specified.

```
SYNTAX  [line]  CURSEN [#channel_no]
```

EDLIN

This procedure is designed to improve the input of information to a basic program. When called the string specified in the parameters is printed on the screen for editing. The string may be edited using the normal keys for delete . When complete, the string may be returned by pressing Enter, cursor up or cursor down. If a third parameter is specified when the procedure is called then this will contain the character used for terminating the input on return.

SYNTAX EDLIN [channel,]string,[terminator]

eg. 100 EDLIN #3,a$,term%

See the program in appendix A for an example of EDLIN's use.

CLOCK

The in-built clock may be activated by simply typing CLOCK [channel]. The channel is optional and will default to channel 1. The clock will be setup in the top right hand corner of the specified channel. It will also acquire the ink and paper attributes of its associated channel. The clock will run as an independant job and may be removed in 2 possible ways.

(i) Close the window to which it is attatched.
(ii) Look up its ID with 'JOBS' and remove it with 'RJOB'

See Appendix A (line 130) for an example of its use.

SECTION 5
FILE HANDLING

The utilities ROM contains a number of procedures and functions to make the handling of files easier.

FOPEN

This function requires two or three parameters and is used for the opening of channels (in place of Superbasic OPEN) . It differs in that it returns a value representing the QDOS error code for the open. Thus is is possible to intelligently open channels and detect errors.

In addition the open type can be specified as a value in the range 0-4 which corresponds to the table below:

0	open existing file (exclusive use)
1	open existing file (shared use)
2	open new file (exclusive use)
3	open new file (overwrite if exists)
4	open the directory

The errors returned correspond to the QDOS error codes and are presented below:

-3	Out of memory
-6	Not opened (too many channels)
-7	File or device does not exist
-8	File already exists
-9	File or device is in use
-12	Bad file or device name

To report errors see the FAULT command.

SYNTAX error_code=FOPEN (#channel,opencode,filename$)

Note that the Data default directory may be used with this command.

See Appendix A (line 570) for an example of its use.

FLEN

This function returns the length of the specified file. The file must be open to a superbasic channel, this being the only parameter required.

SYNTAX length=FLEN(#channel_ no)

FDAT

This function is the same as flen but returns a programs required data space.

SYNTAX data=FDAT(#channel_no)

FTYP

This function returns an integer to say the type of file opened to the specified channel. It returns a 0 for normal files and 1 for files that must be exec'd.

SYNTAX type=FTYP(#channel_no)

TRUNCATE

The file opened to the specified channel is truncated to the current file pointer position. Everything past this point is lost.

SYNTAX [line] TRUNCATE #channel_no

FLUSH

This procedure will ensure that the buffer for the specified channel does not contain any characters waiting to be written to a device.

SYNTAX [line] FLUSH #channel_no

RENAME

This procedure will rename the specified file to the new filename. Note that both device names must be the same and may use the data default device.

```
Example:    RENAME bill to fred
    or:     RENAME flp1_bill to flp1_fred
```

SPOS
SPOS_R

These procedures move the file pointer in the specified channel. SPOS sets the pointer to an absolute address and SPOS_R sets it relative to its previous position.

SYNTAX [LINE] SPOS #channel_no,position

BGET
BPUT

These procedures send and recieve bytes to and from the specified channel.

BPUT [channel,] n is the same as PRINT [channel,] CHR$(n):

BGET [channel,] n is the same as n=CODE(INKEY$([channel,]-1))

in addition several bytes may be sent or recieved with one command .

```
    ie      BPUT #3,100,90,40,34,65
            BGET #3,a,b,c,d
```

Using BGET/BPUT and the SPOS commands bytes may be used to put or read bytes from any point in a file.

FAULT

This procedure will print out the QDOS error message of the negative value which is passed to it.

SYNTAX [LINE] FAULT [channel,]error_code

This command is most useful when used in conjunction with FOPEN as errors in opening files may be printed.

See appendix A (line 510) for an example of its use.

SECTION 6
FILE SPOOLER

The file spooler operates as an independant job and may be run as a background task to copy a file from one place to another. The devices may be channels that are already open or files using the data default device. Note that if the destination file already exists then it will be overwritten by the new data.

