!PR.MODE

!pr.mode,<mode 0-1>,<window>

Selects mode for printing in selected <window>.

0 = Opaque
1 = Transparent

!FLUSH

!flush

When called, this will clear the key-board buffer totaly, eliminating any unwanted key presses that would normally continue to be actioned in succession. Use before reading the key-board to obtain only the key required by the input routine.

!SAVE.SCREEN

!save.screen,<0-1>

This command will save an entire screen in any mode to the cassette in 1 block, instead of the usual 8 x 2k blocks.
The block is saved without a header so must be used with caution!

0 = cassette message will not be displayed
1 = cassette message will be displayed

The block can only be re-loaded with !LOAD.SCREEN.

!LOAD.SCREEN

!load.screen,<0-1>

This command will load a screen previously saved with !SAVE.SCREEN.

0 = cassette message will not be displayed
1 = cassette message will be displayed

MODE and PEN inks must be set to those when the screen was saved with !SAVE.SCREEN, else the screen block will not appear the same.

SIREN SOFTWARE

X-BASIC

Extended Basic System

AMSTRAD CPC464/664 and 6128

C O N T E N T S

SIREN SOFTWARE
76 BRIDGE STREET
MANCHESTER M3 2RJ
TELEPHONE 061-796 6874

DISCLAIMER

Whilst every care has been taken to ensure that the information given here is accurate and complete, Cresent Soft makes no representation or warranties with respect to the contents, and specifically disclaims any implied warrenties, merchantability or fitness for a particular purpose.

The right is also reserved to amend this publication from time to time, without obligation to notify any person or organisation of such amendment.

X-BASIC LOADING INSTRUCTIONS

1 : X-BASIC must be loaded into an area of free ram within the central 32k, (that is within 16384 (&4000) and ramtop-2870 (&0B36)) and initialised with 'CALL addr,<0-1>'

eg.

```
10 MEMORY &7000-1
20 LOAD "X-BASIC.BIN",&7000
30 CALL &7000,<0 or 1>

<0> = Title will not be displayed
<1> = Title will be displayed
```

2 : For X-BASIC to run correctly, during its life with the host program, a clean version of the code must be made. This is achieved thus :

a : Load and run X-BASIC as previously described, write and debug your basic program.

b : Enter as the first line in your program,..

```
10 MEMORY addr-1: load "X-BASIC.BIN",addr: CALL addr,<0-1>
   (addr = Any FREE RAM in the central 32k.)
```

c : Make a copy of your basic program and verify it with CAT.

d : RESET THE COMPUTER

e : Type MEMORY &7000-1 then press ENTER

f : Type LOAD "X-BASIC.BIN",&7000 then press ENTER

g : Insert cassette containing your basic program and wind to the end of the program. Again use CAT.

h : Type SAVE "X-BASIC.BIN",b,&7000,&B35 then press RETURN and follow the cassette instructions.

From now on your program will load, relocate and log-on X-BASIC.

GENERAL NOTES ON THE USE OF X-BASIC

The purpose of this set of RSX extensions is to provide some of those facilities missing from Locomotive BASIC. A total of 58 new basic commands are included which are all accessed by the 'BAR' prefix, ie. the shifted '@' key, which produces a vertical bar on the screen, vis. '|'.

Each new command will be detailed separately, followed by the syntax required, an example, and where neccessary brief notes pertaining to it's use.

For the purpose of brevity in the command description the following conventions will be used :

@a%	:	any PRE-DEFINED integer variable
n	:	any numeric variable (reals would be truncated to integer)
nn	:	any 16 bit number in the range 0 - 65535 (this can be in Hex, with a preceding '&')
dd	:	any ram address, decimal or hex values

All other conventions in the use of brackets apply as usual.

If you are using SYMBOL AFTER to redifine characters then insert this command as the first line in your program, so to precede any line that alters HIMEM.

When a program, or routine, is to use any of the screen manipulating commands, it is advisable that the screen is previously 'set' (with mode <0,1 or 2>), otherwise there may be some confusion between X-BASIC and the firmware, resulting in a system crash.

