

U64 BBC BASIC 1.0.0

User manual

Contents

1. Introduction	2
1.1 Help on the machine	3
1.2 Compared with a BBC Micro	3
1.3 Credits	3
2. Installation	3
2.1 What you need	3
2.2 The cartridges	3
2.3 Installing with make deploy	4
2.4 Starting	4
3. Building from source	4
3.1 Requirements	4
3.2 Building	4
3.3 Where things are	5
4. Starting	5
4.1 A first program	5
4.2 Saving and loading	5
4.3 The keyboard	5
4.4 Help	6
5. Typing and editing lines	6
5.1 The keys at a glance (BBC editor)	6
5.2 Typing a line	7
5.3 Correcting while you type	7
5.4 Earlier lines (history)	7
5.5 Editing a listed program line: copy editing	8
5.6 BASIC's own editing commands	8
5.7 Screen modes	9
5.8 The C64 editor (*EDIT C64)	9
5.9 Good to know	9
5.10 Reference	9
6. BASIC	10
6.1 Variables and operators	10
6.2 Working with programs	11
6.3 Program flow	13
6.4 Procedures and functions	14
6.5 Errors	15
6.6 Numbers	16
6.7 Strings	19
6.8 Data in the program	20
6.9 Keyboard and text on the screen	20
6.10 Time	22
6.11 Machine code	22

7. Files	23
7.1 The filing system	23
7.2 Directories and files	24
7.3 Files from programs: channels	26
7.4 Data in the REU	27
8. Graphics	28
8.1 Screen modes	28
8.2 Drawing	29
8.3 VDU codes	31
8.4 Colours, palettes, fonts and CharPad maps	32
8.5 Sprites	33
8.6 Upic full: 384 x 256 in 16 colours	33
8.7 BASIC V graphics commands	34
8.8 GXR software sprites	35
8.9 The sprite editor	35
9. Sound	36
9.1 Notes and envelopes	36
9.2 The sound engine	37
9.3 Instruments and songs	38
Ultimate Audio: samples and MOD music	39
10. Network	39
11. Ultimate commands	40
11.1 Overview	41
11.2 Drives and disk images	42
11.3 Clock and speed	42
12. Memory	43
12.1 Snapshots	44
12.2 Strings and memory	45
12.3 The REU	45
13. MOS calls	45
14. Developing from a PC	47
15. Limits	47

1. Introduction

U64 BBC BASIC is BBC BASIC 3.10, the BASIC of the Acorn BBC Micro, as a cartridge for the Ultimate 64 and the C64 Ultimate. The interpreter is the original, unchanged (built from Ivo van Poorten's annotated source, checked byte for byte against the reference ROM). Around it sits a BBC-style MOS (machine operating system), written for the C64 and the Ultimate: the screen and its modes, the keyboard, sound on the SIDs, files on the Ultimate's drives, the network, the REU.

The aim is that BBC BASIC *programs* run: the language, and what the VDU, graphics, sound and file statements do. It is not a BBC Micro emulator: the memory map, the BBC's hardware and machine code are not reproduced.

What it adds for this machine: the Ultimate's turbo (BASIC runs many times faster than on a BBC Micro), up to 24 SID voices with instruments and a song player, the REU as storage and as a RAM disk, network connections, and * commands for the Ultimate's drives, clock and settings.

1.1 Help on the machine

Everything in this manual's reference sections is also on the machine: *HELP and a keyword, a command or a topic.

*HELP PRINT
*HELP *CAT
*HELP KEYWORDS

1.2 Compared with a BBC Micro

BBC (also COMPATIBILITY) BBC BASIC programs run: the BBC BASIC 3.10 language and the behaviour of the VDU, graphics, sound and file statements.

This is not a BBC Micro emulator. MODE 7's screen memory is at &7C00 as on the BBC, so programs that write teletext codes there work; the other modes keep their pictures elsewhere (see MODE and MEMORY). The BBC hardware at &FE00, the MOS internals and BBC machine code are not there: programs that poke the BBC's hardware, or CALL machine code written for the BBC, do not work.

The MOS calls a program can use are those in *HELP OSBYTE*, *HELP OSWORD* and *HELP MEMORY.

1.3 Credits

CREDITS BBC BASIC (C) 1982/1983 Acorn Computers and Sophie Wilson.

Annotated source: Ivo van Poorten (github.com/ivop/bbc-basic), based on J.G. Harston's reconstruction (mdfs.net).

BBC Micro characters (*FONT BBC): (C) 1981/1982 Acorn Computers, from the MOS 1.20 ROM.

MOS: Xander Mol, GPL-3. Ultimate code ported from ultimate-uci-oscar64 (Scott Hutter, Francesco Sblendorio, Christian Gleissner, Xander Mol).

github.com/xahmol/u64-bbcbasic

2. Installation

2.1 What you need

An Ultimate 64 (Elite, Elite II) with firmware 3.15a or later (the network, palette and SID settings need it), with an REU enabled (16 MB is best: the RAM disk, songs and the line editor's history use it). Upic Lite graphics (MODE 1, 2, 4, 5, 6 with square pixels) need a 64 MHz machine (Ultimate 64 Elite II, C64 Ultimate).

The **Commodore 64 Ultimate** is not fully supported yet: U64 BBC BASIC relies on features of the Ultimate 64 firmware 3.15a, which Commodore has not yet released in a C64 Ultimate firmware update.

2.2 The cartridges

File	Type	For
u64bbc-tmb.crt	Protovision TwoMegabyter (CRT type 87)	the Ultimate 64 / C64 Ultimate: the Command Interface, REU, Ultimate Audio and turbo all usable

File	Type	For
u64bbc-ef.crt	EasyFlash, subtype 1	VICE, and as a fallback (no Ultimate Audio; the firmware moves the Command Interface to \$DE1C)

Copy the .cfg with the same name next to the .crt. Started from the Ultimate's file browser, the cartridge loads it: the Command Interface, the REU, the Ultimate Audio mapping and the U64 turbo registers are switched on, and 8 UltiSIDs are set up for sound (UltiSID 1 at \$D400, UltiSID 2 at \$D420, each split four ways). The settings are loaded into memory only, not saved to flash. Started another way, *SID SETUP sets up the SIDs.

2.3 Installing with make deploy

From a PC with the source (chapter 3), `make deploy` puts the cartridges, their .cfg files and all demos on the Ultimate over FTP:

```
/<drive>/idi8b/u64bbcbasic/      the cartridges and .cfg files
/<drive>/idi8b/u64bbcbasic/demos/ classics, music, net, sound, ...
```

The Ultimates' addresses and drives come from .env in the source folder (ULTIP1 = 192.168.1.148, ULTDRV1 = SD; a second Ultimate as ULTIP2, ULTDRV2). By hand: copy the same files with the Ultimate's FTP server or a USB stick.

2.4 Starting

Select u64bbc-tmb.crt in the Ultimate's file browser and run it. The machine starts in the C64 text screen at BBC BASIC's prompt:

```
U64 BBC BASIC 1.0.0
BASIC 3.10 (C)1983 Acorn
CPU turbo (maximum)
>
```

The demos: *DIR into a demo folder and CHAIN a program, e.g.

```
*DIR /SD/idi8b/u64bbcbasic/demos/music
CHAIN "MUSIC"
```

3. Building from source

3.1 Requirements

[MADS 2.1.x](#) (built with FreePascal: `fpc -Mdelphi -vh -O3 mads.pas`), Python 3, GNU make, patch. For the tests: VICE (x64sc) and Node.js (npm, for c64bridge).

3.2 Building

```
git clone --recurse-submodules https://github.com/xahmol/u64-bbcbasic
cd u64-bbcbasic
make          # build/u64bbc-tmb.crt, build/u64bbc-ef.crt and .cfg
make compare-core # BBC BASIC 3.10 still identical to the reference ROM
make vice      # run in VICE
make e2e       # end-to-end tests (tests/e2e/README.md)
make deploy    # install on the Ultimate (chapter 2)
make manual    # this manual, from docs/manual/src and the help texts
```

The interpreter is never changed: lib/bbc-basic is Ivo van Poorten's source as a submodule, and the only difference is the build target (src/basic/u64-target.patch); `make compare-core` checks it.

3.3 Where things are

Path	Contents
src/mos/	the MOS in 6502 assembler (MADS): boot, VDU, keyboard, filing system, sound, network, line editor
src/help/help.txt	the *HELP texts (also this manual's reference sections)
tools/	the cartridge builder, tokeniser, help packer, remote tools
demos/	the demo programs
tests/e2e/	the end-to-end tests (VICE and the Ultimates)
docs/	design documents, the plan, this manual

4. Starting

4.1 A first program

```
>10 FOR I%=1 TO 10
>20   PRINT I%, I%*I%
>30 NEXT
>RUN
```

Lines with a number go into the program; lines without run at once. LIST shows the program, NEW clears it. How lines are typed and edited: chapter 5.

4.2 Saving and loading

Programs are saved in the current directory of the Ultimate's file system (chapter 7):

```
*DIR /SD/myprogs
SAVE "SQUARES"
LOAD "SQUARES"
CHAIN "SQUARES"
```

4.3 The keyboard

KEYBOARD Letters type in capitals, as with CAPS LOCK on a BBC Micro; Shift gives lower case. Shift+C= switches CAPS LOCK off and on again: off, letters type in lower case and Shift gives capitals (BASIC's keywords are capitals). OSBYTE &CA reads and sets it (bit 4 = 1: off), as on the BBC.

The ASCII characters the C64 keyboard lacks:

\	pound key	^	up-arrow
_	left-arrow		Shift+pound
~	Shift+up-arrow	{	Shift++
}	Shift+-	`	Shift+@

RUN/STOP is ESCAPE: it stops a program or a listing. RESTORE also sets ESCAPE.

Lines are typed with the MOS line editor (*EDIT), in every screen mode, up to 238 characters:

LEFT, RIGHT	move in the line
CLR/HOME	start of the line
Shift+CLR/HOME	end of the line
DEL, INST	delete, insert space
UP, DOWN	earlier lines typed
CTRL+cursor	move the copy cursor
F1	COPY

RETURN enter the line

COPY editing as on a BBC Micro: CTRL with a cursor key starts a second cursor and moves it over the screen; F1 copies the character under it into the line and moves it on, so a listed line can be copied, in every mode (in the bitmap modes the character is recognised from its pixels, as on a BBC Micro; the 4-pixel modes 0, 1 and 3 need the REU's 4x6 font for that). UP and DOWN go through the last 127 lines typed at the prompt (kept in the REU). *EDIT C64 brings back the C64 screen editor (move the cursor anywhere and press RETURN; 78 characters after the prompt).

GET and INKEY return the cursor keys as 136-139 (left, right, down, up), DEL as 127 and CLR as 12.

4.4 Help

*HELP

*HELP [subject]

Shows help: see HELP.

5. Typing and editing lines

Everything you type at the > prompt, and every answer to an INPUT in a program, goes through a **line editor**. U64 BBC BASIC has two:

Editor	Switch on with	In short
BBC (the default)	*EDIT BBC	Lines up to 238 characters, edited in place, earlier lines on the cursor keys, BBC Micro copy editing. Works the same in every screen mode.
C64	*EDIT C64	The C64's own full-screen editor: move the cursor anywhere on the screen and press RETURN to enter that line. Lines up to 78 characters after the prompt.

*EDIT on its own tells you which one is in use. The choice holds until the machine is restarted. Then it is BBC again.

5.1 The keys at a glance (BBC editor)

The C64 has two cursor keys: CRSR ↵ and CRSR ⇐. With SHIFT they go the other way (up and left).

Key on the C64	What it does
letters, digits, signs	typed at the cursor; the rest of the line moves right
CRSR ⇐ / SHIFT+CRSR ⇐	cursor right / left in the line
CLR/HOME	to the start of the line
SHIFT+CLR/HOME	to the end of the line
INST/DEL	delete the character left of the cursor
SHIFT+INST/DEL (INST)	insert a space at the cursor
SHIFT+CRSR ↵ (up)	the line you entered before (history)

Key on the C64	What it does
CRSR ⌵ (down)	back towards the newest line, then the line you were typing
CTRL+CRSR ⇌ / CTRL+SHIFT+CRSR ⇌	copy cursor right / left
CTRL+CRSR ⌵ / CTRL+SHIFT+CRSR ⌴	copy cursor down / up
F1	COPY: the character under the copy cursor; on an empty line with no copy cursor, the file browser (*BROWSE)
RETURN	enter the whole line, wherever the cursor is in it
RUN/STOP	ESCAPE: forget the line

Letters type as capitals, as with CAPS LOCK on a BBC Micro. Hold SHIFT for lower case. SHIFT+C= switches CAPS LOCK off (letters in lower case, SHIFT for capitals) and on again. BASIC's keywords must be typed in capitals, so CAPS LOCK off is for text in strings and REMs. Programs read and set it with OSBYTE &CA (bit 4: 1 = off), as on the BBC. The ASCII signs the C64 keyboard does not have are listed under *HELP KEYBOARD, for example \ on the pound key and _ on the left-arrow key.

