Name

BBL — MS DOS Device driver to display in **B**ig **B**lock **L**etters

Description

BBL is an installable software device driver for MS DOS that displays on the screen, in big block letters, any characters sent to it. It is a memory-resident (800 bytes of memory) program that can be used by any program or utility as an output device, much as `CON:` might be used for input or output. However, BBL is an *output-only* device: attempts to read from it will return no result.

Except for the few characters used for command and position control initiation, all of the 256-character display set is available for display.

Once loaded, BBL is dormant except when it receives a character for writing. It then diplays the character in one of three *display modes* magnified 8-, 16-, or 24-fold. BBL does not connect to the interrupt system.

BBL wraps to a new line any characters it receives when its virtual cursor is past the end of the current line. It scrolls when the subsequent line would not fit on the screen.

BBL supports backspace, carriage-return, form-feed, and line-feed characters in the same way as standard console output.

Commands for absolute cursor positioning and display-mode control are transmitted to BBL via the character stream.

To avoid confusing situations that can arise when a device driver has a simple name, the name registered with DOS is `BBL-DRVR` — that is the name that should be used to open the device for writing. When this manual refers to BBL, that is the device-driver name that is implied.

Use of BBL does not interfere with and is not affected by other console display use except that the screen may become cluttered with characters from the alternate displays.

Use

BBL is loaded at boot time through the `CONFIG.SYS` file (see *Installation*). It remains dormant until characters are written to it.

To initiate writing in a high-level language, open device `BBL-DRVR` using the appropriate procedure in that language (e.g., `fopen` in `C`). Then write characters to that device to have them displayed in block letters on the screen. Control commands are sequences of characters that cause particular actions. No other operations are possible.

Commands

Commands are limited to cursor-control commands and commands to set the display modes (magnification) on the screen. In version 1 of BBL, the commands available, invoked by the incoming character stream, are:

⟨**FF**⟩ clear screen & move cursor to home

⟨**BS**⟩ move cursor back one character position (same line)

⟨**CR**⟩ move cursor to beginning of line

⟨**LF**⟩ move cursor down one line (or scroll up)

⟨**VT**⟩⟨**digit**⟩ position cursor on line ⟨digit⟩; e.g. ⟨VT⟩1 positions cursor on line 1, no change in column position

⟨**HT**⟩⟨**digit**⟩ position cursor on column ⟨digit⟩; e.g., ⟨HT⟩3 positions cursor on column 3, no change in line position

⟨**SO**⟩⟨**digit**⟩ with ⟨digit⟩ = [0|1|2] means set display mode to 0 (3rows x 10char/row), 1 (1x5) or 2 (1x3).

These are described in greater detail in subsequent sections.

Display Modes

BBL has three display modes: 3x10, 1x5, and 1x3. These modes describe the number of lines and number of characters per line for the three modes.

Mode 0
is the 3x10 virtual display mode. The characters are eight physical screen rows high and eight screen characters wide.

Mode 1
is the default startup mode and is the 1x5 virtual display mode. The characters are sixteen screen rows high and sixteen screen characters wide.

Mode 2
is the 1x3 display mode. The characters are twenty-four screen rows high and twenty-four screen characters high.

The display mode defaults to mode 1, 1x5, on startup but can be changed by a two-character sequence of the form `<SO>[0 | 1 | 2]`, where `SO` is character 14 (decimal) = 0e (hex) in the ASCII character set and the ASCII character '0', '1' or '2' is used to select the mode. In other words, the `SO` character is the Mode-Change command character, and the digit following selects the mode. The screen is cleared on mode-change.

Cursor Control

Screen virtual display positions are numbered with (0,0) as the upper left-hand corner of the display. Row-position addresses increase down the screen; column-position addresses increase to the right from the left edge.