There are several different versions of the spooler and are outlined below.

SPL spool a complete file including the file header to the specified file or device.

SPL_N spool a file without its header.

SPL_P used for text based files - will spool the specified file and insert a Form Feed character after every 60 lines sent. (useful for copying basic programs to a printer.

SYNTAX [line] SPL filename to filename
[line] SPL filename to #channel

VIEW

This is a special version of the file spooler. It is designed for copying files to the screen. If the destination is a window then its size is read and the lines are truncated to prevent wrap-around. When the window is full of text the screen is frozen. Press control-F5 to continue.

As an example, suppose that in a program you wish to display some text help in a help file, then this may be easily accomplished with :- VIEW help_text to #3 or wherever. The text will be copied to the output channel pausing between each 'page'.

SECTION 7
WILD CARD PROCEDURES.

The utilities ROM contains several procedures for file maintenance that can use default devices and wild cards.

WDIR Lists a directory of matching filenames

SYNTAX [line] WDIR [#channel,][wild_name]

WSTAT Lists the file name,length and date of last update.

WDEL Deletes matching files, prompts user for action.

WDEL_F Deletes matching files automatically - no user input.

WCOPY Copies matching files, prompts user for action.

WCOPY_F Copies matching files automatically - no user input.

The wild name can be used to reference whole groups of files by specifying only part of the filename. Filenames can be thought of as split into sections by the underscore character. In a wild name an undersore may be used to replace sections of a filename. In the following examples FLP2_ is assumed to be the data default directory.

wild name	Typical matching files
accounts	flp2_accounts flp2_accounts_may flp2_accountsfile
_dbf	flp2_database_dbf flp2_june_accounts_dbf
june__dbf	flp2_june_accounts_dbf flp2_june_wages_dbf flp2_june_week_2_dbf

When WDEL or WCOPY is entered, each matching filename is printed to the specified or default channel together with the prompt Y/N/A. The user can press one of four possible keys.

Y	Delete/Copy this file.
N	Do not delete/Copy this file.
A	Delete/Copy all remaining matching files.
esc	Quit. Do not delete/copy this or any more files.

* note that pressing any other key will be interpreted as N and if no key at all is pressed for 10 seconds then WDEL/WCOPY will terminate itself.

Wild card copy.

Files may be copied from one device to another using this command. The command takes the form:

WCOPY wildname to wildname

The source name (first parameter) can use the data default device. Any characters in the destination name will be placed in front of the filename of the file being copied.

Example WCOPY mdv1_ TO flp1_

Will copy all of mdv1 to floppy disk. If the destination file already exists then WCOPY will be terminated with the 'already exists' error message. Use FLP_OVER,RAM_OVER or MDV_OVER to defeat this.

STAT

Provides directory device information using the data default or specified device name. Prints on the specified or default channel the medium name, the number of free sectors and the total number available.

SYNTAX [line] STAT [#channel_no,][device_name]

DEFAULT DEVICE NAMES

DATA_USE The data default device name, used by wildcards.

SUB_USE The submit default device name,used by SUBMIT.

SYNTAX [line] PROG_USE device_name

eg: DATA_USE FLP2_

* note that at initialisation the DATA default is 'FLP1_' and SUB is 'FLP1_SUB'

SECTION 8
PROGRAMMABLE FUNCTION KEYS.

The utilities rom sets up a job in the QL called KEY_SCAN whose job it is to monitor the keyboard to see if the function keys have been pressed.

When a function key is pressed, provided that it has been assigned a string using the KEY keyword (see below) then the string will be inserted into the current queue.

LAST LINE RE-CALL

The key scan job also is looking for the combination of Shift and Enter, when detected the last line entered will be re-submitted to the input queue allowing it to be edited and re-entered.

KEYS_OFF

To protect against the function keys affecting other software they may be turned off with this procedure. There are no parameters.

SYNTAX [line] KEYS_OFF

KEYS_ON

Similarly the key_scan job may be re turned on with KEYS_ON.

