DECADE ENGINEERING

XBOB-4 Quick-Start Guide & POS Tutorial

04 June 2015

*See **www.decadenet.com** for the latest revision of this document.*

Introduction

XBOB-4 prints information from RS-232 serial data sources to ordinary NTSC and PAL TV monitors. From the viewpoint of the data generator, XBOB-4 behaves much like a conventional serial printer.

The first couple of pages in this document walk you through the steps to establish communication between a PC and XBOB-4, so you can immediately type text onto the TV screen and begin using commands described in the XBOB-4 Application Guide.

In the second section, special attention is given to challenges often encountered by Point-of-Sale (POS) video security equipment installers. This material is highly relevant to many applications where a pre-existing data source cannot be custom-programmed for best results with XBOB-4.

Preparation

A video signal source at the video input to XBOB-4 is not strictly required, because XBOB-4 can generate video on-board, but you should have one if you wish to investigate video overlay performance. Cameras and DVD players with composite video output (yellow RCA jack) generally work well. In any case, a TV monitor or TV receiver with video input must be connected to the video output jack on XBOB-4. It's ideal to have access to more than one TV monitor, because some flat-screen models can be quirky.

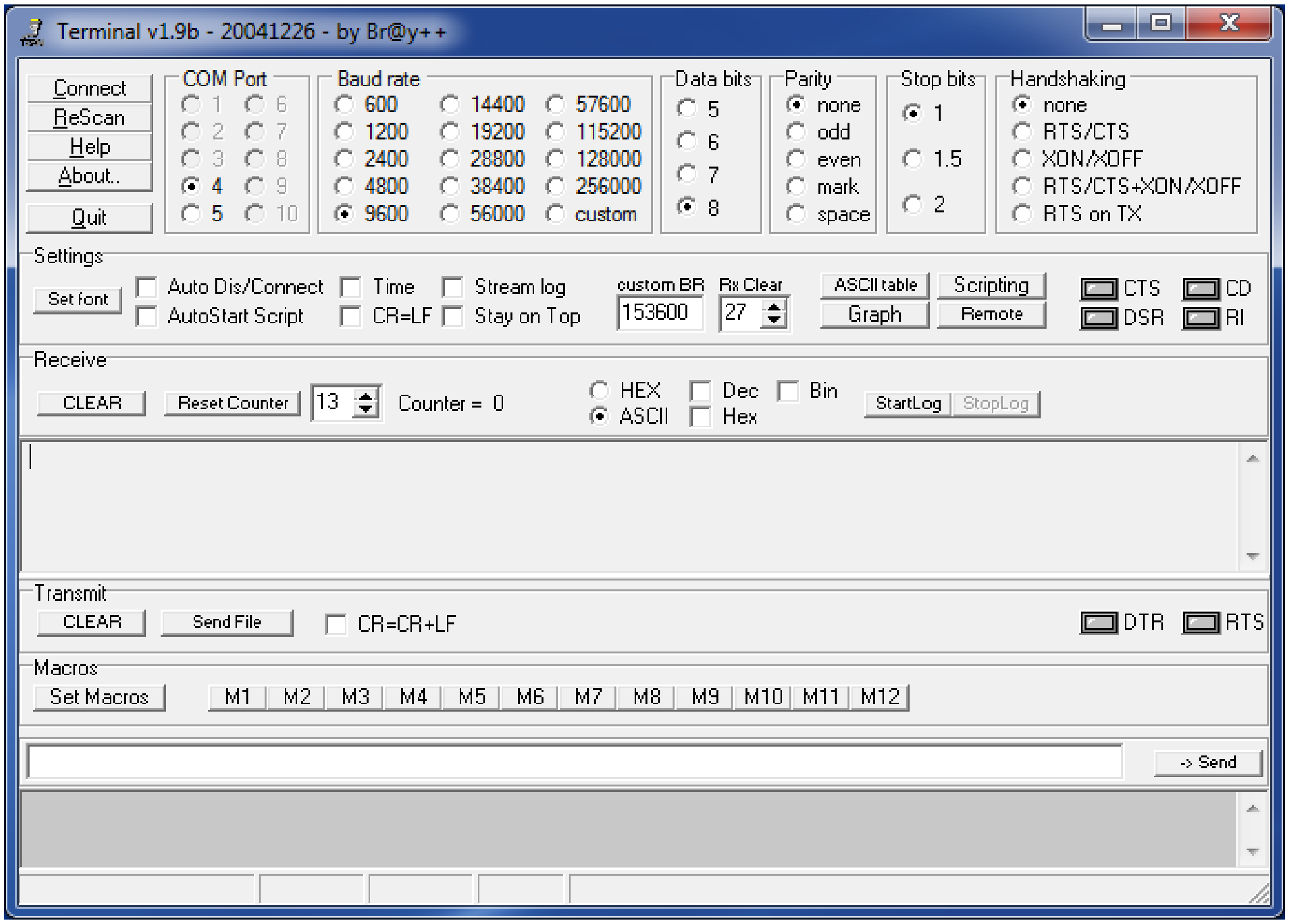
XBOB-4 normally ships with its default baud rate and other communication parameters matching the terminal configuration instructions below. If it's shipped from Decade Engineering configured differently, then a custom configuration sticker will be found on the cabinet bottom and this information should be used instead for terminal setup.