Under normal use, the display scrolls up when the cursor is on the bottom line of the screen and a line-feed (LF = 10 decimal = 0ah hex) is received or a wrap is needed. A wrap from the last column position to the first is automatic when the character to be written would disappear off the right side of the screen — and a line-wrap includes both a carriage return (CR = 13 decimal = 0c hex) and a line-feed.

Upon receipt of a form-feed character (FF = 12 decimal = 0c hex), the screen is cleared and the virtual cursor is moved to the (0,0) [upper left-hand] position.

Upon receipt of a backspace character (BS = 8), the character moves to the left one column. If the cursor is at the left edge of the screen upon receipt of the BS, the cursor remains at column 0 (it does not wrap back to the end of the previous line).

The LF character advances the cursor one row, scrolling if necessary, but does not affect column position.

The CR character returns the cursor to column zero, but does not affect the row position.

The cursor may be moved to an absolute virtual-display position by explicit command. The horizontal position is set with the command sequence ⟨HT⟩⟨*digit*⟩, where HT is the Horizontal Tab ASCII character, position 9 in the character set. The vertical position is set with the command sequence ⟨VT⟩⟨*digit*⟩, where VT is the Vertical Tab ASCII character, position 11 decimal = 0b hex in the character set. If the address of the position requested exceeds the display positions available, BBL issues a BEL character (beeps) and disregards the command entirely.

Error Handling

In case of errors, BBL issues a BEL and disregards the command being processed. For example, attempting to set horizontal position with the command characters HT and 'A' will result in a beep, and both characters will be ignored.

Installation

In the DOS CONFIG.SYS file, insert the following line

```
device=c:bbl.sys <options list>
```

and copy `bbl.sys` from the distribution file area or floppy to c:.

Command-line options are specified by a '/' character followed by keyword, '=', and value. Options include:

/FONT
to guide `BBL` in locating the 8x8 character set in the display-adapter ROM.

EGA
`/FONT=EGA` (which is the default) specifies that the font can be located by using standard EGA-adapter BIOS calls.

VGA
`/FONT=VGA` (which uses the same technique as EGA) specifies that the font can be located by using standard VGA-adapter BIOS calls.

ATT
`/FONT=ATT` specifies that the display adapter is an AT&T 6300 adapter, for which the 8x8 font ROM is at a known location.

SSSS:OOOO
`/FONT=SSSS:OOOO` specifies the segment and offset address (hexadecimal digits) of the 8x8 font ROM address. For example, for the Toshiba T1000 laptop, `/FONT=F000:FA6E` points `BBL` to the ROM 8x8 font.

/MODE
specifies the startup mode for the virtual display. The form is `/MODE=<digit>`, where `digit` is 0, 1, or 2 to specify the desired mode. The default mode is 1.

After installation (and rebooting), run the verification program `BBL-TEST` to verify that `BBL` is operating correctly. If not, the problem is probably in the location of the font ROM as assumed by `BBL` or specified in your command-line option. Verify that you have specified the correct video adapter or tell `BBL` explicitly where the font ROM is located.

Locating the ROM Font

If you need to locate the ROM 8x8 font for your computer, check the BIOS calls (if there is one) to see if there is a call to return the font pointer.

You may find it equally easy to use the DOS `debug` utility to search for the font. The character '1' seems to have a characteristic pattern that can be sought using the `debug` search facility. Run `debug` and then give the command

```
-s F000:0000 FFFE 30 70 30 30 30 30 FC 00
```

to search segment F000, offsets 0000 to FFFE, for the pattern of bytes 30 ... 00. That pattern of bytes seems to be common for the character '1'. If `debug` finds that pattern, it will return the address. Subtract 180 hex from the offset to obtain the beginning of the 8x8 256-character ROM font. Use the segment address reported by `debug` and the computed offset as the `SSSS:0000` argument to the /FONT option on the command line.

Further Developments

An obvious and easy addition would be to provide command sequences to set colors for text and background.