5.2 Typing a line

Type, and press RETURN:

```
>PRINT "Hello"
Hello
>
```

A line longer than the screen is wide simply continues on the next row. At the prompt you can type up to 238 characters, as on a BBC Micro; at the limit the machine beeps and takes no more.

A line with a number in front goes into the program instead of being run at once:

```
>10 PRINT "Hello"
>20 GOTO 10
```

Typing a line number that already exists replaces that line. Typing only the number deletes it.

5.3 Correcting while you type

You do not have to delete back to a mistake. Move to it:

1. Type PRINT 1+3.
2. You meant 12+3. Press SHIFT+CRSR ⇌ twice: the cursor is now on the +.
3. Type 2. The line reads PRINT 12+3; the rest moved right.
4. Press RETURN. The cursor does not need to be at the end.

```
>PRINT 12+3
      15
```

To remove characters, put the cursor just after them and press INST/DEL. CLR/HOME and SHIFT+CLR/HOME jump to the start and the end of the line.

5.4 Earlier lines (history)

Every line you enter at the > prompt is remembered: the last 127 of them.

- SHIFT+CRSR ⌴ (up) replaces the line you are typing with the previous one. Press again to go further back.
- CRSR ⌵ (down) goes forward again. One step past the newest line you get back what you were typing before you started going back.

- The line that appears can be edited like any other before you press RETURN.

Not remembered: empty lines, a line that is the same as the one before it, and answers to INPUT in programs. In an INPUT you can still fetch lines from the history.

The history is kept in the REU (bank 0, \$8000-\$FFFF, see [Memory](#)). It survives a cartridge restart and is lost when the machine is switched off. Without an REU there is no history; the rest of the editor works as described.

5.5 Editing a listed program line: copy editing

This is the BBC Micro way to change a line that is on the screen, for example after LIST. You build a new line by copying characters from the old one with a second cursor, the **copy cursor**, and type your changes in between.

Say the program has a mistake in line 30:

```
>LIST
 10 FOR I%=1 TO 10
 20 PRINT I%*I%
 30 NEXT J%
>
```

The cursor is on the empty prompt line. To make line 30 read NEXT I%:

1. Hold CTRL and press SHIFT+CRSR ⌵ once. A second cursor appears one row up, on the line 30 NEXT J%, in the column where your cursor was. It is shown as an inverted character.
2. Still with CTRL, press CRSR ⇐ twice: the copy cursor is now on the 3 of 30.
3. Press F1 (COPY) once for each character to copy (or hold it: it repeats). 30 NEXT is eight presses (the leading spaces of the listing need not be copied: BASIC ignores them). Each press copies the character under the copy cursor into your line and moves the copy cursor one step right.
4. The copy cursor is now on the J. Do not copy it. Type I instead; the copy cursor stays where it is.
5. Skip the J in the old line: CTRL+CRSR ⇐ once.
6. Press F1 once more for the %.
7. Press RETURN. Your new line 30 NEXT I% replaces the old line 30, and the copy cursor disappears.

With LIST0 switched on, the listing has extra spaces (after the line number and in front of indented lines). Copy them along: BASIC does not mind spaces there, it only takes more presses of F1.

The rules, in short:

- **Copy** with F1.
- **Change** by typing. Typing adds to your line and leaves the copy cursor alone.
- **Leave out** by moving the copy cursor past it with CTRL+cursor keys.
- **Insert** extra text by typing it before you copy on.

You can copy from anywhere on the screen, and from several places into one line. Copying works on the text screen (MODE 100, MODE 6 with the VIC-II) and in teletext MODE 7. The graphics modes keep only pixels, not characters, so there F1 copies nothing.

For a line you typed recently, the history is quicker: press SHIFT+CRSR ⌵ until it appears, correct it, press RETURN.

5.6 BASIC's own editing commands

Command	What it does
LIST, LIST 100,200	show the program, or part of it
*LIST	the program indented by its structure, keywords spelt out

Command	What it does
LIST 7	LIST indents loops from now on
AUTO, AUTO 100,5	line numbers typed for you (ESCAPE stops)
DELETE 100,200	delete a range of lines
RENUMBER, RENUMBER 1000,10	number the lines again, GOTOs and GOSUBs follow
NEW, OLD	clear the program; bring it back after an accidental NEW

See [BASIC keywords](#) for the details.

5.7 Screen modes

The BBC editor works the same in every mode. In the graphics modes and in MODE 7 it uses the mode's own text cursor and text window (VDU 28): a long line wraps at the window's right edge.

5.8 The C64 editor (*EDIT C64)

In the C64 text screen, *EDIT C64 gives you the editor of the C64 itself:

- Move the cursor anywhere on the screen with the cursor keys, change what is there, and press RETURN: the whole screen line under the cursor is entered. That is how a listed line is edited.
- INST/DEL, SHIFT+INST/DEL and CLR/HOME work as on a C64; SHIFT+CLR/HOME clears the screen.
- A line can be 78 characters long after the > prompt (two screen rows).
- No history, no copy cursor.

In the graphics modes and in MODE 7, *EDIT C64 means typing without editing: characters, INST/DEL for the last one, RETURN.

*EDIT BBC switches back.

5.9 Good to know

- **ESCAPE** (RUN/STOP) while typing forgets the line: BASIC answers Escape and gives a fresh prompt. RESTORE does the same.
- **Function keys:** *KEY gives F1-F8 a text to type (*KEY 2 LIST|M: F2 lists the program). F1 is COPY until *KEY 1 is set; on an empty line at the prompt it opens the file browser (*BROWSE, chapter 7).
- **Key repeat:** the cursor keys, space and INST/DEL repeat while held; CTRL with a cursor key and F1 (COPY) do too, after the same short delay: hold F1 to copy the rest of a line.
- **From a PC** (VS Code, tools/u64remote.py, the MCP server; see [Developing from a PC](#)): a line sent from the PC wins. A half-typed line on the machine is dropped from the screen, and the PC's line is run.
- GET and INKEY in your own programs see the cursor keys as 136-139 (left, right, down, up), INST/DEL as 127, CLR/HOME as 30 and SHIFT+CLR/HOME as 12. The editor is not involved there.

5.10 Reference

*EDIT

*EDIT BBC the MOS line editor
 *EDIT C64 the C64 screen editor
 *EDIT shows which one is used

BBC (the default): lines up to 238 characters in every screen mode, edited in place, with the last 127 lines on UP and DOWN and BBC copy editing (CTRL+cursor keys, F1 is COPY). See KEYBOARD.

C64: the KERNAL's full-screen editor in the text screen, as on a C64: move the cursor anywhere on the screen and press RETURN to enter that line; 78 characters after the prompt. In the graphics modes lines are typed without editing.

***KEY**

*KEY n text set function key n
*KEY n clear it
*KEY list the keys set

Function key Fn (1-8; F2, F4, F6, F8 are Shift with F1, F3, F5, F7) then types the text, at the prompt and in programs (GET, INKEY, INPUT), as on a BBC Micro. Codes in the text: |M RETURN, |A-|Z control codes, |? DELETE, ||, |" ", |! the next character + 128. Quotes around the text are optional.

*KEY 2 LIST|M
*KEY 3 "RUN|M"

F1 is the line editor's COPY until *KEY 1* is set. Up to 63 characters a key; keys 0 and 9 are kept (BBC programs set them) but have no key. The texts are kept in the REU. With EDIT C64 the keys type nothing.

6. BASIC

The language is BBC BASIC 3.10, unchanged. This chapter has the syntax of every keyword, grouped by what it does; the keywords for files, graphics and sound are in their own chapters (7, 8, 9). Keywords are written in capitals; A%-Z% are the fast resident integer variables.

6.1 Variables and operators

VARIABLES Names are letters, digits and _ and may be long. Types by suffix:

A real (5 bytes, ~9 digits)
A% integer (32 bits)
A\$ string (up to 255 chars)

The integers @% and A% to Z% are resident: they survive RUN, NEW and CLEAR. @% sets the PRINT format; A%, X%, Y% and C% pass registers to CALL and USR.

Arrays: DIM A(10), DIM N\$(5,5). Strings keep their space when a new value fits. Space that strings give up is only reclaimed by CLEAR or RUN.

OPERATORS Arithmetic: + - * / ^ (power), DIV and MOD (integers). Comparison: = <> < > <= >=, giving TRUE (-1) or FALSE (0). Logical/bitwise: AND OR EOR NOT.

Indirection reads and writes memory:

?A byte at address A
!A 4-byte integer at A
\$A string at A (CR ended)
A?2 byte at A+2
A!4 integer at A+4

Example:

DIM B% 10: ?B%=65: PRINT ?B%

Hexadecimal: &FF is 255; PRINT ~X prints X in hexadecimal.

LET

[LET] v=expr

Assignment; LET may be left out.

DIM

DIM A(n), B\$(x,y)

DIM P% n

Declares arrays, or reserves n+1 bytes of memory and puts the address in P%.

6.2 Working with programs

AUTO

AUTO [start[,step]]

Numbers new lines automatically, from 10 in steps of 10 by default. ESCAPE stops.

LIST

LIST [first][,last]

Lists the program. ESCAPE stops.

LISTO

LISTO n

Listing options: 1 space after the line number, 2 indent FOR loops, 4 indent REPEAT loops (add them up).

***LIST**

*LIST	all
*LIST n	line n
*LIST n,	from line n
*LIST ,m	up to line m
*LIST n,m	lines n to m

Lists the program indented by its structure: two spaces per level of DEF PROC ... ENDPROC, a DEF FN that ends with a line starting "=", FOR ... NEXT and REPEAT ... UNTIL (LISTO 7 indents only the loops). The program itself is not changed; spaces at the start of its lines are replaced by the indentation. ESCAPE stops.

DELETE

DELETE first,last

Deletes program lines first to last.

RENUMBER

RENUMBER [start[,step]]

Renumbers the program, GOTO and GOSUB targets included.

NEW

NEW

Removes the program (OLD brings it back if nothing was typed in between).

OLD

OLD

Brings back a program after NEW or BREAK.

RUN

RUN

Runs the program, clearing the variables.

CHAIN

CHAIN "name"

Loads and runs a program.

LOAD

LOAD "name"

Loads a program (see FILES). For memory dumps see *LOAD.

SAVE

SAVE "name"

Saves the program (see FILES). For memory see *SAVE.

CLEAR

CLEAR

Removes all variables except the resident integers @% and A%-Z%.

END

END

Ends the program.

STOP

STOP

Stops the program with a message.

REM

REM text

A remark; the rest of the line is ignored.

***SEARCH (also *FIND, FIND, *REPLACE, REPLACE, TOOLKIT)**

*SEARCH s1[,s2] (asks Parameters)

*FIND s1 list the lines

*REPLACE s1,s2 replace, asking

Searches the program, as Beebug's Toolkit Plus: every line holding s1 is listed. With s2, each match may be replaced (Replace Y/N:). *SEARCH then asks for Parameters, letters typed together: C case ignored, D not in DATA, R not in REM, " not in strings, K s1 and s2 are keywords, G replace all without asking. # in s1 matches any character. Without K, keywords are not found (they are stored as tokens); line numbers after GOTO are never matched. A replacement clears the variables, as editing does.

*FIND count
*SEARCH PRINT,REPORT (then K)

***XREF (also *VARLIST, VARLIST)**

*XREF [R I S A F P H L]
*VARLIST (as *XREF L)

Lists the variables, arrays, procedures and functions with their values and the lines that use them, as Beebug's Toolkit Plus. Letters choose: R reals, I integers, S strings, A arrays, F functions, P procedures (none: all); H values in hexadecimal, L no line numbers. Only what the program has made so far is known: run it first. A% to Z% are listed when the program uses them. Values are printed as PRINT would (@%).

***CRUNCH (also *PACK, PACK)**

*CRUNCH [C R S :] (also *PACK)

Makes the program shorter, as Beebug's Toolkit Plus: spaces, REM statements and extra colons go, and lines are joined. A letter switches that part off: C joining, R REMs, S spaces, : colons (B, L, T and V are accepted; brackets, LET, THEN and names are never changed). It asks first and prints the sizes; keep a copy. Lines named by GOTO, GOSUB or RESTORE keep their number (and their REM); lines are not joined after IF, ON, ELSE, DATA or a * command, nor are DEF and DATA lines joined on. Strings, DATA, * commands and assembler stay as they are. A computed GOTO's target is not known: use C there. Needs an REU.

6.3 Program flow

IF

IF cond [THEN] stmts [ELSE stmts]

Runs the statements when the condition is TRUE (not 0). The whole IF is one line.

THEN

IF cond THEN statements

Optional word in an IF line.

ELSE

IF cond THEN a ELSE b

The part of an IF line run when the condition is false. Also after ON ... GOTO.