It is strongly advised that you list and inspect the demonstration program to obtain the correct method of use for X-BASIC'S 58 commands, and practice with their characteristics until well known before including them within your basic program.

Full error checking has been included within X-BASIC, and any parameter errors will output the message :-

PARAMETER ERROR
PRESS <ESC>

Unless you are in direct command mode you must press <ESC> otherwise unpredictable results will arise.

The BASIC interpreter will then stop, and display the message :-

'Break in <n>'

Where <n> is the number of the line containing the command following the X-BASIC parameter error.

Any other errors will be dealt with by the editor or the BASIC interpreter.

All messages returned by X-BASIC will appear in window £7. However the position of this window has not been determined by X-BASIC and must be set up by the host program. It may of course be left as the default whole screen determined by the firmware, but this would cause the screen to clear each time a message had to be displayed. It is therefore recommended that window £7 is created, at least 30 characters wide by 3 lines deep, early in the host program, wherever the programmer feels is convenient.

!FRAME

!frame

Waits for the frame fly-back signal before commencing with next operation, eliminating flicker and making scrolling and animation smoother.
eg.

```
10 MODE 1
20 FOR a=1 to 10
30 |FRAME
40 |LINE.LEFT,10
50 REM alternative position for |frame
60 NEXT
70 END
```

The actual position of |FRAME will depend on what is to follow and only trial and error will find the ideal location.

|LINE.W.LEFT

|line.w.left,<wrap 0-1>,<No. of chars 1-80>,<col 1-80 >,<row 1-25>

Scrolls any part of any line left, <No of chars> is always in the range of 1-80 regardless of the present mode, so to move a whole character 1 position in mode 2 will require |LINE.W.LEFT to be called once, twice in mode 1 and 4 times in mode 0.

<wrap> 0 = no wrap around
1 = wrap around

|LINE.W.RIGHT

|line.w.right,<wrap 0-1>,<No of chars 1-80>,<col 1-80>,<row 1-25>

Scrolls any part of any line right.

As |LINE.W.LEFT

LINE.LEFT

|line.left,<list of line Numbers seperated by a ','>

Scrolls each line listed, left, 1 character position with wrap-around. Line No's may be duplicated so to scroll the same line more than once. 80 column mode is assumed at all times, so to scroll left 1 character in mode 1, |LINE.LEFT will need to be called twice or duplicate the line No.
eg.

```
10 MODE 1
20 PRINT "X-BASIC"
30 |LINE.LEFT,1,1: rem line number called twice
40 END
```

LINE.RIGHT

|line.right,<list of line No seperated by a ','>

As |LINE.LEFT but movement is to the right.

!WINDOW.UP

!win dow.up,<left col>,<right col>,<top row>,<bottom row>,<pen>,<n>

Scrolls a defined window up <n> rows filling the bottom row of the window with the selected <pen>ink.
Has no wrap around'

!WINDOW.DOWN

!window.down,<left col>,<right col>,<top row>,<bottom row>,<pen>,<n>

As !WIN.UP but movement is down and the top row is filled.

!SCROLL.UP

!scroll.up, pen , n

Scrolls the WHOLE screen up <n> lines filling the bottom line with the selected pen ink.
This uses hardware to do the scrolling and must not be used in conjunction with any other scroll command, a system crash may result!

!SCROLL.DOWN

!scroll.down, pen , n

Scrolls WHOLE screen down <n> lines filling the top line with the selected <pen>ink.

As !SCROLL.UP but movement is down.

!SCROLL.LEFT

!scroll.left,<chars 1-80>

Scrolls WHOLE screen left (1-80) character positions.

As !SCROLL.UP but movement is left.

!SCROLL.RIGHT

!scroll.right,<chars 1-80>

Scrolls WHOLE screen right <1-80> character positions.

As !SCROLL.UP but movement is right.