SYNTAX [line] KEYS_ON

* note that the TASK software automatically dissables the function keys when run.

KEY

This procedure requires two parameters, the first is the key number in the range 1 to 5, and the second is the string which is to be appended to the key.

SYNTAX [line] KEY no,'string'

eg: KEY 1,'DIR FLP1 '&CHR$(10)
Will produce a directory of flp1 everytime F1 is pressed.

SECTION 9
MEMORY MANAGMENT

FREE_MEM

This function returns the amount of available memory in the QL minus $200 which is the amount required for one slave block. It is inadvisable to take the full amount of memory as this will slow disk operations down quite a bit.

Syntax variable = FREE_MEM

or PRINT FREE_MEM

ALCHP

This function allocates memory (the amount is specified in the parameter) and returns the base address of the allocated memory. If not enough memory is available then the address returned is 0.

Syntax address = ALCHP(memory_required)

* note that unlike RESPR ,ALCHP can be entered at any time provided that FREE_MEM shows that there is enough space.

RECHP

This procedure releases an area of memory previously allocated with ALCHP. It requires 1 parameter, this being the base address of the space to release.

SYNTAX [line] RECHP address

CLCHP

This procedure releases all areas allocated by ALCHP. Its is provided for use where the address of the memory to be allocated has been forgotten. Note that if the memory released contains an active job then the system will almost certainly crash.

SYNTAX [line] CLCHP

SECTION 10
JOB CONTROL

JOBS

Prints out information on all the jobs running in the machine at the time. Printed out to the specified or the default channel is the job id, its tag, its owner, and its priority. Finally if the job has been setup with a standard header then the jobs name is printed.

Syntax JOBS [channel id]

example printout

```
Job Tag   Own Pri   Name
0   0     0    32
1   0     0   S1   CLOCK
2   1     0    1   KEY_SCAN
```

* note that suspended jobs are shown with an S preceeding the priority.

SPJOB

Sets up the priority of a job. Three parameters are required and these are, job,tag & priority.

Syntax [line] SPJOB job,id,priority

RJOB

Removes a job. Three parameters are required and these are, job, tag & error code. If another job is waiting for completion of the job that is to be removed then it will be released with the error code in D0.

Syntax [line] RJOB job,id,error_code

SECTION 10
MULTITASKING WITH TASK

TASK is a very powerful superbasic procedure. Using TASK it is possible to run all of the PSION programs simultaneously. In order for TASK to know what programs to run it needs a set of parameters which it reads from a file. This file may be created with a text editor or from a basic program (see the basic program in Appendix A).

The parameters are as follows:

```
filename
workspace
filename
workspace
(up to four files and workspaces)
```

The filename must be the full device name (no defaults), and the workspace is the memory that will 'appear' to be available to the program when it is initialised. The amount of memory required by a package is difficult to calculate ,and is largely a case of trial and error. The PSION packages generally need about 100K.

Assuming a file called PSION has been set up with the following parameters:

```
FLP1_QUILL
100000
FLP1_EASEL
100000
```

When the line TASK PSION (enter) is entered the tasker will proceed to set up the two programs specified (providing of course that there is enough memory available, and the files specified can be found). After a short while the TASK CONTROLLER menu will appear.

To select the option required press one of the function keys 1 to 4 . You will then be in the selected package and everything will work just as if it where the only program running at that time.
To call up the TASK CONTROLLER hold down the control key and press the character "S". The menu will re-appear for you to select which program is required.
When a new program is entered all the data in the other programs remains intact and can be returned to at any time.

Any time that the TASK CONTROLLER is displayed pressing control F1 will return you to BASIC. It is important to note that stopping the packages in this way will lose any data not stored and any open files will not be closed.

The TASK CONTROLLER should be able to run any packages that are normally run from basic with 'exec' or 'exec_w'. If the control S function clashes with a key in the package then it can be changed at the start of the parameters file. If the first character in the file is a % then the decimal number following it is the value of the key used to call up the TASK CONTROLLER.