FOR

FOR v=start TO end [STEP s]

...

NEXT [v]

Repeats the statements with v counting from start to end.

TO

FOR v=a TO b

The end value of a FOR loop.

STEP

FOR v=a TO b STEP s

The step of a FOR loop.

NEXT

NEXT [v]

Ends a FOR loop. See FOR.

REPEAT

REPEAT

...

UNTIL cond

Repeats until the condition is TRUE.

UNTIL

UNTIL cond

Ends a REPEAT loop.

GOTO

GOTO line

Continues at line.

GOSUB

GOSUB line

Calls the subroutine at line; RETURN comes back.

RETURN

RETURN

Ends a GOSUB subroutine.

ON

ON n GOTO l1,l2,... [ELSE ...]

ON n GOSUB l1,l2,...

Jumps to the n-th line in the list. See also ERROR.

6.4 Procedures and functions

Structured programs use DEF PROC and DEF FN with LOCAL variables instead of GOSUB; see the example programs in demos/.

DEF

DEF PROCname[(params)]

DEF FNname[(params)]=expr

Defines a procedure or function. See PROC, FN.

PROC

DEF PROCname(params)

...

ENDPROC

PROCname(args)

A named procedure with its own LOCAL variables.

ENDPROC

ENDPROC

Ends a procedure. See PROC.

FN

```
DEF FNname(params)=expr  
x=FNname(args)
```

A user-defined function; multi-line functions end with =value.

```
DEF FNsq(x)=x*x  
PRINT FNsq(7)
```

LOCAL

```
LOCAL v[,v...]
```

Makes variables local to a procedure or function.

6.5 Errors

ERRORS An error stops the program and prints a message; ON ERROR catches it. ERR gives the error number, ERL the line, REPORT prints the message.

Errors from the Ultimate keep the Ultimate's text with these numbers:

```
214 file not found  
189 access denied  
192 too many open files  
199 other Ultimate errors  
198 no room (LOAD too big)  
222 channel not open  
254 bad command  
252 bad number
```

ERROR

```
ON ERROR statements  
ON ERROR OFF
```

Statements to run when an error occurs. See ERRORS.

ERR

ERR

The number of the last error. See ERRORS.

ERL

ERL

The line number of the last error.

REPORT

REPORT

Prints the message of the last error.

TRACE

TRACE ON | TRACE OFF | TRACE n

Prints line numbers as they run (up to line n).

6.6 Numbers**ABS**

ABS(n)

The absolute value of n.

PRINT ABS(-3.5) gives 3.5

SGN

SGN(n)

-1, 0 or 1 by the sign of n.

INT

INT(n)

The largest whole number not above n.

MOD

a MOD b

The remainder of integer division.

DIV

a DIV b

Integer division.

PRINT 7 DIV 2 gives 3

SQR

SQR(n)

Square root.

EXP

EXP(n)

e to the power n.

LN

LN(n)

Natural logarithm.

LOG

LOG(n)

Logarithm base 10.

SIN

SIN(r)

Sine of r radians.

COS

COS(r)

Cosine of r radians.

TAN

TAN(r)

Tangent of r radians.

ASN

ASN(n)

Arc sine of n, in radians.

ACS

ACS(n)

Arc cosine of n, in radians.

ATN

ATN(n)

Arc tangent of n, in radians.

DEG

DEG(r)

Converts radians to degrees.

RAD

RAD(d)

Converts degrees to radians.

PI

PI

3.14159265.

RND

RND random integer

RND(1) 0 to 1

RND(n) 1 to n

RND(-n) sets the seed

Random numbers. The MOS seeds RND from real entropy at every start, so each start gives another sequence; *RANDOMIZE seeds it again. RND(-n) gives a reproducible sequence.

AND

a AND b

Bitwise AND of two integers; with comparisons a logical AND.

```
IF X>0 AND X<10 PRINT "OK"
```

OR

a OR b

Bitwise OR; with comparisons a logical OR.

EOR

a EOR b

Bitwise exclusive OR.

NOT

NOT n

Bitwise inversion; NOT TRUE is FALSE.

TRUE

TRUE

The value -1.

FALSE

FALSE

The value 0.

*TABLE

```
*TABLE SIN addr [steps]  sine table
*TABLE COS addr [steps]  cosine
*TABLE SQR addr [n]       square roots
*TABLE RECIP addr [n]     1/x
```

Fast sine and cosine for programs that need many of them (demos, generative graphics): a table of 1024 whole numbers, a full turn in 1024 steps, each the value times 65536, in 4096 bytes of the program's memory. A table read takes about 30 microseconds, SIN itself about 775.

```
DIM S% 4095:*TABLE SIN S%
Y%=S%!(4*(A% AND 1023))
```

gives $\text{SIN}(A\% \pi / 512) 65536$, rounded (65536 is 1). addr is a number or one of A%-Z%; the table must lie between PAGE and HIMEM. COS is the same a quarter turn on.

TABLE SIN addr 256 (or 512) makes a smaller table with a full turn in that many steps: 1K for 256, for modes that leave little memory (MODE 2 has under 5K). One table is enough for both: COS is $S\%!(4((A\%+64) \text{ AND } 255))$ with 256 steps.

TABLE SQR and TABLE RECIP hold n entries (256, 512, 1024 when left out, 2048 or 4096; 4 bytes each), entry i being $\text{SQR}(i) * 65536$ or $65536/i$, rounded (entry 0 of RECIP: &7FFFFFFF). A table read replaces SQR (about 700 microseconds) or a division of reals.

```
DIM Q% 4095:*TABLE SQR Q%
R%=Q%!(4*D%) DIV 65536
```

gives $\text{SQR}(D\%)$ for D% up to 1023.

***RANDOMIZE (also *RANDOMISE, RANDOMISE)**

***RANDOMIZE** (also ***RANDOMISE**)

Seeds RND afresh from real entropy: the time of day of the Ultimate's clock, the centisecond clock and the hardware timers at the moment of the call. The MOS does this at every start, so RND gives another sequence each time; call it in a program right after a key press for more. RND(-n) still starts a reproducible sequence.

6.7 Strings

ASC

ASC(s\$)

ASCII code of the first character of s\$; -1 for an empty string.

CHR\$

CHR\$(n)

The character with code n.

PRINT CHR\$(65) gives A

LEFT\$

LEFT\$(s\$,n)

The first n characters of s\$.

MID\$

MID\$(s\$,start[,n])

n characters of s\$ from position start.

RIGHT\$

RIGHT\$(s\$,n)

The last n characters of s\$.

LEN

LEN(s\$)

The length of s\$.

INSTR

INSTR(a\$,b\$[,start])

The position of b\$ in a\$, or 0.

PRINT INSTR("HELLO","LL") gives 3

STR\$

STR\$(n)

STR\$(~(n))

n as a string (decimal, or hexadecimal with ~).

STRING\$

STRING\$(n,s\$)

s\$ repeated n times.

VAL

VAL(s\$)

The number at the start of s\$.

EVAL

EVAL(s\$)

Evaluates the expression in s\$.

PRINT EVAL("2*21") gives 42

6.8 Data in the program

DATA

DATA item[,item...]

Values for READ. See READ, RESTORE.

READ

READ v[,v...]

Reads the next DATA values.

RESTORE

RESTORE [line]

Lets READ start again at the first DATA, or at a line.

6.9 Keyboard and text on the screen

PRINT

PRINT [items]

Prints values. Separators: ; joins, , moves to the next field (set by @%), ' starts a new line. TAB(x) and SPC(n) position the cursor; ~ prints hexadecimal; a ; at the end keeps the cursor on the line.

PRINT "A=";A,~B;

INPUT

INPUT ["prompt",] v[,v...]

INPUT LINE a\$

Reads values from the keyboard. INPUT LINE reads a whole line.

GET

k=GET

Waits for a key and returns its code. See KEYBOARD.

GET\$

k\$=GET\$

Waits for a key and returns it as a string.

INKEY

k=INKEY(t)

Waits up to t centiseconds for a key; -1 if none came. INKEY(0) does not wait.

INKEY(-n) is TRUE while the key with the BBC's number n is held (the C64 key in its place): -1 SHIFT, -2 CTRL, -99 SPACE, -74 RETURN, -113 ESCAPE (RUN/STOP), -26 / -122 cursor left / right, -58 / -42 up / down, f1-f8 on F1-F7 (f2, f4, f6, f8 with Shift: -114 f1, -115 f2, -116 f3, -21 f4, -117 f5, -118 f6, -23 f7, -119 f8), -106 COPY (F1); letters and digits as on the BBC (-66 A, -98 Z, ...). Keys the C64 lacks (f0, f9, TAB, CAPS LOCK, []) are never held. INKEY(-256) gives 255: a BBC B with OS 1.2.

IF INKEY(-98) THEN X%=X%-1

INKEY\$

k\$=INKEY\$(t)

As INKEY, returning a string (empty if no key).

TAB

PRINT TAB(x)

PRINT TAB(x,y)

Moves to column x, or to column x of row y.

SPC

PRINT SPC(n)

Prints n spaces.

POS

POS

The column of the text cursor.

VPOS

VPOS

The row of the text cursor.

COUNT

COUNT

The number of characters printed since the last new line.

WIDTH

WIDTH n

Starts a new line after n characters (WIDTH 0: off).

CLS

CLS

Clears the text screen.

COLOUR (also COLOR)

COLOUR n

Sets the text colour (n) or background (128+n). See COLOURS.

On the text screen COLOUR 128+n prints the following characters in reverse video in colour n (black text on a coloured bar); COLOUR 128 ends it. The whole screen's background is VDU 19,0,n,0,0,0.

***PETSCII (also *ASCII, ASCII)**

*PETSCII print as a C64 does

*ASCII back to BBC (ASCII)

PETSCII mode, for PETSCII art and C64-style screens (text modes such as MODE 100; needs an REU). Bytes printed go to the screen as on a C64: the C64's own character set (upper case and graphics; CHR\$14 switches to lower case, CHR\$142 back), colour codes (CHR\$5 white, 28 red, 144 black, ...), CHR\$18 reverse on, 146 off, 147 clears the screen, 19 home, 17/145/29/157 move the cursor. CHR\$13 starts a new line. BBC's PRINT TAB(x,y) and COLOUR do not work in this mode; MODE and VDU 23 do. GET and INKEY give PETSCII codes, so the C= and SHIFT graphics keys read as on a C64. Lines you type are shown as usual.

The mode ends with *ASCII, MODE or an error (BASIC's messages are ASCII).

*PETSCII

PRINT CHR\$147;CHR\$28;"RED";CHR\$18;"R"

***SEQ**

*SEQ file [LOWER|UPPER] show a .seq

Shows a PETSCII file (PETSCII art usually comes as .seq, e.g. from Petmate) on a cleared text screen in PETSCII mode, then waits for a key; ESCAPE stops. LOWER shows it in the lower case character set, UPPER in upper case and graphics (screens made in lower case often do not say so themselves); without either the set stays as it is (upper case when PETSCII mode was off). If PETSCII mode was off it goes off again and the screen is cleared. Any filing system: SD, USB, /REU, IEC drives.

6.10 Time

TIME

TIME

TIME=n

A clock counting centiseconds; can be set. It keeps real time at every CPU speed. For the date and time see *TIME.

6.11 Machine code

BBC BASIC has an assembler built in ([...]). Zero page &70-&8F is free for your own machine code, as on a BBC Micro; the MOS entry points are in chapter 13.

ASSEMBLER BBC BASIC contains a 6502 assembler. Square brackets enclose assembly language; P% is the address to assemble at; OPT sets listing and errors.

```
10 DIM C% 20
20 FOR Z%=0 TO 2 STEP 2
30 P%=C%:[OPT Z%
40 LDA #65:JSR &C7EE:RTS
50 ]:NEXT
60 CALL C%
```

OPT 0 assembles silently; OPT 2 reports errors; add 1 to list. Two passes resolve forward references. Labels start with a dot.

U64: OSWRCH is &C7EE, not &FFEE (see *HELP MOS). Zero page &70-&8F is free for your code.

OPT

[OPT n

Assembler option. See ASSEMBLER.

CALL

CALL address[,params]

Calls machine code. A, X, Y and the carry come from A%, X%, Y% and C%.

USR

x=USR(address)

Calls machine code and returns A, X, Y and P packed in an integer. See CALL.

OSCLI

OSCLI string

Runs a * command from a string, so it can contain variables.

OSCLI "TURBO "+STR\$(S%)

7. Files

Files live on the Ultimate's drives (SD card, USB sticks, its RAM disk /Temp) and, with an REU, on the MOS's own RAM disk /REU. Paths are the Ultimate's: /SD/games/elite, or relative to the current directory (*DIR).

7.1 The filing system

FILES Files are kept on the Ultimate's own storage: USB sticks (/Usb0, /Usb1), the SD card (/SD) and the RAM disk (/Temp, gone after power off). Names are Ultimate paths, relative to the current directory.