!GET.KEY

!get.key,<@a%>

Returns in the variable a% the ASCII code of the key pressed, if a key was pressed when the test was made.
This command does not wait for a key to be pressed.
eg.

```
10 MODE 1
20 a%=0: !TIME.0
30 WHILE TIME<1000
40 !GET.KEY,@a%
50 IF a%=32 then PRINT "-YOU PRESSED THE SPACE BAR-"
60 WEND
70 !CONTROL,7: !PAUSE,100: !CONTROL,7
80 END
```

!WAIT.KEY

!wait.key,<@a%>

As !GET.KEY but waits until a key is pressed before returning.
eg.

```
10 MODE 1
20 a%=0
30 PRINT "PRESS <SPACE BAR>"
40 !WAIT.KEY,@a%
50 PRINT a%
60 !CONTROL,7: !PAUSE,100: !CONTROL,7
70 END
```

!GET.CURSOR

!get.cursor,<window 0-7>,<col>,<row>,<@a%>

Returns, in the variable a%, the screen address of the top left corner of a character position. Used to find the address for POKEing values directly to a screen character position.
Also used in conjunction with !ENCODE. (qv)
eg.

```
10 MODE 1
20 a%=0: b%=0
30 !ENCODE,2,@b%
40 !GET.CUR,10,10,@a%
50 FOR x=a% to a%+16384 STEP 2048
60 !PAUSE,100
70 POKE x,b%
80 NEXT
90 END
```

As the address returned in the variable a% is an integer value, 65536 will need to be added to a% if you wish to print the number on the screen.

!GET.POINT

!get.point,<x co-ord>,<y co-ord>,<@a%>

As !GET.CUR but returns the screen address of an x,y point.

!READ.CHAR

!read.char,<window 0-7),<@a%)

Returns, in the variable a%, the ASCII code of a character on the screen at the current cursor position, or returns 0 if the character is not identifiable. ie. A line drawing.

eg.

```
10 a%=0
20 LOCATE 10,10: PRINT "AbCdEf...G!"
30 FOR x=10 to 20
40 LOCATE x,10: !READ.CHAR,@a%
50 LOCATE x,11: PRINT CHR$(a%)
60 !PAUSE,30
70 NEXT
80 END
```

!DEEK

!deek,<dd),<@a%)

Used in the same way as PEEK, but returns a 16 bit number in the variable a% calculated from the address dd and dd+1.

!DOKE

!doke,<dd),<nn)

Used in the same way as POKE, but passes a 16 bit number nn, into the address dd and dd+1.

!GR.MODE

!gr.mode,<0-3)

Selects graphics write mode.

0 = FORCED		new ink replaces old ink
1 = XOR		new ink xored with old ink
2 = AND		new ink and old ink
3 = OR		new ink or old ink

!GR.PAPER

!gr.paper,<0-15)

Sets graphics paper. ie. background ink.

!GR.PEN

!gr.pen,<0-15)

Sets graphics pen without having to plot a point.

!SLOW

!slow,<n)

Allows you to slow down the operaton of any program (including programs written in code). Any value may be selected to gain effect, help de-bug a program or even slow down listing. Selecting !SLOW,&2000 will be of most use as this catches the frame fly-back signal thus making operation very smooth.

eg.

```
10 !SLOW,&2000
20 CLS
30 !FAST
40 END
```

To switch off the interrupt use !FAST. !SLOW,0 will have the same effect.

!FAST

!fast

Used to switch off !SLOW.

!EXPAND.FUNC

!expand.func,<length 44 to 8000),<address 16384 to 49152)

Allows you to expand the function key buffer from its default setting. Up to 255 characters per f/key is possible.

<address). MUST lie within the central 32k. area of ram and care should be taken not to overwrite any existing code/data.

<length). MUST be no less than 44. A figure higher than 8000 would be of no use.

eg.

```
10 !EXPAND.FUNC,1000,&6000
20 KEY 0,"type in 1 to 255 chrs"
30 REM run & then press key <0) on the key pad
```

!FILL.BOX

!fill.box,<left),<right),<top),<bottom),<pen)

Fills a character area with a specified pen.