The 8x8 base ROM font used for characters are readable at a distance, but better (e.g., 9x14 or better) fonts are available in EGA/VGA adapters. Higher-resolution modes would permit, for example, using 16x16 character cells on 43x132 screens (2 rows x 8 cols or 1x4 of virtual display). The resulting displays would be more attractive. The code for this would probably not be difficult, but the code would require testing on many different EGA/VGA adapters; the current version has the advantage of fairly general portability (I hope).

Author, Copyright, and Distribution

BBL was written by David Todd, Chemistry Department and Computing Center, Wesleyan University, August, 1992; for John Sease, Chemistry Department. Author's address:


```
H. David Todd
Computing Center
Wesleyan University
Middletown, CT  06459
email via Internet: hdtodd@eagle.wesleyan.edu
```

- Charge fees or ask donations in exchange for copies of the software.
- Distribute the software with commercial products without the written permission from the author and copyright owner.
- Remove author or copyright information from the software and documentation.

The following QBASIC program demonstrates how to open BBL as a device and write two lines (default mode 1; the first line will be scrolled up).

```
/* EXAMPLE 1 */
    'Demonstrate use of BBL from a BASIC program

    OPEN "bbl-drvr" FOR OUTPUT AS #1
    mode$ = CHR$(14)      'mode introducer
    vt$ = CHR$(11)        'vertical position introducer
    ht$ = CHR$(9)         'horizontal position introducer
    bs$ = CHR$(8)         'backspace
    cr$ = CHR$(13)        'carriage return
    lf$ = CHR$(10)        'line feed

    PRINT #1, mode$; "0"    'set to mode 0 and write out two lines
    PRINT #1, "12.34"
    PRINT #1, "56.78"

    PRINT #1, mode$; "1"    'just one line in mode 1
    PRINT #1, "12.34"

    PRINT #1, mode$; "2"    ' ... and in mode 2, only 3 chars wide
    PRINT #1, "012"

    start = TIMER
    PRINT #1, mode$; "1"       'in mode 1 ...
    FOR i = 0 TO 99
    PRINT #1, ht$; "1"         'go to position 1
    PRINT #1, USING "###"; i   'and write an integer
    NEXT i
    endt = TIMER
    PRINT "Elapsed time for 100 iterations = "; endt - start; "sec"

    END
```

The following example is a fragment from a Turbo C program that serves as the testing program for BBL.

```
/* EXAMPLE 2 */
/*  Turbo C program to test use of BBL device driver */

#include        <stdio.h>
#include        <conio.h>
#include        <dos.h>

FILE *out;
void pause(void);
void mode(int m);
void hpos(int col);
void vpos(int row);

int main(void)
{
        printf("Program to test BBL device driver\n");
        printf("When paused, press any key to continue\n\n");
        if ( (out=fopen("BBL-DRVR","wt")) == NULL) {
                fprintf(stderr, "Cannot open output file.\n");
                return 1;
        }

// mode 0 wrap
        clrscr();
        printf("One long string, wraps to three lines in mode 1");
        pause();
        clrscr();
        mode(0);
        fputs("This line should wrap",out);
        pause();

// wrap in mode 1  --- scrolls
        clrscr();
        printf("One string (pause in middle) scrolls in mode 1");
        pause();
        clrscr();
        mode(1);
        fputs("Wrap ",out);
        pause();
```

```
        fputs("Scrol",out);
        pause();

// wrap in mode 2 --- scrolls
        clrscr();
        printf("One string (pause in middle) scrolls in mode 2");
        pause();
        clrscr();
        mode(2);
        fputs("Wrp",out);
        pause();
        fputs("Scl",out);
        pause();

        return 0;
}

void mode(int m)
{
        fputc('\x0e',out); fputc('0'+m,out);
}

void vpos(int row)
{
        fputc('\v',out); fputc('0'+row,out);
}

void hpos(int col)
{
        fputc('\t',out); fputc('0'+col,out);
}

void pause(void)
{
        fflush(out);
        while (!kbhit()) ;
        getch();
}
```