*DIR /Usb0/bbc	choose a directory
*CAT	list it
SAVE "game"	save the program
LOAD "game"	load it again
CHAIN "game"	load and run

Two files can be open at the same time (channels &11 and &12): OPENIN, OPENOUT, OPENUP, BGET#, BPUT#, PRINT#, INPUT#, PTR#, EXT#, EOF#, CLOSE#.

Programs are saved in the BBC's tokenised format, so programs from BBC Micro disc images can be loaded once extracted to a file.

See also: *CAT*, *DIR*, *LOAD*, *SAVE*, *TYPE*, *DELETE*, *RENAME*, *MKDIR*.

/REU (also RAMDISK, RAM) The REU RAM disk: up to 127 files in the REU from 1 MB up (an REU of 2 MB or more), as the directory */REU*. *DIR /REU* makes it current (*DIR ..* goes back), or give names as */REU/NAME*.

LOAD, *SAVE*, *CHAIN*, *LOAD*, *SAVE*, *CAT*, *DELETE*, *RENAME* and *TYPE* work there at REU speed, as do *OPENIN*, *OPENOUT*, *OPENUP*, *BGET#*, *BPUT#*, *PTR#*, *EXT#* and *EOF#*. **COPY* copies between */REU* and the Ultimate's drives:

**COPY GAME /REU*

**COPY /REU/GAME /Usb0/games*

Names up to 16 characters, no directories. The disk survives a reset, not switching off.

7.2 Directories and files

***BROWSE**

**BROWSE* (F1 on an empty line)

A file browser for the current folder, with the keys of UBoot64-v2's: cursor up and down move, P and U a page down and up, T and E the top and the end; cursor right (or RETURN) enters a folder, cursor left (or DEL) goes one folder up, / (or the up arrow) to the root, which lists the drives and */REU*. Q or ESCAPE leaves. At the prompt F1 on an empty line opens it (F1 is COPY once a copy cursor is out).

RETURN on a file: a name without a full stop is CHAINED as a BASIC program; .SEQ is shown (*SEQ*), .MOD played (*MODPLAY*), .WAV played as sample 32, .D64 .D71 .D81 .G64 mounted (**MOUNT*), .UPIC loaded and shown, .VPL loaded as the palette; other files are typed. L lists (types) a file.

SPACE marks entries (a * in front). C copies and X cuts the marked ones (or the selected one), V pastes them into the folder shown: folders with everything in them. A name already there asks: Y replaces (a folder is merged), N skips, A replaces all, Q stops. A move on one drive is a rename; otherwise a copy, then the source deleted. */REU* has no folders: only files go there.

D deletes the marked entries (or the selected one) after a question; a folder that is not empty asks again (Y/N/A/Q) and goes with everything in it. R renames, N makes a new folder. On the Ultimate's drives names with spaces work everywhere, but opening a file goes through a * command, where a name ends at a space. RUN/STOP stops a copy or delete; errors are shown and the browser goes on. Folders are followed 8 levels deep.

***CAT (also *.)**

**CAT* (also *.)

Lists the current directory: the path, then the files in two columns. Directories end in /. ESCAPE stops the listing.

***DIR (also *CD, CD)**

**DIR <path>* (also **CD*)

**DIR ~*

Changes the current directory. Paths are Ultimate paths: */Usb0/bbc*, */SD*, */Temp*, .. for the parent; ~ is the Ultimate's home directory (User Interface settings).

**DIR /Temp*

**DIR ..*

***PWD**

**PWD*

Prints the current directory.

***MKDIR (also *CDIR, CDIR)**

*MKDIR <name> (also *CDIR)

Creates a directory.

***DELETE**

*DELETE <name>

Deletes a file.

***RENAME**

*RENAME <old> <new>

Renames a file.

***COPY**

*COPY <file> <directory>

Copies a file into another directory on the Ultimate's storage, under the same name. An existing file there is not overwritten (error "File exists").

*COPY game /USB0/backup

***TYPE**

*TYPE <name>

Prints a text file. Lines end in CR; LF and other control codes are skipped. ESCAPE stops.

***LOAD**

*LOAD <name> <address>

Loads a file into memory at an address (decimal or &hex), between &1D00 (&900 with *SOUND NONE) and &7FFF.

*LOAD screen &3000

***SAVE**

*SAVE <name> <start> <end>

*SAVE <name> <start> +<length>

Saves memory to a file. The end address is not saved. Memory up to &7FFF can be saved.

*SAVE code &3000 +&200

***INFO**

*INFO <name>

Shows a file as DFS does: name, L if locked, load and execution address, length (hexadecimal). Files on the Ultimate's drives have no addresses (0).

***ACCESS**

*ACCESS <name> L lock

*ACCESS <name> unlock

Locks a file on the RAM disk (/REU): a locked file cannot be deleted, replaced, renamed or opened for output ("Locked"). The Ultimate's drives have no lock: their files give an error.

***DUMP**

***DUMP** <name>

Shows a file in hexadecimal: the offset, 8 bytes and the same bytes as characters (. for control codes). ESCAPE stops.

***RUN**

***RUN** <name>

Loads machine code and calls it. A file with a load address (*SAVE on /REU) goes there and is called at its execution address. Otherwise the file is a C64 program file: its first two bytes are the load address, and it is called there. Only &0400-&7FFF.

***SPOOL**

***SPOOL** <name> start
***SPOOL** end

Copies everything printed into a file, as on a BBC Micro: *SPOOL name, then for example LIST, then SPOOL alone*. The output is collected in the REU's last 64K (up to 47K) and written to the file when the spool ends (also by CLOSE#0 or another *SPOOL*), *so it is fast and safe while the MOS prints during file operations. Needs an REU; the RAM disk ends below the REU's last 128K (this 64K and MAP's).*

***SPOOL** listing
LIST
***SPOOL**

***EXEC**

***EXEC** <name>

Reads the file as if it were typed: every byte becomes a key, at the prompt and in programs (GET, INKEY, INPUT), until the file ends; *EXEC alone or ESCAPE stops it. LF ends a line too (files from a PC). A program typed in as text can be loaded this way, or a list of commands run.

***EXEC** commands

***BUILD**

***BUILD** <name>

Writes the lines you type into a file, each ended by CR, until ESCAPE. The line numbers shown are not written. Made for *EXEC files.

7.3 Files from programs: channels

Two files can be open at the same time. OPENIN reads, OPENOUT creates, OPENUP reads and writes; all return a channel number (0 when the file cannot be opened).

OPENIN

c=OPENIN "name"

Opens a file for reading; 0 if it cannot be opened. See FILES.

OPENOUT

c=OPENOUT "name"

Creates a file (or empties an existing one) for writing.

OPENUP

c=OPENUP "name"

Opens an existing file for reading and writing; 0 if it cannot be opened.

BGET# (also BGET)

b=BGET#c

Reads one byte from channel c. See FILES.

BPUT# (also BPUT)

BPUT#c, b

Writes byte b to channel c.

PRINT#

PRINT#c, v[, v...]

Writes values to channel c, typed, for INPUT#.

INPUT#

INPUT#c, v[, v...]

Reads values written by PRINT#c.

PTR# (also PTR)

PTR#c

PTR#c=n

The position in the file on channel c; can be set.

EXT# (also EXT)

EXT#c

The length of the file on channel c.

EOF# (also EOF)

EOF#c

TRUE at the end of the file on channel c.

CLOSE# (also CLOSE)

CLOSE#c

Closes channel c; CLOSE#0 closes all channels.

7.4 Data in the REU

Programs can keep data in the REU, from &40000 up (the MOS uses the first 256K):

***SRWRITE**

*SRWRITE <start> <end> <REU addr>

*SRWRITE <start> +<len> <REU addr>

Copies memory to the REU, as the BBC Master's sideways RAM command. Programs keep data there (tables, levels, text, pictures) and fetch it with *SRREAD*. *REU addresses &40000-&8FFFF (320K; above them the Upic picture, UPIC, the snapshots, *SNAPSHOT, and from &100000 the RAM disk); memory from PAGE to &7FFF. Numbers decimal or &hex.*

*SRWRITE &3000 +&1000 &40000

OSWORD &42 does the same from a program (see OSWORD).

***SRREAD**

*SRREAD <start> <end> <REU addr>

*SRREAD <start> +<len> <REU addr>

Copies from the REU into memory at start, the reverse of *SRWRITE.

*SRREAD &3000 +&1000 &40000

***REULOAD (also *REUSAVE, REUSAVE)**

*REULOAD <file> [<REU addr>]

*REUSAVE <file> <REU addr> <len>

Load a file straight into the REU, or save part of the REU to a file: the Ultimate copies the bytes itself, they never pass through the C64's memory, so megabytes take moments. Program area only: &40000-&8FFFF, as *SRWRITE* and *SRREAD*. *REULOAD loads at &40000 when no address is given, and stops at &8FFFF. Files on the Ultimate's drives (not /REU). Numbers decimal or &hex.

*REUSAVE LEVELS &40000 &20000

*REULOAD LEVELS

*SRREAD &3000 +&800 &40000

***REU**

*REU SAVE

*REU LOAD [Y]

Saves the whole REU to the Ultimate's REU preload image (the file set in its menu under C64 and Cartridge Settings), and loads it back: the /REU RAM disk and a program's REU data survive switching off. *REU LOAD replaces everything in the REU, so it asks first unless Y follows. Set the REU preload in the Ultimate's menu to load it at power-on instead.

8. Graphics

The BBC Micro's screen modes 0-7 (80-column modes as their 40-column twins), drawn with the BBC's coordinates (1280 x 1024, 0,0 at the bottom left); on a 64 MHz Ultimate as Upic Lite pictures with square pixels and 256 lines, elsewhere with the VIC-II. MODE 7 is teletext, MODE 100 the C64's text screen, MODE 103 and 104 the C64's own bitmap modes.

8.1 Screen modes

MODE

MODE n

Selects a screen mode and clears it.

MODE	size	col	text	U64
0	320x200	2	80x32	VIC-II
1	320x256	4	40x32	Upic Lite
2	160x256	16	20x32	Upic Lite
3	text	2	80x25	VIC-II
4	320x256	2	40x32	Upic Lite
5	160x256	4	20x32	Upic Lite
6	text		40x25	Upic Lite
7	teletext		40x25	VIC-II
100	C64 text		40x25	
103	320x200	16	40x25	hires
104	160x200	16	20x25	multicol.
110	384x256	16	40x25	Upic full
111	288x256	16	40x25	Upic 48 MHz

MODE 7 is teletext, as on the BBC: screen memory at &7C00 (HIMEM &7C00), colour codes 129-135 (text) and 145-151 (mosaic graphics), 141 double height, 154 separated graphics, 157 new background, 156 black background, 158 hold graphics, 136 flashing. Writing to &7C00 directly (?&7C00=...) shows within a few frames. The machine starts in the C64 text screen; MODE 100 returns to it.

MODE 0 and MODE 3 have 80 columns in a 4x6 font (X11 misc-fixed, public domain), each character 4 pixels wide: MODE 0 with graphics as MODE 4 (rows of 6 lines), MODE 3 text in rows of 8. They stay on the VIC-II (HIMEM &8000): Upic Lite blurs such thin letters. *HELP texts stay 40 wide*; CAT uses the width. MODE 1 on the VIC-II (MODE 129, *SHADOW) uses the same font for its 40 columns: its 160 double-wide pixels leave 4 per character. MODE 6 uses rows of 10 lines (250 lines), as on the BBC; without Upic Lite they are the 40x25 text screen. MODE 103: 2 colours per 8x8 cell, MODE 104: 4 per 4x8 cell, as the VIC-II does.

Upic Lite: a live picture of 320x256 (MODE 2 and 5: 160x256 with double-width pixels, 16 or 4 colours anywhere) with square pixels, as on the BBC, on 64 MHz machines (Ultimate 64 Elite II, C64 Ultimate) with firmware 3.15+. The screen memory lies below HIMEM as on the BBC (MODE 4, 5, 6: &5800; MODE 1 and 2: &3000), and BASIC runs at about 1/6 of its speed while the picture is shown. Elsewhere these modes use the VIC-II (200 lines, 32 rows of 6-line text).

Upic full (MODE 110, 111): graphics go into a 16-colour picture in the REU (OS units 1536 x 1024 in both: 4 per pixel in 110, 16/3 in 111, so circles stay round), text to a 40x25 console on the screen; the picture shows with *UPIC SHOW (see UPIC). MODE 111 has 288 pixels: 3 of every 4 that a 48 MHz Ultimate shows; MODE 110 needs 64 MHz. HIMEM stays &8000.

*SHADOW

*SHADOW shadow modes
 *SHADOW 1 normal modes

As on a BBC Master: from the next MODE on, a shadow mode keeps the screen out of BASIC's memory, so HIMEM stays &8000. Here that is the VIC-II version of the mode instead of Upic Lite. MODE 128+n is shadow for that mode only. OSBYTE &72 does the same from a program (X = 0 shadow, 1 normal), OSBYTE &75 bit 4 says whether the current mode is one. See *MEMORY.