<left) & <right) = 1 to 80
<top) & <bottom) = 1 to 25
<pen) = 0 to 15

eg.

```
10 MODE 1
20 !FILL.BOX,5,35,2,23,3
30 END'
```

!BIG

!big,<col>,<n>,<row>

This produces <n> double height characters from <col> on <row>. <col> & <n> assume 80 characters per line in all modes.

eg.

```
10 MODE 0
20 PRINT "AMSTRAD"
30 !BIG,1,28,1
40 !pause,600
50 END
```

!INVERT

!invert,<col>,<row>,<pen1>,<pen2>,<n>

Inverts <n> characters on <row> from <col> using <pen1> & <pen2>. A second call of !INVERT using the same parameter will revert to original mode.

eg.

```
10 MODE 1: BORDER 0: INK 0,0: PEN 5
20 LOCATE 3,10: PRINT"INVERT": LOCATE 12,10: PRINT"INVERT"
30 FOR f%=15 TO 5 STEP-1
40 !INVERT,2,f%,a,b,8
50 FOR f%=5 TO 15
50 !INVERT,2,f%,a-1,b-1,8
60 NEXT
70 a=(a+1)MOD 16: b=a+1
80 GOTO 30
90 END
```

Useful when highlighting text in a menu or high score table.

!ENCODE

!encode,<pen>,<@a%>

Returns in the variable a% an encoded value of <pen> that will cover a whole character byte addressed on the screen, this may then be masked so that any single bit of the byte addressed may be set and poked to the screen.
Also used in conjunction with !GET.CUR.

!CONTROL

!control,<list of control or ASCII codes separated by a ','>

Allows simple use of control codes within a basic program. A maximum of 32 control or ASCII codes may be entered into 1 line removing the need to use multiple CHR$(?).

eg.

```
10 !CONTROL,12,7,31,8,10,14,3,15,1,72,69,76,76,79,7
```


!DRAW.ABS

!draw.abs,<list of:<x co-ord,y co-ord>>[,x co-ord,y co-ord]...

Allows a single line to contain up to 16 un-signed absolute x-y coordinates. Only pairs of coordinates will be accepted.

!DRAW.REL

!draw.rel,<list of:<x co-ord,y co-ord>>[,x co-ord,y co-ord]...

Allows a single line to contain up to 16 signed relative x-y coordinates. Only pairs of coordinates will be accepted.

eg.

```
10 MODE 1
20 !DRAW.REL,100,100,30,-20,200,100
30 END
```

!CAPS.ON

!caps.on

Switches on caps-lock facility from within a program.

!CAPS.OFF

!caps.off

Switches off caps-lock facility from within a program.

!CLEAR.WIN

!clear.win,<window 0-7>,<pen 0-15>

Clears the selected <window> with the selected <pen> but leaves the cursor position unchanged.

!PAUSE

!pause,<n>

An uninteruptable pause.

When <n>=300 the pause time is 1 second.

!WARM

!warm

Resets as much of the firmware & hardware that is possible without loosing your basic or m/code program.

!TIME.0

!time.0

Resets the internal clock to zero, allowing very accurate timing of loops ect.

eg.

```
10 !TIME.0
20 FOR f%=1 to 10000
30 NEXT
40 PRINT TIME
50 !CONTROL,7
60 END
```

!INSERT

!insert,<n>,<from>,<line No>

This provides the unusual facility of allowing the inclusion of machine code within basic REM lines, and must be used with caution. USE ONLY IN DIRECT MODE!

<n> = 1 to a MAX of 250 bytes.

<from> = any ram address. This is the address from where !INSERT will obtain the necessary code, it is assumed that you will load your short machine code routine into spare ram and pass the start address to !INSERT.

<line No> = any line number, but to prevent confusion by the firmware when listing and running a program it is advised that the last lines in a program are used, and the reserved word 'END' is used in the preceding line.

Once code has been inserted into a basic line then it may be called with !CALL.LINE, and from now on the code will be saved with the basic program.

eg.