The XBOB-4 data port pin assignment is for a standard male/female 9-pin modem cable with all pins wired straight through to a PC COM port. This hookup will not work with null-modem cables. To follow this guide closely, use a Windows PC with available serial COM port or USB/Serial adapter, and Bray's Terminal software. I/O Gear USB adapters such as model GUC232A are recommended, but most adapter brands are acceptable for this exercise. Bray's Terminal is a free download and provides a useful macro capability.

Power up the TV monitor and select the correct video input if necessary. Make sure that XBOB-4 is connected to a PC serial COM port, via USB adapter if necessary, and power it up. A big "BOB-4" logo will appear on the TV monitor unless the factory default bootscript has been cleared.

Using Bray's Terminal

We're going to use what is arguably the simplest and most direct method to make XBOB-4 print text on the TV monitor: a terminal emulation program running on your PC. Terminal programs make the PC behave like a dumb serial terminal — once the standard user interface for central 'mainframe' computers. These exercises were developed with Bray's Terminal V1.9B running on a Win7 Pro 64-bit machine. Terminal doesn't require 'installation.' Just place the downloaded .exe file in a convenient location and launch it. Depending on version, the Terminal window looks something like this:


Terminal v1.9b - 20041226 - by Br@y++

COM Port: 1 2 3 [✓ 4] 5 6 7 8 9 10

Baud rate: 600 1200 2400 4800 [✓ 9600] 14400 19200 28800 38400 56000 57600 115200 128000 256000 custom

Data bits: 5 6 7 [✓ 8]

Parity: [✓ none] odd even mark space

Stop bits: [✓ 1] 1.5 2

Handshaking: [✓ none] RTS/CTS XON/XOFF RTS/CTS+XON/XOFF RTS on TX

custom BR: 153600

Rx Clear: 27

Receive: 13 Counter = 0 HEX [✓ ASCII]

Terminal should find available COM Ports automatically. If the port number buttons are all gray, confirm that another program isn't using the desired port and click the ReScan button. As shown above, select an available COM Port, set Baud Rate to 9600, Data bits to 8, Parity to none, Stop bits to 1, and Handshaking to none. Click **Connect** – the same button should now say **Disconnect**. The **CTS** indicator turns green at this time, but it's safely ignored.

In case your PC has assigned a COM port number beyond Terminal's range, open Device Manager > Ports (COM & LPT), and double-click the desired port. A window titled "USB Serial Port (COMx) Properties" should appear. Click the "Port Settings" tab and then click the "Advanced…" button. In the COM Port Number drop-down menu, select an available port assignment within Terminal's range and then click "OK." Click the "ReScan" button in Terminal, which should now recognize the modified port.

To confirm Terminal communication with XBOB-4, type some text into the **Send** edit box and click the **Send** button. If your hookup is working, the big BOB-4 logo clears and your text immediately prints with the default 12x13 font in the upper left corner of the TV screen. Garbage printing often indicates a baud rate mismatch.