8.2 Drawing

CLG

CLG

Clears the graphics window in the background colour (GCOL 0,128+n).

GCOL

GCOL action,colour

Sets the graphics colour and how it is drawn: action 0 store, 1 OR, 2 AND, 3 EOR, 4 invert. Colour 128+n sets the background (CLG).

Pattern fill (ECF, as on the BBC Master): action + 16 * n uses pattern n (1-4) instead of the colour, for everything drawn. VDU 23,12+n-1,c1,...,c8 defines pattern n as 2x4 pixels of logical colours (row by row); VDU 23,11,0;0;0;0; restores the default patterns. VDU 23,2-5 (GXR and Master) define pattern 1-4 as eight screen bytes in the BBC layout of the mode; the first two pixels of the first four bytes are used. VDU 23,6 (the dot-dash line pattern) and *FX 163,242 are accepted; dotted lines plot every second pixel.

```
GCOL 0,1:DRAW 1279,1023
GCOL 16,0:PLOT 157,640,512
```

MOVE

```
MOVE x,y
```

Moves the graphics cursor (PLOT 4,x,y).

DRAW

```
DRAW x,y
```

Draws a line from the graphics cursor to x,y in the graphics colour (PLOT 5,x,y). Coordinates: 1280 x 1024, 0,0 at the bottom left, as on the BBC.

PLOT

```
PLOT k,x,y
```

Graphics plotting, as on the BBC: k AND 3 = 0 moves, 1 draws in the graphics colour, 2 inverts, 3 draws in the background colour; k AND 4 = 0: x,y relative to the graphics cursor.

k 0-7	line
k 8-15	line, no end point
k 16-31	dotted line
k 32-63	line, no start point
k 64-71	point
k 72-79	line fill left/right (GXR)
k 80-87	filled triangle
k 88-95	line fill right (GXR)
k 96-103	filled rectangle
k 112-119	filled parallelogram
k 128-135	flood fill (background)
k 136-143	flood fill to foreground
k 144-151	circle
k 152-159	filled circle
k 160-167	arc
k 168-175	segment
k 176-183	sector
k 184-191	block move/copy (GXR)
k 192-199	ellipse
k 200-207	filled ellipse
k 216-223	rectangle (U64)
k 224-231	triangle (U64)
k 232-239	sprite (GXR, see SGET)
k 240-247	parallelogram (U64)

Triangles use the last three points; a rectangle's corners are the previous point and x,y; a parallelogram's fourth corner is worked out as on the BBC. Circles: the centre is the previous point, x,y a point on the circle (PLOT 153,r,0 after MOVE cx,cy). Ellipses: centre, then a point at the end of the horizontal axis, then x,y gives the height. The (U64) outline codes are extensions; the BBC has no outline triangle, rectangle or parallelogram. Flood fill 128-135 fills the background-coloured area around x,y; 136-143 fills up to the foreground colour. Arcs,

segments and sectors: the centre is the point two plots back, the previous point gives the radius and the start, x,y the end direction; drawn anticlockwise.

Line fills (as the GXR and the Master): 72-79 fills the row at x,y left and right over background-coloured pixels; 88-95 fills to the right over pixels that are not the background colour. Block move/copy: the rectangle between the last two points goes with its bottom left corner to x,y; 185/189 move it (background left behind), 186/190 copy it. Overlapping rectangles are handled.

```
MOVE 0,0:MOVE 99,99:PLOT 190,400,0
```

POINT

POINT(x,y)

The colour of a pixel (logical colour), or -1 outside the graphics window.

8.3 VDU codes

VDU

```
VDU n[,n...]  
VDU n;
```

Sends bytes to the screen driver; n; sends two bytes. See VDUCODES for the codes.

VDUCODES VDU codes work as on a BBC Micro:

```
4      text at the text cursor  
5      text at the graphics cursor  
7      bell (SOUND &13,-15,101,6)  
8-11   cursor left/right/down/up  
12     clear the text window (CLS)  
13     start of line  
16     clear the graphics window (CLG)  
17,n   COLOUR n  
18,a,n GCOL a,n  
19,l,p,0,0,0 colour l shows colour p  
19,l,16,r,g,b colour l as RGB  
20     default colours and palette  
22,n   MODE n  
23,1,n cursor off (0) / on (1)  
23,11  default ECF patterns  
23,12-15 define ECF pattern 1-4  
23,c,8 rows define character c  
24,l;b;r;t; graphics window  
25,k,x;y; PLOT k,x,y  
26     default windows  
28,l,b,r,t text window  
29,x;y; graphics origin  
30     cursor home  
31,x,y TAB(x,y)  
127    delete
```

Windows, the graphics codes and the palette act in the graphics modes (MODE 1, 2, 4, 5, 103, 104); in text modes 12, 17 and 22 act.

VDU 23,c,r1,...,r8 (c = 32-255) defines a character: 8 rows, top first, bit 7 left. The text screen shows codes 32-127; graphics modes show all of them with an REU (the Ultimates; kept over MODE changes). MODE 7 keeps its teletext glyphs. OSWORD 10 reads a definition.

8.4 Colours, palettes, fonts and CharPad maps

COLOURS COLOUR n sets the text colour, COLOUR 128+n the background. BBC colour numbers map to C64 colours:

0 black	1 red	2 green
3 yellow	4 blue	5 magenta
6 cyan	7 white	8 orange
9 brown	10 light red	
11 dark grey	12 grey	
13 light green	14 light blue	
15 light grey		

*PALETTE

```
*PALETTE LOAD <file> [VPL]
*PALETTE SAVE <file>
*PALETTE BBC      BBC's pure colours
*PALETTE C64      Ultimate's defaults
*PALETTE RESET    palette at start-up
```

The RGB colours of the 16 C64 colours, as the Ultimate shows them. Files are 48 bytes (red, green, blue per colour, in C64 colour order); with VPL a VICE palette file. BBC gives the 8 C64 colours the BBC colours use the BBC's pure red, green, ... The start-up palette is the one chosen in the Ultimate's menu.

VDU 19,l,16,r,g,b sets logical colour l to an RGB colour (0-255 each): in Upic Lite modes that colour only, elsewhere the C64 colour l shows.

Needs firmware 3.15a or later; all but C64 need an REU. Upic Lite and MODE 7 use the palette for their own picture and keep it until the next MODE change.

*FONT

```
*FONT LOAD <file> [fmt]  load
*FONT SAVE <file> [fmt]  save
*FONT BBC      BBC Micro characters
*FONT C64      C64 character ROM
*FONT DEFAULT  the start-up font
```

fmt: C64 (default), BBC or ASCII.

A CharPad project (.ctm, versions 6-9) is recognised by its contents: its first 256 characters become the font (needs an REU). *MAP LOAD also loads its map.

C64 is a C64 charset: 2048 bytes in screen-code order (2050 with a load address also loads). BBC is 768 bytes, characters 32-127 in ASCII order, as in a BBC Micro MOS ROM at &C000. ASCII is 2048 bytes, characters 0-255 in ASCII order.

With an REU (the Ultimates) the font shows on the text screen and in the graphics modes and stays over MODE changes. Without one only the text screen changes. MODE 7 builds its teletext glyphs from the font when it starts. VDU 23 changes single characters. The BBC characters are Acorn's (see CREDITS).

*MAP

```
*MAP LOAD <file>  load
*MAP DRAW mx,my,w,h AT col,row
                        show a part
*MAP GET mx,my     M% = char
*MAP              its size
```

Maps from CharPad (.ctm project files, versions 6-9; needs an REU). LOAD makes the project's characters the font, sets the screen colour and keeps the map in the REU: tiles become characters, each with its colour (per project, tile or character). Up to 65535 characters wide or

high and 30720 in all, so wide scrolling levels fit. The RAM disk then ends 128K before the REU's end.

DRAW copies w x h characters from map column mx, row my to the text screen at col, row (text modes only, e.g. MODE 100), with their colours, cut at the edges of the map and the screen. It waits for the right moment of the frame itself, so the picture never tears and a loop needs no *FX 19. Arguments are numbers or A%-Z%, so a loop scrolls*: MAP DRAW X%,0,40,25 AT 0,0.

GET puts the character code at mx,my in M% (0 outside the map).

8.5 Sprites

*SPRITE (also SPRITES)

```
*SPRITE ON | OFF      the multiplexer
*SPRITE n ON x,y      show sprite n
*SPRITE n OFF         hide it
*SPRITE n SHAPE s      its shape (0-63)
*SPRITE n COLOUR c     its colour
*SPRITE n SIZE w,h     1 or 2 each way
*SPRITE n MULTI ON     multicolour (OFF)
*SPRITE n BEHIND       behind graphics
*SPRITE n FRONT        in front of them
*SPRITE COLOURS a,b    the shared pair
*SPRITE DEFINE s h     shape: 126 hex
*SPRITE LOAD s file    shapes from s on
*SPRITE                the sprites shown
```

Up to 32 sprites (0-31) of 24 x 21 dots on the VIC-II's 8, reused down the screen by a raster interrupt (at most 8 on one line: a 9th drops out where 9 meet). *SPRITE ON loads that multiplexer below HIMEM and lowers HIMEM by 1.25K (at the prompt at once, in a program with the next MODE: put it before MODE)*; *SPRITE OFF* gives the memory back, as **SOUND NONE* does for sound. A *MODE* hides all sprites.

x,y are BBC graphics coordinates (0-1279, 0-1023), of the sprite's bottom left corner; colours are BBC colours 0-15. Shapes: 3 bytes a row of 24 dots, 21 rows (as SpritePad), kept in the REU:

```
*SPRITE ON
*SPRITE DEFINE 0 FFFFFFFF...
*SPRITE 0 ON 640,512
```

SPRITE LOAD reads SpritePad files (SPD versions 1 to 5, SpritePad 1.8 to Pro 3; their sprites, not tiles or animations), files with a 3-byte header, or plain 64-byte shapes. A shape from a SpritePad file keeps its colour and multicolour setting: *SPRITE n SHAPE s* gives them to the sprite, and *SPRITE n COLOUR* afterwards changes the colour. Multicolour sprites have 12 x 21 double-wide dots in 4 colours: transparent, the sprite's own, and the pair *SPRITE COLOURS* sets for all sprites (the VIC-II has one pair).

In the graphics modes and *MODE 7* the bitmap leaves room for two shapes: the multiplexer copies each sprite's shape in as it comes, so sprites on the same lines can have at most 2 different shapes (the text screen holds all 64).

OSWORD &71 does it from programs without command parsing: block +0 sprite, +1 action (0 hide, 1 show at +2/+3 x, +4/+5 y, 2 shape +6, 3 colour +6, 4 collisions into +6: bit 0 another sprite, bit 1 the picture, cleared when read).

Upic Lite cannot show VIC-II sprites: use *MODE 128+n* there. Needs an REU.

8.6 Upic full: 384 x 256 in 16 colours

*UPIC

```

*UPIC SHOW [n]      show the picture
*UPIC LOAD file     a Upic file
*UPIC SAVE file     the picture
*UPIC CLEAR [c]     new, colour c
*UPIC AUTO ON|OFF   automatic shows

```

Upic (Aleksi Eeben): 384 x 256 pixels in 16 colours, every pixel its own colour, drawn by the CPU at 48 or 64 MHz. The picture is kept in the REU (&90000) and shown on request: SHOW moves BASIC's memory aside, shows the picture until a key is pressed, and puts everything back (the program goes on where it was; TIME counts on). Music pauses meanwhile: the CPU spends every cycle on the picture, so the SIDs are muted and a MOD stands still, then both go on. On a 48 MHz Ultimate 3 of every 4 pixels show.

LOAD reads Upic v1.3 files (as Upic Paint and the Upic Image Converter save them, with their palette) or a bare 49152-byte picture; SAVE writes v1.3. CLEAR starts a new picture in colour c with the C64 palette.

```

*UPIC LOAD harbour.upic
*UPIC SHOW

```

SHOW n shows it for n centiseconds at most (a key ends it earlier): progress views while a program draws.

MODE 110 and 111 draw into this picture: graphics there, text on a 40x25 console; colours are the mode's logical colours (as MODE 2), VDU 19 changes them (VDU 19,l,16,r,g,b any RGB). When the picture has changed it is shown by itself: when a program ends (until a key), in GET (the key is GET's) and in INKEY of half a second or more (at most that long). AUTO OFF leaves only SHOW.

Needs an REU of 1 MB; SHOW needs a 48 or 64 MHz Ultimate.