In the previous example : if the line %24 is added to the list, ie:

```
%24
FLP1_QUILL
100000
FLP1_EASEL
100000
```

This will change the character that summons TASK CONTROLLER to CONTROL-X .
A full list of the keys and their values can be found in the QL user quide in the concepts section.

SECTION 12
SUBMIT COMMAND

The SUBMIT procedure is designed for use where a fixed set of instructions must be used over and over. Putting the instructions into a file and then submitting the file simplifys this process. In addition the submit procedure can be asked to wait for another program to start up before outputing any more characters.
examples:

(1) A window is to be opened, the jobs running in the machine listed on it, then close the window.

A file must be created containing the following:

```
OPEN #3,CON_250x100
BORDER #3,2,5:CLS  #3:INK #7
JOBS #3
CLOSE #3
```

Assuming the file is saved as FLP1_LISTJOBS then typing:

```
SUBMIT FLP1_LISTJOBS
```

will have the required affect.

2) A program is to be run (in this case metacomco's ed) with pre-determined file and workspace.

A file must be created containing the following:

```
EXEC_W FLP1_ED
%Q
FLP1_FILENAME_HERE
25K
N
```

If the file is saved as FLP1_RUNED then typing:

```
SUBMIT FLP1_RUNED
```

will run the editor with the specified file in its workspace. In this example, first the job is started, then submit sees the %Q. It does not pass this information on to the editor but waits for it to setup its input queue. The rest of the commands in the submit file are all related to the prompts that the editor generates, source file?, workspace?, move window?.

SUB_DEF

The SUB_DEF procedure is used to set up the default directory and can be up to 20 characters long. When RAM_EXT is first used the SUB_DEF is setup to be FLP1_SUB_ . This means that if all user generated submit files are started with the letters 'SUB' ie 'SUB_LISTJOBS' and are kept on drive 1 then there are two advantages:(1) You only need to type in SUBMIT LISTJOBS to have the same effect as SUBMIT FLP1_SUB_LISTJOBS and (2) all submit files are easily spotted in a directory by virtue of the fact that they all begin with SUB.

SECTION 13
Windows, Icons & Fonts

The utilities ROM also contains EIGEN's excellent windows, Icons and Fonts extensions, together with an FX-80 compatible screen dump.

The extensions are linked in by simply typing WIF_EXT. Another 17 procedures and 2 functions are now available to the user. These are:

EDIT_ICONS	create library of icons.
EDIT_FONTS	create new fonts.
FONT_L	replace the font on a given channel.
FONT_H	
FONT_LH	
ICON	print an icon.
ICONS_USE	load an icon library for use.
FONTL_USE	load in a low font.
FONTH_USE	load in a high font.
CHINK	set the cursor increment.
OPEN_W	open up a window.
CLOSE_W	close a window.
UP_W	move a window up.
DOWN_W	move a window down.
LEFT_W	move a window left.
RIGHT_W	move a window right.
SWAP_W	swap overlapping windows.
X_W	return X co-ord of a window.
Y_W	return Y co-ord of a window.

WINDOWS.

OPEN_W

The open procedure saves the specified screen area thereby allowing it to be overwritten and enabling the background to be restored later. There are no defaults.

SYNTAX: OPEN_W #channel,width,height,x,y

Note that in order to make the most economical use of memory the x-coordinate and the width of the window must be a multiple of 8.

CLOSE_W

The close procedure restores the screen under the window and closes the Basic channel. Note that Superbasic's CLOSE should not be used on channels opened with OPEN_W.
When closing windows that overlap the top window should always be closed first.

SYNTAX: CLOSE_W #channel

RIGHT_W, LEFT_W, UP_W, DOWN_W.

These procedures move the specified window in the corresponding direction. RIGHT_W and LEFT_W only move the windows in mulitples of 8 pixels.

SYNTAX: RIGHT_W #channel
etc.

SWAP_W

This procedure will swap the two windows specified given that the #n lies below #m.

SYNTAX SWAP_W #n,#m

X_W, Y_W

These functions return th X & Y coordinates of the top left-hand corner of the specified channel.