Write your program :-

```
10 ' start of program
20 !CALL.LINE,1000
30 END
1000 REM ..... at least 3 '.'
```

Then issue a direct command !INSERT,3,&BB6C,1000

REM lines that will include code must have at least the number of '.' as the length of code that will be inserted before !INSERT is used, else a system crash may result.
DO NOT attempt to edit any line containing code, this will almost certainly cause a system crash, and when renumbering a program make the necessary change to the lines containing !CALL.LINE,? as they are not renumbered with the rest of the program.

!CALL.LINE

!call.line,<n>

Calls and runs the inline code previously set up by !INSERT. If the line number is not found then !CALL.LINE hands control back to basic.

!ENVA

!enva,<channel>,<type>,<duration>

Gives direct access to the AY-3-8912 programmable sound generator envalope registers. Once an envelope shape has been set-up it will continue to run until cancelled by SOUND 7,0,0,0.

<channel> = 1-2-3

<type> = envelope shape
8 = Saw tooth
9 = Jump up and ramp down holding zero volume
10 = Jump up then repeatedly ramp down and up again
11 = Jump up, ramp down then jump up and hold maximum volume
12 = Repeated ramp up and drop down
13 = Ramp up and hold maximum volume
14 = Repeated ramp up and down
15 = Ramp up and drop down holding zero volume

<duration> = Period between rise and fall of ramps (in the range 0 to 65536)

eg.

```
10 FOR f=200 to 300 STEP 5
20 !ENVA,1,14,f: SOUND 2,50,10
30 !PAUSE,200
40 NEXT
50 !PAUSE,600
60 SOUND 7,0,0,0
70 END
```

MOVE.MEM

!move.mem,<n>,<from>,<to>

Moves a block of memory regardless of rom state.

<n> = 0 to 65535
<from> = any ram address
<to> = any ram address

eg.

```
10 MODE 1
20 FOR f=2 to 20
30 LOCATE 1,f
40 PRINT STRING$(30,"*")
50 NEXT
60 FOR f=1 to 10
70 !MOVE.MEM,16384,&C000,&c001
80 NEXT
90 END
```

!CURSOR.ON

!cusor.on,<window 0-7>

Enables the systea cursor.

!CURSOR.OFF

!cursor.off,<window 0-7>

Disables the sytea cursor when it would normally be displayed.

eg.

```
10 !CURSOR.OFF
20 INPUT "Cursor now off   press <ENTER> ";a$
30 !CURSOR.ON
40 INPUT "Cursor now on    press <ENTER> ";a$
50 END
```

!RESET.OFF

!reset.off

Disables both the <ESC> key and soft reset <CTRL>/<SHIFT> <ESC>. Therefore totaly protecting your program from interupt or intrusion.

!RESET.ON

!reset.on

Allows you the facility of enabling the soft reset to gain access to a program that had previously been protected with !RESET.OFF.

!BREAK.OFF

!break.off

Disables the <ESC> key, giving partial protection from interuption or intrusion.
The soft reset is still enabled.

eg.

```
10 !BREAK.OFF
20 PRINT "PRESS <ESC> NOW"
30 !PAUSE,600
40 PRINT "<ESC> NOW ACTIVE, TRY AGAIN"
50 !BREAK.ON
60 !PAUSE,900
70 END
```

!BREAK.ON

!break.on

Enables the <ESC> key when previously disabled by !BREAK.OFF.

!MOTOR.ON

!motor.on

Switches on the cassette motor if not already on and pauses momentarily until the motor gains speed.

!MOTOR.OFF

!motor.off

Switches off the cassette motor if not already off.

!SHIFT.ON

!shift.on

Switches on the shift lock facility from within a program.

!SHIFT.OFF

!shift.off

Switches off the shift lock facility from within a program.

!PUT.CURSOR

!put.cursor,<window 0-7>

Displays the system cursor at the current position set by locate. The cursor must be removed with !POP.CURSOR before altering the cursor position, else the cursor will continue to be displayed at the old location.

!POP.CURSOR

!pop.cursor,<window 0-7>

Used to remove a cursor square previously positioned by !PUT.CURSOR.

!INVERSE

!inverse,<window 0-7>

Switches currrent text printing to inverse video in the selected window. A second call to !INVERSE will revert to original mode.