8.7 BASIC V graphics commands

***RECTANGLE (also *CIRCLE, CIRCLE, *ELLIPSE, ELLIPSE, *LINE, *POINT, *COLOUR, *GCOL, *OFF, OFF, *ON, BASICV, BASIC V)**

```

*RECTANGLE [FILL] x,y,w[,h]
*CIRCLE [FILL] x,y,r
*ELLIPSE [FILL] x,y,a,b
*LINE x1,y1,x2,y2
*POINT x,y
*COLOUR n,r,g,b
*GCOL [a,]c TINT t
*OFF *ON

```

The Archimedes' BASIC V graphics as MOS commands, for new programs (the interpreter is BBC BASIC 3.10). Each becomes the BBC's PLOT or VDU: RECTANGLE from x,y w wide and h high (h = w if left out; outline or FILL), CIRCLE centre and radius, ELLIPSE centre and half-axes a (across) and b (up), LINE and POINT, COLOUR n,r,g,b is VDU 19,n,16,r,g,b (any RGB for logical colour n), OFF and ON hide and show the text cursor.

GCOL c TINT t takes an Archimedes 256-colour (c 0-63: red bits 0-1, green 2-3, blue 4-5; tint 0, 64, 128, 192) and uses the nearest of the mode's palette colours: set the palette with *COLOUR first for exact colours (needs an REU).

Arguments are numbers (decimal, &hex, negative) or the variables A%-Z%; no expressions:

```

X%=640:Y%=512:R%=200
*CIRCLE FILL X%,Y%,R%
*RECTANGLE 0,0,1279,100

```

In the graphics modes RECTANGLE, CIRCLE, ELLIPSE, LINE and POINT take a fast path in RAM (768 bytes below HIMEM): about as quick as PLOT itself, some 25 times quicker than other * commands.

8.8 GXR software sprites

***SGET (also *SCHOOSE, SCHOOSE, *SDELETE, SDELETE, *SNEW, SNEW, *SRENUMBER, SRENUMBER, *SLOAD, SLOAD, *SMERGE, SMERGE, *SSAVE, SSAVE, *SSPACE, SSPACE, *SMASK, SMASK, GXR)**

*SGET n the last 2 points' box
*SCHOOSE n select sprite n
PLOT 237,x,y plot it (232-239)
*SDELETE n *SNEW (all)
*SRENUMBER n,m n becomes m
*SMASK n c|OFF colour c left out
*SLOAD f *SMERGE f *SSAVE f
*SSPACE n (accepted)
VDU 23,27,0,n select; 23,27,1,n get

Software sprites as the BBC Master's GXR: pictures taken from the screen and plotted anywhere, drawn into the picture by the graphics engine, so they work in every graphics mode, Upic Lite and Upic full included (VIC-II sprites: see SPRITE).

SGET n takes the rectangle between the last two graphics points (MOVE, MOVE), up to 192 x 255 pixels, as sprite n (0-255) and selects it; SCHOOSE n selects one. PLOT 232-239 plots the selected sprite with its bottom-left corner at x,y (k AND 3 and the GCOL action as for any PLOT: GCOL 3 EOR draws and erases), clipped to the graphics window. A sprite keeps its size in pixels of the mode it was taken in.

SMASK n c (U64) leaves colour c out when sprite n is plotted, so it sits over a background without its box; SMASK n OFF plots every pixel again (as the GXR).

*SLOAD reads a sprite file (all sprites replaced), SMERGE adds one: our own format, or real GXR sprite files from a BBC Micro or Master (MODE 0, 1, 2, 4, 5), decoded from their mode. SSAVE writes our format. Sprites live in the REU (46K); SSPACE is accepted for GXR programs. Needs an REU. *SEdit n edits sprite n (see SEDIT).*

```
MOVE 0,0:MOVE 63,63:*SGET 1
*SCHOOSE 1
GCOL 0,0:PLOT 237,640,512
```

***FLOOD (also *NOFLOOD, NOFLOOD, *GXR, *NOGXR, NOGXR)**

*FLOOD *NOFLOOD *GXR *NOGXR

Accepted for programs written for the BBC Micro's Graphics Extension ROM; they do nothing here. The GXR's functions are always on: flood fills, line fills, block copy, circles, ellipses, arcs, patterns and sprites (see PLOT, GCOL, SGET).

8.9 The sprite editor

***SEdit (also EDITOR)**

*SEdit n GXR sprite n (0-255)
*SEdit VIC s VIC-II shape s (0-63)

The sprite editor, for both kinds: the GXR software sprites (SGET) and the VIC-II hardware sprites' shapes (SPRITE). Each pixel is a block on the text screen; a VIC shape is also shown at its real size. RETURN keeps the sprite, ESCAPE leaves it as it was; the screen mode before is set again (cleared). A new GXR sprite starts at 16 x 16.

cursor keys move
0-9 + - choose the colour
SPACE paint
P pick the colour there
F fill the area
C clear
H V flip across / down
I J K L shift up/left/down/right

S size (GXR)
 G copy another sprite in
 M multicolour (VIC shape)

A VIC shape in multicolour shows 12 double-wide pixels: 1 and 3 are the pair of *SPRITE COLOURS, 2 the sprite's own colour. GXR sprites have colours 0-15 (as logical colours of the mode they are plotted in).

9. Sound

SOUND and ENVELOPE as on a BBC Micro, on the SIDs. In SID mode (the start-up mode) the engine uses all voices of up to 8 SIDs (24 voices), with instruments; *SOUND BBC sounds like a BBC Micro. Songs are written with SOUND while recording and play in the background.

9.1 Notes and envelopes

SOUND

SOUND c,a,p,d

Queues a note, as on a BBC Micro. c = &HSFN: N channel (0 noise, 1-3 tone), F=1 flush the queue first, S sync with S other channels (they start together), H=1 hold (the previous note's release goes on). a: -15 (loud) to 0, or envelope 1-16. p: pitch, 4 steps per semitone, 53 = middle C, 89 = A 440 Hz; channel 0: 0-2 periodic noise, 4-6 white noise (high to low), 3 and 7 at channel 1's pitch. d: 1/20 s, 255 or -1 for ever.

This is BBC mode (*SOUND BBC): channels 1-3 on SID 1, channel 0 on the second SID (or with one SID, on channel 3's voice while it plays).

SID mode (the start-up mode): c = &FSVV: VV voice 0-23 (voice v MOD 3 of SID v DIV 3), S sync with S other voices (0-15), F: 1 flush, 2 hold, 4 tie (new pitch, no new attack). a: instrument 1-64; instrument + 256 * loudness (1-15, scales the instrument's own sustain: &0A08 is instrument 8 at 10/15; drums stay drums); -15..-1 the voice's instrument at that loudness; 0 a rest. p: as BBC mode, also below 0 for low notes.

Voices 24-30 (SID mode) are Ultimate Audio's sample channels 0-6: SOUND 24+c,n,p,d plays sample n (*SAMPLE*), n + 256 loudness (1-15) for its volume, -15..-1 the channel's last sample at that loudness, 0 silence; p 53 is the sample's own rate (4 steps a semitone, as above); d in 1/20 s, 0 the whole sample, 255 looped. A sample starts at once (no queue).

Three notes wait per voice; see ADVAL.

ENVELOPE

ENVELOPE n,t,pi1,pi2,pi3,pn1,pn2,pn3,
 aa,ad,as,ar,ala,ald

Defines envelope n (1-16), as on a BBC Micro. t: step length in 1/100 s (add 128: the pitch part does not repeat). pi1-pi3: pitch change per step in three sections of pn1-pn3 steps. aa, ad: amplitude change per step in the attack and decay, up to the levels ala and ald (0-126); as: change per step while the note lasts; ar: in the release, down to 0. Use it with SOUND c,n,p,d.

ADVAL

ADVAL(n)

ADVAL(-5) to ADVAL(-8): free room in the sound queue of channel 0 to 3, as on a BBC Micro: 15 when empty, 4 bytes per note waiting. In SID mode ADVAL(-5) to ADVAL(-28) for voices 0-23, and ADVAL(-29) to ADVAL(-52) the note voice 0-23 plays now (1-96, 49 = middle C; 0: none), for meters while a song plays (*SONG).

The analogue channels read the C64's control ports (see *ADVAL): ADVAL(1) and ADVAL(2) are joystick 1's x and y (control port 2), ADVAL(3) and ADVAL(4) joystick 2 (port 1); 0-65520,

high is left and up, 32768 in the middle (a joystick gives 65520, 32768 or 0). ADVAL(0) AND 3 gives the fire buttons (bit 0 joystick 1), ADVAL(-1) the keys waiting in the keyboard buffer. Other values return 0.

```
IF ADVAL(0) AND 1 THEN PROCfire
X%=ADVAL(1) DIV 256
```

***SPEECH (also *SAY, SAY, *SPEAK, SPEAK, SAM)**

```
*SPEECH LOAD f [f]   your SAM files
*SPEECH              what is loaded
*SPEECH SPEED n      (also PITCH,
                     THROAT, MOUTH)
*SAY text            English
*SPEAK phonemes      SAM's spelling
```

Speech with S.A.M. (Software Automatic Mouth, Don't Ask Software 1982), which is not part of U64 BBC BASIC: load your own copy of the C64 package once, its ".sam 1" (SAM, loads at &9500) and ".sam 2" (Reciter, &7D00, for English), copied as files without spaces in their names; they are told apart by their load address and kept in the REU (its last 64K, with *SPOOL*). SAY speaks English through Reciter, *SPEAK SAM's phonemes (e.g. HEH4LOW). Defaults: speed 72 (bigger is slower), pitch 64 (smaller is higher), throat and mouth 128. The screen is blank while SAM speaks, at 1 MHz; the program then goes on. Up to 230 characters; a word without a vowel (BBC) is spelt out. Needs an REU.

```
*SPEECH LOAD sam1.prg sam2.prg
*SAY Hello, I am a BBC Micro.
```

9.2 The sound engine

***SOUND**

```
*SOUND          show the mode
*SOUND SID      SID mode (start-up)
*SOUND BBC      BBC Micro mode
*SOUND NONE     no engine: more memory
*SOUND OFF      sound off
*SOUND ON       sound on again
*SOUND STOP     silence, queues empty
```

SID mode: SOUND plays all voices of up to 8 SIDs with instruments (see SOUND, *INSTRUMENT). BBC mode: SOUND and ENVELOPE as on a BBC Micro, channels 0-3. Changing the mode stops all sound; envelopes are kept (with an REU).

NONE takes the sound engine out of memory: &0900-&1CFF is free, and a program may set PAGE=&900 for 31K. SOUND is then ignored. *SOUND SID or BBC brings the engine back, but only while PAGE is &1D00 or above.

***SID**

```
*SID          list the SIDs
*SID PAN <unit> <p>    stereo
*SID VOL <unit> <dB>   volume
*SID SETUP    8 SIDs
```

The SIDs found at start-up: up to 8, at &D400-&D7E0. The cartridge's .cfg sets up 8 UltiSIDs (two UltiSIDs split four ways, as Heartbeat Soundtracker does) and switches them on, UltiSID 1 left and UltiSID 2 right.

*SID SETUP sets up the 8 SIDs as the cartridge's .cfg does (two UltiSIDs split four ways, UltiSID 1 left, UltiSID 2 right) and finds them again. The Ultimate applies the .cfg only when the cartridge is started from its file browser, so BBC BASIC does this at start-up by itself when it finds fewer than 8 SIDs. Lasts until the Ultimate is reset.

unit: 1 or 2 (UltiSID 1 or 2; with the .cfg the 1st, 3rd, 5th and 7th SID are UltiSID 1's) or S1, S2 (the SID sockets). p: -5 (left) to 5 (right), 0 centre. dB: -42 to 6, or OFF; rounded to the Ultimate's steps (-42 -36 -30 -27 -24, then -18 to 6 in single dB). The setting takes effect at once and lasts until the Ultimate is reset. Needs firmware 3.15a.

9.3 Instruments and songs

*INSTRUMENT

```
*INSTRUMENT n          list it
*INSTRUMENT n parts    change it
*INSTRUMENT n NAME text
*INSTRUMENT n COPY m
*INSTRUMENT DEFAULT
*INSTRUMENT SAVE <file>
*INSTRUMENT LOAD <file>
```

64 instruments for SID mode; 1-16 are ready-made: SQUARE PIANO ORGAN STRINGS BRASS LEAD BASS FLUTE PLUCK BELL KICK SNARE HIHAT PAD ARP SYNC. Needs an REU. Parts:

TRI SAW PULSE NOISE RING SYNC: the waveform (several mix). ADSR a d s r: the envelope (0-15 each). PW w [rate top bottom]: pulse width 0-4095; with a sweep, the width moves by rate (-128 to 127) each interrupt, turning at top and bottom (the width's high part, 0-15).

FILTER [LOW] [BAND] [HIGH] [RES r] [CUT c [rate top bottom]]: through the filter; c the cut-off (0-255), a sweep as for PW (top, bottom 0-255); STOP: the sweep stops at its end, BOUNCE: it turns. NOFILTER.