SYNTAX a= X_W(#channel)
b= Y_W(#channel)

ICONS.

In order to use Eigen Icons you will first need to generate a file of icons.
This is done by entering EDIT_ICONS. You will be presented with a menu. This will allow you to (i) edit a particular icon from the file currently in memory, (ii) load an icon file, (iii) save the current icon file, (iv) view all 32 icons in the current icon file or (v) return to Superbasic.
When editing an icon you may move th cursor around the screen using the arrow keys and keys 't', 'y', 'g' and 'h' for diagonal motion.
Holding down the SHIFT key whilst moving the cursor will fill in pixels.
Holding down the control key whilst moving the cursor will delete pixels.
Holding down the shift/control keys and pressing 'C' will clear the icon completely.
Press ESC to return to the icon menu.

When loading or saving icon files:

(i) do not append _icons (this is done automatically)
(ii) you must specify the device name.

If you mistakenly request a save or load option then simply press ENTER after the device name prompt to abort.

ICONS IN SUPERBASIC

To load an icon file use: ICONS_USE device_name

EG: ICONS_USE flp1_lib1

You are provided with one procedure - dont worry though it's very flexible:

ICON index,x,y,ink,paper,border (all parameters needed).

The parameters are defined as follows:

index - icon number (1-32)
x - x-coordinate of icon (must be a multiple of 8)

```
y       - y-coordinate of icon
ink     - ink colour for icon
paper   - paper colour for icon
border  - border colour for icon
```

In order to implement an icon menu you must write a superbasic program that operates as follows:

(i) display your icons on the screen with the same ink paper and border colours.
(ii) highlight the first icon by using a different ink or paper colour.
(iii) take a SPACE keypress to indicate that the next icon should be highlighted.
(iv) take an ENTER keypress to indicate that an option from your icon menu has been selected.

FONTS.

In order to use Eigen fonts you must first generate a file of fonts.
This is done by entering EDIT_FONTS. You will be presented with a menu giving the options: (i) edit a particular font from the file currently in memory, (ii) load a font file, (iii) save the current font file, (iv) view all 128 fonts in the current font file or (v) return to Superbasic.
All of these functions are used in a similar way to the icon editor, and you should refer to the section on icons for details. Please note that the font indexes run from 0-127.

Fonts in Superbasic.

To load a font file use:
```
FONTL_USE device_filename
FONTH_USE device_filename
```

The L and H distinguish between the low and high 128 sections of the character set.

Lets now consider the character fonts on the QL. These have codes 0-255. Please refer to 'character set and keys' in the concepts section of the QL user guide.
In fact the 256 characters are split into 2 groups: 0-127 and 128-255. We will refer to these as low fonts and high fonts respectively.
The 128 fonts in an Eigen font file may be used to replace the Low fonts, the High fonts or both.However, we recommend that you replace only the high fonts. Keeping the Low fonts free for their normal use.
On the QL different windows may have different character fonts. For example, you may leave both Low and High fonts unchanged on channel #0, replace the High fonts on channel #2 with an Eigen font file and replace the Low fonts on channel #6 with another font file.
To replace the fonts on a channel use the following commands:

```
FONT_L  #n      - replace Low fonts on #n
FONT_H  #n      - replace High fonts on #n
FONT_LH #n      - replace Low and High fonts on #n
```
#n defaults to #1.

CHINK

With this procedure the number of pixels the cursor increments in horizontal and vertical directions may be changed.

SYNTAX: CHINK #channel,x_inc,y_inc

SCREEN DUMP

The Eigen screen dump is probably the most advanced screen dump for the Sinclair QL. It produces a full sized printout, allows selection of any part of the screen, simulates colour, automatically distinguishes between high and low resolution modes and allows a dump at any time without the need to enter a command.

How to start an Eigen Dump.

Eigen dump is invoked from the KEYSCAN job therefore both RAM_EXT and WIF_EXT must have been entered to do a dump.
The dump can be started by two different combinations of keys, these are :

CTRL-P	do a dump to the parallel port.
CTRL-D	do a dump to serial port 1.

When invoked you will be presented with two vertical and two horizontal lines which define the area to be sent to the printer. These lines may be moved by using the arrow keys in conjunction with SHIFT. Pressing ENTER will execute the dump. Alternatively, pressing ESC will abort the dump.

Occasionally the image to be printed may be better presented in reverse format. To achieve this, pressing CTRL-R whilst the bars are on the screen will reverse the whole screen prior to printing. After the dump to reverse the screen press CTRL-P, CTRL-R and ESC.

The screen dump may be turned on and off with the commands KEY_ON and KEY_OFF (see section 6).

****

****

APPENDIX A

The following basic program may be used to generate the ASCII only files needed for TASK and SUBMIT. It also serves to illustrate several of the utilities commands. Type the program in, save it, then run it. AT the ?? enter L for Load, S for save and Q for quit.

```
100 OPEN #5,NDW_300x24a100x30
110 INK #5,7:CSIZE #5,2,1:BORDER #5,2,2:CLS #5
120 PRINT #5,'       MINI_EDITOR'
130 CLOCK #5
140 OPEN #3,NDW_300x130a100x50
150 BORDER #3,2,2:CLS #3:INK #3,7
160 maxrow=10:DIM a$(maxrow,80)
170 row=maxrow+1:command:row=1
180 :
190 REPeat EDDY
200   pcur 0:PRINT #3,'> ';
210   EDLIN #3,a$(row),term%
220   pcur 0:PRINT #3,'  ';
230   IF term%=208 THEN row=row-1:ELSE row=row+1
240   IF row=0 THEN row=1
250   IF row=(maxrow+1) THEN command
260 END REPeat EDDY
270 :
280 DEFine PROCedure command
290   pcur 0:PRINT #3,'??'
300   BGET #3,a:IF a=208 THEN
310    pcur 0:PRINT #3,'  ':row=maxrow
320    return
330   END IF
340   if a=115 or a=83 THEN fsave
350   if a=108 or a=76 THEN fload
360   if a=113 or a=81 THEN CLOSE #3:CLOSE #5:STOP
370   GO TO 290
380 END DEFine
390 :
400 DEFine PROCedure fsave
410   open_file 6,3
420   FOR rec=1 TO maxrow:PRINT #6,a$(rec):NEXT rec
430   CLOSE #6
440   pcur 2:CLS #3,4
450 END DEFine
460 :
470 DEFine PROCedure fload
480   open_file 6,1
490   FOR rec=1 TO maxrow:INPUT #6,a$(rec):NEXT rec
500   CLOSE #6
510   CLS #3:FOR rrow=1 TO maxrow:AT #3,rrow,2:PRINT #3,A$(rrow)
520   NEXT rrow
530 END DEFine
540 :
550 DEFine PROCedure open_file(ch,opcode)
560   pcur 2:INPUT #3,' Enter filename ';file$
570   ferr=FOPEN(#ch,opcode,file$)
580   IF ferr<>0 THEN
590   pcur 2:CLS #3,4
600   pcur 3:FAULT #3,ferr:PAUSE 150:goto 560
610   ENDIF
620 END DEFine
630 :
640 DEFine PROCedure pcur(x)
650   AT #3,row,x
660 END DEFine
```

micro
control systems

Micro Control Systems
Electron House
Bridge Street
Sandiacre
Nottingham NG10 5BA
Telephone (0602) 391204

ADDENDUM

Since writing this manual certain problems have come to light, these are explained below.

Page 1 (Installation)

"It is not recommended that the utilities ROM cartridge is fitted and the utilities loaded from disk at the same time as this will duplicate the keywords and may confuse the QL."

Page 21 (Appendix A)

"The program in Appendix A is designed as a simple way of putting characters into a file. It also serves to demonstrate the use of EDLIN, FOPEN, FERR, CURSEN, CURDIS, BPUT, BGET and NDW procedures, functions and device.
Type in the program and save it - It may be used as follows:

When started there will be two question marks (??) at the top of the screen, this is the prompt. At the prompt press;

'L'	To load a file.
'S'	To save a file.
'Q'	To quit without saving.