VIB delay width rate: vibrato after delay interrupts, width in 1/8 semitones, rate 1-63. PORTA speed: slides to each new note (speed/64 semitone per interrupt). FINE f: tuning in 1/64 semitones (-128 to 127). RESTART / NORESTART: hard restart (a clean new attack).

WAVES w ...: up to 16 waveform steps (255 n: go on at step n; 0: end); ARP a ...: semitone offsets for the same steps (chords, drums); SPEED n: interrupts per step.

```
*INSTRUMENT 20 COPY 2 SAW
*INSTRUMENT 20 ADSR 0 8 0 9
*INSTRUMENT 21 PULSE PW 800 6 14 2
*INSTRUMENT 21 VIB 20 2 6
*INSTRUMENT 22 WAVES 65 65 255 0
*INSTRUMENT 22 ARP 0 4 7 SPEED 2
```

*SONG (also TRACKER)

```
*SONG                show it
*SONG NEW [bpm [rpb]]
*SONG RECORD [OFF]
*SONG PLAY [position]
*SONG STOP
*SONG TEMPO bpm
*SONG TITLE text
*SONG COMPOSER text
*SONG SAVE <file>
*SONG LOAD <file>
```

A song of up to 24 voices kept in the REU, played in the background by the SID engine (*SOUND SID*), in every screen mode. Build it from BASIC: after SONG RECORD, SOUND writes notes into the song instead of playing them; the duration counts in rows (a beat has rpb rows, default 4; tempo bpm, default 120). Each voice keeps its own place, so voices can be written one after another. *SONG RECORD OFF ends the song after the furthest note.

```
*SONG NEW 100
*SONG RECORD
SOUND 0,4,53,4:SOUND 0,4,61,4
```

```
SOUND 1,7,25,8
*SONG RECORD OFF
*SONG PLAY
```

Pitch as SOUND, rounded to semitones; amplitude 1-64 an instrument, -15 to -1 the voice's instrument at that level, 0 a rest. Up to 28 patterns of 64 rows (1792 rows). ADVAL(-29-v) gives the note voice v plays (49 = middle C), to draw meters while it plays.

Ultimate Audio: samples and MOD music

On the U64 with its 16 MB REU, Ultimate Audio's seven sample channels play sound effects (SOUND on voices 24-30) and ProTracker MOD music in the background.

*SAMPLE

```
*SAMPLE LOAD <n> <file> [<rate>]
*SAMPLE CLEAR      remove them all
*SAMPLE           list them
```

Sound samples for Ultimate Audio (U64, 16 MB REU): up to 32, 256K in all, kept in the REU. A file is an 8-bit mono WAV (its rate from the file) or raw signed 8-bit data at rate Hz (8000 when not given), from the Ultimate's drives. SOUND plays them on voices 24-30, also while a MOD plays (on 28-30 then).

```
*SAMPLE LOAD 1 BOOM.WAV
SOUND 24,1,53,0
```

*MODPLAY (also *MODSTOP, MODSTOP)

```
*MODPLAY <file>  play a MOD
*MODSTOP         stop it
```

Plays a ProTracker MOD (4 channels, 15 or 31 samples, up to 768K) in the background on Ultimate Audio (U64, 16 MB REU), while the program runs. The MOD player takes the place of the SID sound engine: SOUND on the SID voices is silent until *MODSTOP*, which brings it back; samples still play (SOUND 28-30, SAMPLE). The interrupt runs at the MOD's tempo meanwhile; TIME stays right.

```
*MODPLAY /SD/MUSIC/SONG.MOD
```

The player is ModPlayer_16k by Freshness (6510nl), CSDb 131522, ported to MADS.

10. Network

The Ultimate's network: TCP and UDP connections and web pages as file channels, JSON replies read by path. Plain HTTP only: the Ultimate's firmware has no HTTPS. The demo demos/net/U64TEXT (a Ceefax-style information service) uses all of it.

*NET (also NETWORK)

```
*NET           the address
*NET GET <url> [file] fetch
*NET POST <url> <body> [file]
*NET PUT <url> <body> [file]
*NET DELETE <url> [file]
*NET HEADER [Name: value]
*NET TIME [server] set the clock
*NET RANDOM      seed RND
```

The Ultimate's network. *NET shows its IP address, netmask, gateway and MAC address.

NET GET, POST, PUT and DELETE send an HTTP request over plain http (not https: the Ultimate cannot do encryption). The reply goes to the file (any size, also on the RAM disk) or,

without one, to the screen. A body is “text” or the name of a file (up to 64K). NET HEADER adds a line such as Content-Type: application/json or an API key to every following request; *NET HEADER alone removes them. Needs an REU.

```
*NET GET http://example.com/ PAGE
*NET HEADER Content-Type: text/plain
*NET POST http://h/api "Hi" R
```

A reply with status 400 or more raises error 223 (e.g. HTTP 404 Not Found), which ON ERROR can catch; OSBYTE &65 gives the last status code in X and Y (0: none):

```
A%=&65:S%=USR&C7F4
S%=(S% AND &FFFF00) DIV 256
```

NET TIME has the Ultimate set its clock from a time server (its own SNTP client, in its time zone setting); the server is optional. NET RANDOM seeds RND from the drand public randomness beacon, so a program gets a sequence nobody can predict.

Files on the network open as channels: OPENIN “http://...” reads a web reply like a file (BGET#, EOF#, EXT#; up to 16 KB), and OPENUP “TCP:host:port” or “UDP:host:port” opens a connection: BPUT# sends (a TCP line goes out at its CR or LF), BGET# waits for the next byte (ESCAPE stops it), EXT# is the number of bytes waiting, EOF# is TRUE when the other side has closed. PTR#c=0 sends what is waiting (a UDP datagram). At most two files and connections are open at a time.

*JSON

```
*JSON GET <url>    fetch and keep
*JSON <path>       show a value
*JSON NEW          start a body
*JSON SET <key> <value>
*JSON POST <url>   send the body
```

A JSON reply from a web address (plain http), read and kept by the Ultimate (firmware 3.15). Paths go down the reply with / and pick an array’s element with %n (from 0):

```
*JSON GET http://api.open-notify.org/
    iss-now.json
*JSON iss_position/latitude
*JSON %0/title
```

A string is shown as it is, a whole number in decimal, TRUE or FALSE; an object or array shows nothing (OSWORD &70 gives its count).

JSON NEW, SET and POST send JSON: the Ultimate builds an object from the keys set, the value being “text”, TRUE, FALSE, a number (32 bits, &hex too) or A%-Z%. JSON POST sends it; the reply (JSON) is kept as after JSON GET. NET HEADER’s lines go along (as with *JSON GET).

```
*JSON NEW
*JSON SET name "BBC"
*JSON SET score S%
*JSON POST http://host/scores
```

Programs use OSWORD &70 with a 9-byte block: +0/+1 the path’s address, +2 its length, +3/+4 where the value goes, +5 the room there. Back: +6 the type (0 none, 1 number, 2 TRUE/FALSE, 3 string, 4 object, 5 array), +7 the value’s length, +8 an object’s or array’s count. demos/net/JSONLIB has FNjson(path), FNjsonnum(path) and FNjsoncount(path\$).

11. Ultimate commands

The Ultimate’s drives, disk images, clock and speed.

11.1 Overview

ULTIMATE U64 BBC BASIC uses the Ultimate's command interface (UCI) for files, the clock and identification, and the U64 turbo registers for speed.

*UVER firmware targets
*TIME date and time
*TURBO CPU speed
*CAT files (see FILES)

The cartridge's .cfg file switches on everything needed when the cartridge is started from the Ultimate's file browser: Command Interface, REU (16 MB), Ultimate Audio and the U64 turbo registers. Started another way, the start-up applies the same settings itself when the REU is not 16 MB or the turbo registers are off (see *SETUP).

More: *DRIVE*, *MOUNT*, *MENU*, *REBOOT*, *SAVEMEM*, *CONFIG*, *NET*, *UPIC*, **PALETTE*.

***STATUS (also *SYS, SYS)**

*STATUS (also *SYS)
*STATUS SID the SIDs

Shows the system: MOS and cartridge, the Ultimate's hardware, CPU speed, REU size, BASIC's memory (PAGE, TOP, LOMEM, VARTOP, HIMEM, free bytes), open files, the current directory, the drives, the network address and the clock.

*STATUS SID lists the SIDs the Ultimate has set up: each one's address (a second one for a socket holding two SIDs) and whether it is an UltiSID or a chip in a socket. The U64 lists its UltiSIDs first.

***UVER**

*UVER

Shows the identification of the Ultimate's DOS and control targets.

***MENU (also *REBOOT, REBOOT)**

*MENU the Ultimate's menu
*REBOOT [Y] restart the C64

MENU opens the Ultimate's menu, as its freeze button does; the program goes on where it was when the menu is left. REBOOT restarts the C64 (and the cartridge): it asks "Reboot? (Y/N)" first unless Y is given. The program in memory is lost.

***SAVEMEM**

*SAVEMEM file 64K of RAM to a file

Ultimate 64 only: saves all 64K of the C64's RAM, as it is now, to a file (overwritten if it exists), for looking at memory on a PC. A name without a leading / is in the current directory.

***CONFIG (also *SETUP, SETUP)**

*CONFIG file Ultimate settings
*SETUP the settings for BASIC

CONFIG loads Ultimate settings from a .cfg file (the format of the Ultimate's own "Save settings": [section] lines, then item=value lines); only the items in the file change, for this session (they are not saved in the Ultimate's flash). Firmware 3.15 or later; on an SD card or USB stick put the file in a folder (/SD/cfg/my.cfg: firmware 3.15a cannot open one in the drive's root).

SETUP applies the settings this BASIC needs (the same as the u64bbc .cfg file next to the cartridge: command interface, REU of 16 MB, Ultimate Audio, the U64's turbo registers, the

SIDs), then finds the REU and the turbo again. The start-up does this by itself when the REU is not 16 MB or a U64 has its turbo registers off, and says so in one line.

11.2 Drives and disk images

***DRIVE**

*DRIVE A|B ON|OFF

Switches one of the Ultimate's drives on or off.

***MOUNT**

*MOUNT <image> [A|B|<device>]

Mounts a disk image (D64, D71, D81, G64 ...) on one of the Ultimate's drives, drive A if none is given.

*MOUNT /USB0/games/elite.d64

*MOUNT disk2.d64 B

***UNMOUNT**

*UNMOUNT [A|B|<device>]

Removes the disk image from a drive (drive A if none is given).

***SWAP**

*SWAP [A|B|<device>]

Moves a drive (drive A if none is given) on to the next disk of its disk set, as the Ultimate's swap function does for multi-disk games.

***PARTITION**

*PARTITION <n> <path> [<name>]

*PARTITION <n> OFF

Makes partition n of the Ultimate's SoftIEC drive point to a directory. From firmware 3.15 the SoftIEC drive reaches files only through partitions; without one it serves /Temp. OFF removes a partition (not effective on firmware 3.15a).

*PARTITION 3 /USB0/c64 GAMES

*CP 3

***PARTITIONS**

*PARTITIONS

Lists the SoftIEC drive's partitions (its "\$=P" directory).

***CP**

*CP <n>

Selects partition n on the SoftIEC drive (the drive's own CP command).

11.3 Clock and speed

***TIME**

*TIME

Prints the date and time of the Ultimate's clock. The BASIC variable TIME is a separate centisecond counter.

***SETTIME**

*SETTIME yyyy/mm/dd hh:mm:ss

Sets the Ultimate's clock. The Ultimate's settings can forbid this (error "Function prohibited").

***TURBO**

*TURBO show the speed index
*TURBO MAX fastest (index 15)
*TURBO OFF 1 MHz (index 0)
*TURBO <n> speed index 0-15

Needs the U64 turbo registers (Turbo Control in the Ultimate's settings, switched on by the cartridge's .cfg). Index 15 is 48 MHz on an Ultimate 64 and 64 MHz on an Elite II or C64 Ultimate. See SPEED.

SPEED On an Ultimate 64 or C64 Ultimate with the U64 turbo registers enabled, BBC BASIC starts at the highest speed of the machine: 48 MHz (Ultimate 64, Elite) or 64 MHz (Elite II, C64 Ultimate). That is 25 to 30 times a BBC Micro.

TIME stays in real time at every speed. *TURBO shows or changes the speed. On an Ultimate II+ or a plain C64 the speed is 1 MHz.

12. Memory

The C64's 64K hold BASIC, the MOS, the screens and your program; the REU holds what does not fit. This chapter shows where everything is and how to get more room for programs.

MEMORY Memory map while BASIC runs:

&0000-&0051	BASIC zero page
&0052-&006F	MOS zero page
&0070-&008F	free for machine code
&0090-&00FF	C64 KERNAL
&0100-&01FF	6502 stack
&0200-&021F	BBC MOS vectors
&02A7-&02FF	MOS error buffer
&033C-&03FB	MOS variables
&0400-&07FF	BASIC workspace
&0800-&08FF	MOS command buffer
&0900-&1CFF	sound engine
&1D00-&7FFF	program (PAGE-HIMEM)
&8000-&BFFF	BBC BASIC (RAM copy)
&C000-&C3FF	screen / colours
&C400-&C7FF	resident MOS
&C800-&CFFF	VDU core
&D000-&DFFF	I/O, graphics below
&E000-&FFFF	KERNAL, bitmap below

PAGE is &1D00 (as on a BBC Micro with ADFS) and HIMEM &8000: 25.75K for programs. *SOUND NONE* takes the sound engine out: a program may then set PAGE=&900 (31K). Some modes need RAM below HIMEM: MODE 7 keeps its screen at &7C00, the Upic Lite modes their picture at &5800 (MODE 4, 5, 6) or &3000 (MODE 1, 2); HIMEM is lowered to match, as on a BBC Micro. The graphics modes keep a further 768 bytes below that for the fast graphics commands (POINT, LINE, ...), *SPRITE ON 1.25K for the sprites. Under the I/O area is the drawing engine, under the KERNAL the bitmap (&E000) and the text font (&F000). The area &70-&8F is free for your own machine code, as on a BBC Micro.

Free for programs per mode (HIMEM - PAGE; on a 64 MHz Ultimate):

Mode	HIMEM	free	no sound
100, 134	&8000	24.75K	29.75K
7, 135	&7C00	23.75K	28.75K
0, 3	&7D00	24K	29K
103, 104	&7D00	24K	29K
110, 111	&7D00	24K	29K
128-133	&7D00	24K	29K
6	&5800	14.75K	19.75K
4, 5	&5500	14K	19K
1, 2	&2D00	4K	9K

“no sound”: *SOUND NONE*, then *PAGE=&900*. SPRITE ON (VIC-II modes only) takes 1.25K more. The Upic Lite modes (1, 2, 4, 5, 6) keep their picture below HIMEM; the graphics modes keep 768 bytes for the fast * graphics commands. 48 MHz Ultimates use the VIC-II versions of MODE 0, 1 and 4, and VICE of every mode: &7D00 there. Short of memory in MODE 1 or 2: MODE 129 or 130 (the VIC-II versions), or *SOUND NONE.

BASIC runs from RAM because the Ultimate 64 reads cartridge ROM at 1 MHz even in turbo mode.

MEMORY shows the map of the running program; MEMORY UPIC OFF (or *SHADOW) keeps HIMEM at &8000 in every mode but MODE 7.

***MEMORY**

```
*MEMORY          memory map
*MEMORY UPIC OFF  VIC-II modes
*MEMORY UPIC ON   Upic Lite
*MEMORY UPIC AUTO (as ON)
```

Shows PAGE, TOP, LOMEM, VARTOP, HIMEM, free memory and where the screen is.

On a 64 MHz Ultimate the BBC modes 0-6 use Upic Lite: exact 320x256 like a BBC Micro, with the picture below HIMEM (&5800 or &3000), as on the BBC. UPIC OFF makes the next MODE use the VIC-II version instead: 200 lines, but HIMEM stays &8000. The same as *SHADOW. MODE 7 always keeps its screen at &7C00.

PAGE

PAGE

The start of the program (&1D00). With *SOUND NONE it may be set as low as &900.

HIMEM

HIMEM

The top of BASIC's memory: &8000, lower in modes that need RAM below it (see MEMORY).

LOMEM

LOMEM

The start of the variables (after the program).

12.1 Snapshots

***SNAPSHOT**

```
*SNAPSHOT n      take (slot 1-9)
*SNAPSHOT LOAD n  back to it
*SNAPSHOT        list the slots
*SNAPSHOT SAVE n f slot to file f
*SNAPSHOT LOAD f  from file f
```

A snapshot keeps BASIC's world in the REU: the program, its variables, BASIC's workspace and zero page (not &52-&6F), the screen mode, with the time it was taken. Taking and restoring take a moment. Take it at the prompt; LOAD brings the program and the variables back as they were (the screen's content is not kept). PAGE must be the same as when it was taken.

```
*SNAPSHOT 1
NEW
*SNAPSHOT LOAD 1
```

Slots 1-9 lie at &B8000-&FFFFFF (*SRWRITE and OSWORD &42 stop below &90000) and go when the power goes off; SAVE copies a slot to a file on the Ultimate's drives, LOAD with a name reads one back. Needs an REU of 1 MB or more.

12.2 Strings and memory

BBC BASIC never gives string space back while a program runs (there is no garbage collection, on a BBC Micro neither). A string that grows beyond the space it had gets new space, and the old space stays used until the program ends (or CLEAR, RUN). Long-running programs that build strings can therefore end with "No room". What helps:

- Give a string its largest length once, at the start: `A$=STRING$(200,""):A$=""`. Later values up to 200 characters reuse that space.
- Avoid growing a string a character at a time in a loop (`A$=A$+CHR$(C%)`) when other strings are made in between; build it in memory with `?` and `$` instead (`DIM B% 255`, then `B%?I%=C%` and `$B%`).
- Keep large data out of BASIC's memory: in the REU (*SRWRITE, *SRREAD: chapter 7), read back when needed. The demo U64TEXT keeps its news and blog texts there.
- `PRINT ~(?2+256*?3), ~(?4+256*?5)` shows the end of the variables and BASIC's stack: the room between them is what is left.

12.3 The REU

REU address	Used for
bank 0, \$0000-\$17FF	fonts (bitmap modes, text screen, start-up font)
bank 0, \$1830-\$18BF	palettes
bank 0, \$2000-\$30CF	instruments, envelopes
bank 0, \$3100-\$34FF	network channels' buffers
bank 0, \$3500-\$3501	the line editor's history: head, count
bank 0, \$4000-\$7FFF	HTTP replies (*NET GET, *JSON GET, http:// channels)
bank 0, 8000-FFFF	the line editor's history (127 lines)
banks 1-3	the song (*SONG)
&40000-&8FFFF	programs' data (*SRWRITE, OSWORD &42)
&90000-&9FFFF	the Upic full picture (*UPIC, MODE 110/111)
&A0000-&AFFFF	BASIC's RAM while the Upic picture is shown
&B0000-&FFFFFF	snapshots: working slot 0, slots 1-9 (*SNAPSHOT)
&100000 up	the RAM disk /REU

13. MOS calls

The MOS entry points have the BBC's low bytes in page &C7 (OSWRCH is &C7EE where a BBC has &FFEE); the RAM vectors are at &0200 as on the BBC. Programs call them with CALL or USR (A%, X%, Y% are passed in the registers).

MOS The MOS (machine operating system) entry points, at BBC Micro low bytes in page &C7:

OSFIND &C7CE	OSGBPB &C7D1
OSBPUT &C7D4	OSBGET &C7D7
OSARGS &C7DA	OSFILE &C7DD
OSRDCH &C7E0	OSASCI &C7E3
OSNEWL &C7E7	OSWRCH &C7EE
OSWORD &C7F1	OSBYTE &C7F4
OSCLI &C7F7	

The RAM vectors are at &0200-&021F as on a BBC Micro (BRKV &0202, WRCHV &020E, ...). See OSBYTE, OSWORD and VDUCODES.

OSBYTE OSBYTE calls (*FX a,x,y) available:

&00	OS version (X=1)
&0F	flush keyboard buffer
&15	flush keyboard buffer
&72	shadow for next MODE
&75	VDU status (4: shadow)
&7C	clear ESCAPE
&7D	set ESCAPE
&7E	acknowledge ESCAPE
&7F	EOF# of channel X
&81	INKEY
&82	high address (&FFFF)
&83	OSHWM / PAGE (&1D00)
&84	HIMEM of the mode
&85	HIMEM for a mode
&86	POS (X) and VPOS (Y)
&87	character at cursor (all modes)
&04	cursor keys (kept)
&0B	key repeat (0 off)
&0C	key repeat (0 default)
&13	wait for the frame's end
&80	ADVAL
&A3	GXR (accepted)
&C8	bit 0: ESCAPE off
&CA	keyboard status (CAPS LOCK)
&DC	escape character
&DA	VDU queue length
&E5	ESCAPE key off (X<>0)

Other calls are ignored. Key repeat uses the KERNAL's speed: *FX 11,n* and *FX 12,n* with *n* > 0 make every key repeat. *FX 220,c makes the key typing ASCII *c* set ESCAPE (27: only RUN/STOP).

OSWORD OSWORD calls available: 0 reads a line (the line editor, *EDIT*), 1 reads *TIME*, 2 sets *TIME* (5 bytes, centiseconds), 9 reads a pixel (*POINT*), 10 reads a character definition (block +0 = character; +1 to +8 = rows), 14 reads the Ultimate's clock, &42 copies between memory and the REU (block +0: &80 to the REU, 0 from it; +1/+2 address; +5/+6 length; +7 the REU's 64K bank (4-255); +8/+9 address in the bank; as the Master's sideways RAM transfer), &70 reads a value of the JSON reply JSON GET keeps (see JSON), &71 moves sprites (see SPRITE). Others are ignored.

OSWORD 14 as on a BBC Master: put 0 in the first byte of a 25-byte block for the text "Wed,07 Oct 2026.08:42:55" and a CR, or 1 for 7 BCD bytes: year (2 digits), month, day, day of week (1 = Sunday), hour, minute, second.

```
DIM B% 25: ?B%=0: A%=14
X%=B% MOD 256: Y%=B% DIV 256
CALL &C7F1: PRINT $B%
```

***FX**

*FX a[,x[,y]]

Calls OSBYTE a with X=x and Y=y. See OSBYTE.

*FX 15 flush the keyboard buffer

***ADVAL**

*ADVAL list the ports

*ADVAL JOY [p] joysticks (both)

*ADVAL PADDLE [p] paddles (port 1)

*ADVAL MOUSE [p] 1351 (port 1)

Chooses what ADVAL reads from a C64 control port. A joystick gives 65520, 32768 or 0 per axis. Paddles give their position: paddle A is the x channel, paddle B the y channel, their buttons the port's fire. A 1351 mouse (proportional mode) moves a position that starts in the middle: right and up make x lower and y higher, as a BBC joystick; its left button is fire. Read a mouse often: more than 31 counts between two reads are lost. Port 2 is BBC joystick 1 (ADVAL 1, 2), port 1 joystick 2 (ADVAL 3, 4).

*ADVAL MOUSE 1

14. Developing from a PC

Programs can be written on a PC and run on the machine over the network: with the VS Code extension (vscode/, Ctrl+F5 runs the file), with tools/u64remote.py on the command line, or by an AI coding agent through the MCP server tools/u64mcp.py (docs/AI_AGENTS.md). A line sent from the PC drops a half-typed line on the machine (chapter 5).

REMOTE Programs can be written on a PC and run here: the MOS has a mailbox that a PC (or an AI coding agent) reaches through the Ultimate's network API. At the > prompt the MOS accepts lines and programs from it as if typed, and copies the output back.

```
tools/u64remote.py run prog.bas
tools/u64remote.py exec "PRINT 6*7"
```

AI agents use the same through the MCP server tools/u64mcp.py. Nothing changes for the user at the keyboard. See docs/REMOTE_DEV.md and docs/AI_AGENTS.md.

DEBUG Programs sent from a PC can be debugged there: breakpoints, single steps and a view of the variables (VS Code: F5; command line: tools/u64remote.py debug; AI agents: bbc_debug).

It uses TRACE: BASIC reports each line before it runs and the MOS pauses at a breakpoint. The line after a FOR or REPEAT is reported only on the first pass (NEXT and UNTIL jump back into the loop line), so for later passes put the breakpoint on the NEXT or UNTIL line. See REMOTE.

15. Limits

What this BASIC can and cannot do, and how fast it is. The error messages are in chapter 6 (6.5).

LIMITS What this BASIC can and cannot do, compared with a BBC Micro:

Typed lines: 238 characters (78 with *EDIT C64*); *programs sent from a PC (see REMOTE) may have lines up to 251 characters*. Memory: PAGE &1D00 (&900 with SOUND NONE), HIMEM &8000 in most modes (less in Upic Lite MODE 1 and 2: see MODE); LOAD and SAVE work between PAGE and &7FFF. Files: two open at a time; OSGBP is not there (BGET# and

BPUT# are). Graphics: points, lines, triangles, rectangles, parallelograms, circles, ellipses, pattern fills, flood fill, arcs, segments, sectors, as on a BBC Micro with the GXR, plus the fast * graphics commands. Sound: SOUND and ENVELOPE as on a BBC Micro, plus instruments on 24 SID voices, Ultimate Audio samples and MOD music. * commands take numbers or the variables A% to Z%, not expressions (use OSCLI with STR\$ for those). BBC BASIC 3.10 itself has no array parameters for PROC and FN, and MODE is refused inside a PROC or FN when it would move HIMEM. Not there: the BBC's hardware at &FE00 and BBC machine code (see BBC).