Pressing the cursor down key will put you into edit mode . The Up and down cursor keys move the cursor to the previous and next lines. At any line simply type in the data. When done move the cursor to the top line, the two question marks will appear, now type 'S' then enter your filename. The device name is not needed (Unless you wish to use other than the data_use default)."

JM USERS

There are a few differences when using the JM version of the QL (PRINT VER$ to find your version). The main difference that concerns the utilities rom is the way in which it installs the extra keywords. Because the JM will not accept new machine code keywords installed as part of a basic program they must be entered in direct mode. The easiest way to do this is to put them in a boot file that does not contain line numbers. To do this use the program in appendix A and type in the following:

```
a=respr(16340)
lbytes flp1_mcs_tools,a
call a
ram_ext
lrun flp1_your_boot
```

Save this file as 'boot'. When run it will load in the file called 'your boot' which may be any basic program.

```
100 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
110 REMark     simple text editor program
120 REMark     Micro Control Systems
130 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
140 :
150 REMark initialisation
160 :
170 OPEN #3,ndw_345x256a80x0
180 BORDER #3,1,2:CLS #3:INK #3,7
190 maxrow=24:DIM A$(maxrow,80)
200 FILE$=''
210 row=0:command:row=1
220 :
230 REMark main program loop
240 :
250 REPeat EDdy
260   pcur 0:PRINT #3,'> ';
270   EDLIN #3,A$(row),termX
280   pcur 0:PRINT #3,'  '
290   IF termX=208 THEN row=row-1: ELSE row=row+1:REMark test for up arrow
300   IF row=0 THEN command:REMark if at top line then do command
310   IF row=(maxrow+1) THEN row =maxrow:REMark if at bottom then stay there
320 END REPeat EDdy
330 :
340 DEFine PROCedure command
350   pcur 0:PRINT #3,'??'
360   BGET #3,a:IF a=216 THEN
370     pcur 0:PRINT #3,'  ':row=1
380     RETurn
390   END IF
400   IF a=115 OR a=83 THEN fsave :        REMark Test for character = 'S'
410   IF a=113 OR a=81 THEN CLOSE #3:STOP:REMark Test for quit
420   IF a=108 OR a=76 THEN fload:         REMark test for character = 'L'
430   IF a=90 OR a=122 THEN zap:           REMark Test for character = 'Z'
440   GO TO 350
450 END DEFine
460 :
470 REMark file save command
480 :
490 DEFine PROCedure fsave
500 open_file 6,3
510 IF ferr<>0 THEN GO TO 540
520 FOR rec=1 TO maxrow:PRINT #6,A$(rec):NEXT rec
530 CLOSE #6
540 pcur 2:CLS #3,4
550 END DEFine
560 :
570 REMark load a file command
580 :
590 DEFine PROCedure fload
600 open_file 6,1
610 IF ferr<>0 THEN pcur 2:CLS #3,4:RETurn
620 FOR rec=1 TO maxrow:INPUT #6,A$(rec):NEXT rec
630 CLOSE #6
640 CLS #3:FOR rrow=1 TO maxrow:AT #3,rrow,2:PRINT #3,A$(rrow):NEXT rrow
650 END DEFine
660 :
670 REMark general file open
680 :
690 DEFine PROCedure open_file(ch,opcode)
700   pcur 2:INK #3,4:PRINT #3,' Enter filename ':INK #3,7:EDLIN #3,FILE$
710   IF LEN(FILE$)=0 THEN ferr=-1:RETurn
720   ferr=FOPEN(#ch,opcode,FILE$) :REMark actually open the file
730   IF ferr<>0 THEN
740   pcur 2:CLS #3,4
750   pcur 3:FAULT #3,ferr:PAUSE 150:GO TO 700:REMark if error then print it
760   END IF
770 END DEFine
780 :
790 DEFine PROCedure pcur(x)
800 AT #3,row,x
810 END DEFine
820 :
830 DEFine PROCedure zap
840   FOR rrow=1 TO maxrow:A$(rrow)='':NEXT rrow:CLS #3
850 END DEFine zap
860 :
```