Now let's learn to transmit commands to XBOB-4. One of the most common application requirements is to clear the screen. XBOB-4 obediently does this upon receipt of the **J** command. Here's the **J** command syntax statement as it appears in the XBOB-4 Application Guide:

```
<CSI>nJ
```

<CSI> stands for Control Sequence Introducer, which must be entered as "**$1b[**" or "**$1B[**" in Terminal. 1B is the ASCII Escape code as a hexadecimal number, while the dollar sign commands translation into ASCII. Other terminal emulators may handle this issue differently. In Bray's Terminal's **Send** edit box, a complete **J** command looks like:

```
$1b[2J
```

The **n** parameter in the syntax statement was given a value of 2 in order to clear the entire screen instead of a portion. Click **Send** to transmit the command. If you send another string of printable text after clearing the screen, you will see that print position was also restored to the upper left corner by this version of the **J** command. See the **J** command entry in the XBOB-4 Commands section of the App Guide for details.

Carriage-Return <CR> and Line-Feed <LF> codes may be embedded in printable text sent via Terminal in the same way that Escape <ESC> was entered, that is, <CR> is entered as **`$0D`**, and <LF> is entered as **`$0A`**. Try them out!

It's often useful to review XBOB-4's current configuration. For that purpose, send

```
$1b[5}
```

Yup, the right-brace is another command specifier. This command also provides a convenient way to test outbound communication. The lengthy configuration report scrolls beyond the receive window in Terminal, but you can easily scroll back to the beginning if desired. Here's an example report:

```
BOB4 (software v4.3.5, boot v4.3.1, logic rev.22, board v4.2)

Running parameters:
 video mode: local NTSC non-interlaced
 pixel rate: 9.375 MHz
 first pixel position: 123
 last pixel position: 539
 pixels per line: 416
 first line: 39
 last line: 246
 lines per frame: 208
 SPI memory device 0: none (0 bytes)
 SPI memory device 1: none (0 bytes)
 SPI memory device 2: none (0 bytes)
 SPI memory device 3: none (0 bytes)
 slave SPI mode: 0
 comms ACR pins: 0x9 (SPI master, rate = conf)

Configuration:
 NTSC (16=0)
 interlaced: no (17=0)
 external video mode: automatic (18=3)
 external video mode follow: yes (19=1)
 high pixel rate: yes (20=1)
 underscan restriction: yes (21=1)
 frame buffer size: 3 (34560 bytes) (22=3)
 pixel rate: 0 MHz (23=0)
 horizontal position: 0 (24=0)
 horizontal size: 0 (25=0)
 vertical position: 0 (26=0)
 early lines: 0 (36=0)
 vertical size: 0 (27=0)
 overlay enabled: yes (28=1)
 local horizontal adjustment: 54 (29=54)
 NTSC chroma rate: 100.000 % (30=100000)
 PAL chroma rate: 100.000 % (31=100000)
 blinking enabled: no (32=0)
 blink period: 10 (33=10)
 blink duty: 50 (34=50)
 screen flip: normal (35=0)
 BOB-4 only mode: yes (9=1)
 STX allowed for string start: no (10=0)
 CR/LF mode: not set (11=0)
 config baud rate: 9600 (40=9600)
 flow control: XON/OFF (41=1)
 8 bits, 1 stop bit, no parity, echo mode off (42=4)
 slave SPI signaling mode: (pin configured) (44=4)
 boot script size: 13 bytes (8=13)
 l0: low (48=0)
 l1: low (49=0)
 l2: low (50=0)
 l3: low (51=0)
 l4: low (52=0)
 SPI memory device 0: none (rate=8) (56=0)
 SPI memory device 1: none (rate=8) (57=0)
 SPI memory device 2: none (rate=8) (58=0)
 SPI memory device 3: none (rate=8) (59=0)

Fonts:
   0: bob3          304 chars  12 x 13   2 bpp  internal
   1: 8x13           96 chars  8 x 13    2 bpp  internal
   2: target         14 chars  13 x 13   2 bpp  internal
   3: misc           96 chars  8 x 14    1 bpp  internal
   4: 6x10           96 chars  6 x 10    2 bpp  internal
   5: 13x34          96 chars  13 x 34   2 bpp  internal
   6: 20x40          96 chars  20 x 40   2 bpp  internal
   7: bob4            1 chars  136 x 33  2 bpp  internal
```

This information is highly condensed. Configuration statements such as “16=0” mean that configuration item 16 is set to zero, making XBOB-4 compatible with NTSC video rather than PAL video. All of the corresponding **v** commands are listed and described in the App Guide by parameter **n** value, e.g. 16. The example setting could have resulted from a user command entered like this, where parameter **m** is zero:

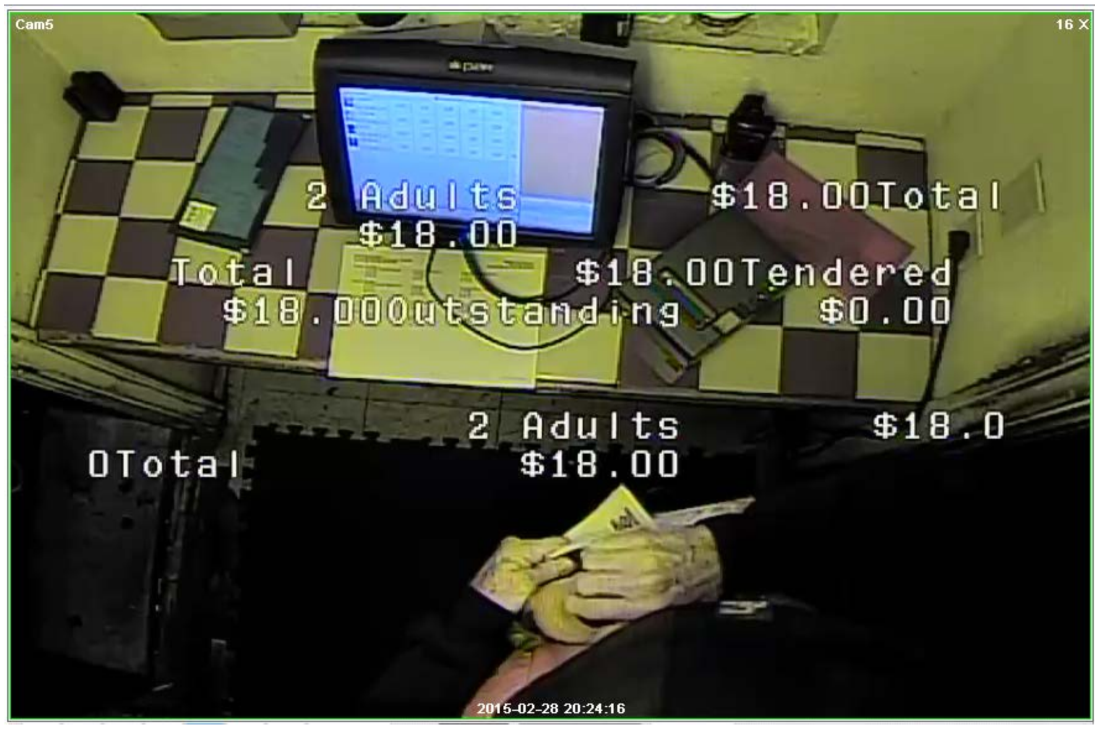
```
$1b[16;0v
```

Many of the **v** commands don’t take effect until they’re saved into flash memory and XBOB-4 is rebooted. The configuration save command may be issued just once after a salvo of configuration setting commands. Here it is:

```
$1b[1v
```

POS Application Hints

When XBOB-4 is attached to existing equipment, for example in a Point-of-Sale (POS) video security system where installers have little control over the data source, data display formatting can be problematic. There's usually no way to reprogram the data source equipment, so what's a body to do? Here's an example of one customer's discouraging initial result:

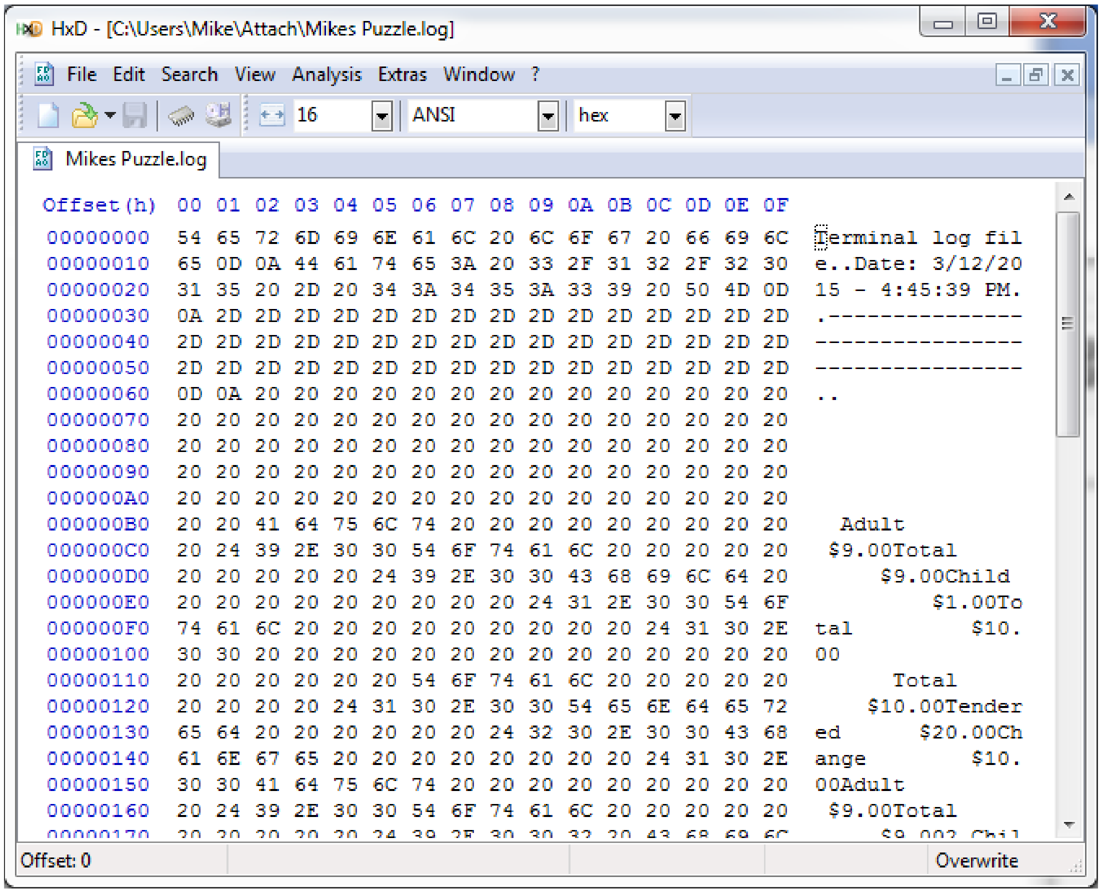


In messy cases like this, it's best to capture a sample of the data flowing from the POS terminal to XBOB-4. Display formatting tactics often become obvious when it's possible to analyze the data stream in detail. Terminal's logging feature provides a convenient way to accomplish this. To use it, connect the data source directly to the PC running Terminal. Click the **StartLog** button, give the log file a name, then trigger a few data generation events, and finally click **StopLog**. The resulting log file can be opened with any text editor. For a POS system, it might look like this example:

```
Terminal log file
Date: 3/12/2015 - 4:45:39 PM
-----------------------------------------------

Adult          $9.00Total         $9.00Child         $1.00Total
$10.00                    Total        $10.00Tendered
$20.00Change        $10.00Adult         $9.00Total         $9.002
Child       $2.00Total        $11.00                    Total
$11.00Tendered      $11.00Outstanding    $0.00
-----------------------------------------------
Date: 3/12/2015 - 4:59:54 PM
End log file
```

We notice right away that Carriage Returns <CR> and Line Feeds <LF> seem not to be present in captured data. Like Escape <ESC>, these non-printing control codes are interpreted by the display device. To be certain that we're seeing everything in the data stream, let's open the log file with a hex editor such as HxD:

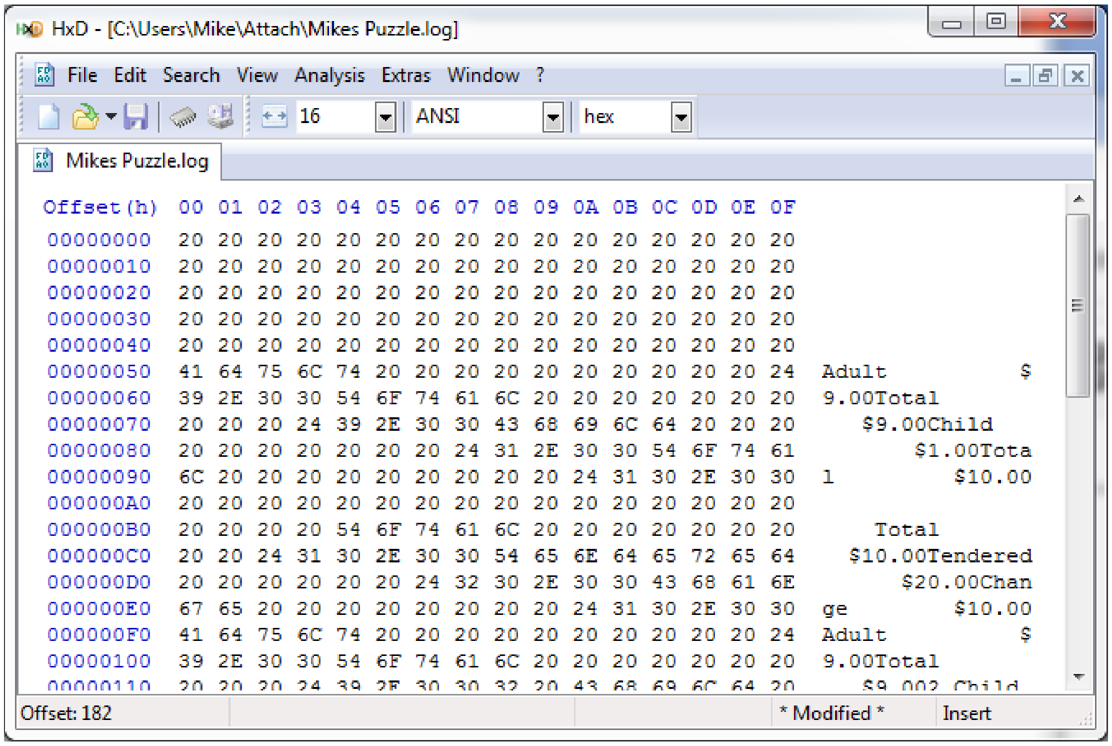

HxD - [C:\Users\Mike\Attach\Mikes Puzzle.log]
File Edit Search View Analysis Extras Window ?
16
ANSI
hex
Mikes Puzzle.log

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  54 65 72 6D 69 6E 61 6C 20 6C 6F 67 20 66 69 6C  Terminal log fil
00000010  65 0D 0A 44 61 74 65 3A 20 33 2F 31 32 2F 32 30  e..Date: 3/12/20
00000020  31 35 20 2D 20 34 3A 34 35 3A 33 39 20 50 4D 0D  15 - 4:45:39 PM.
00000030  0A 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D  .---------------
00000040  2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D  ----------------
00000050  2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D  ----------------
00000060  0D 0A 20 20 20 20 20 20 20 20 20 20 20 20 20 20  ..
00000070  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000080  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000090  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000A0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000B0  20 20 41 64 75 6C 74 20 20 20 20 20 20 20 20 20    Adult
000000C0  20 24 39 2E 30 30 54 6F 74 61 6C 20 20 20 20 20   $9.00Total
000000D0  20 20 20 20 20 24 39 2E 30 30 43 68 69 6C 64 20       $9.00Child
000000E0  20 20 20 20 20 20 20 20 20 24 31 2E 30 30 54 6F           $1.00To
000000F0  74 61 6C 20 20 20 20 20 20 20 20 20 24 31 30 2E  tal         $10.
00000100  30 30 20 20 20 20 20 20 20 20 20 20 20 20 20 20  00
00000110  20 20 20 20 20 20 54 6F 74 61 6C 20 20 20 20 20        Total
00000120  20 20 20 20 24 31 30 2E 30 30 54 65 6E 64 65 72      $10.00Tender
00000130  65 64 20 20 20 20 20 20 24 32 30 2E 30 30 43 68  ed      $20.00Ch
00000140  61 6E 67 65 20 20 20 20 20 20 20 20 24 31 30 2E  ange        $10.
00000150  30 30 41 64 75 6C 74 20 20 20 20 20 20 20 20 20  00Adult
00000160  20 24 39 2E 30 30 54 6F 74 61 6C 20 20 20 20 20   $9.00Total
00000170  20 20 20 20 20 24 39 2E 30 30 32 20 43 68 69 6C       $9.002 Chil
```

Offset: 0 Overwrite

At this point, it's a good idea to strip out header and footer data that was inserted by Bray's Terminal. Note that the 0D and 0A codes at offset 60 are part of the added header, so they have to go, along with similar additions in the footer. Now it's easy to confirm that <CR> and <LF> don't occur in captured data by searching for them or simply observing that hex codes 0D and 0A are nowhere to be found:

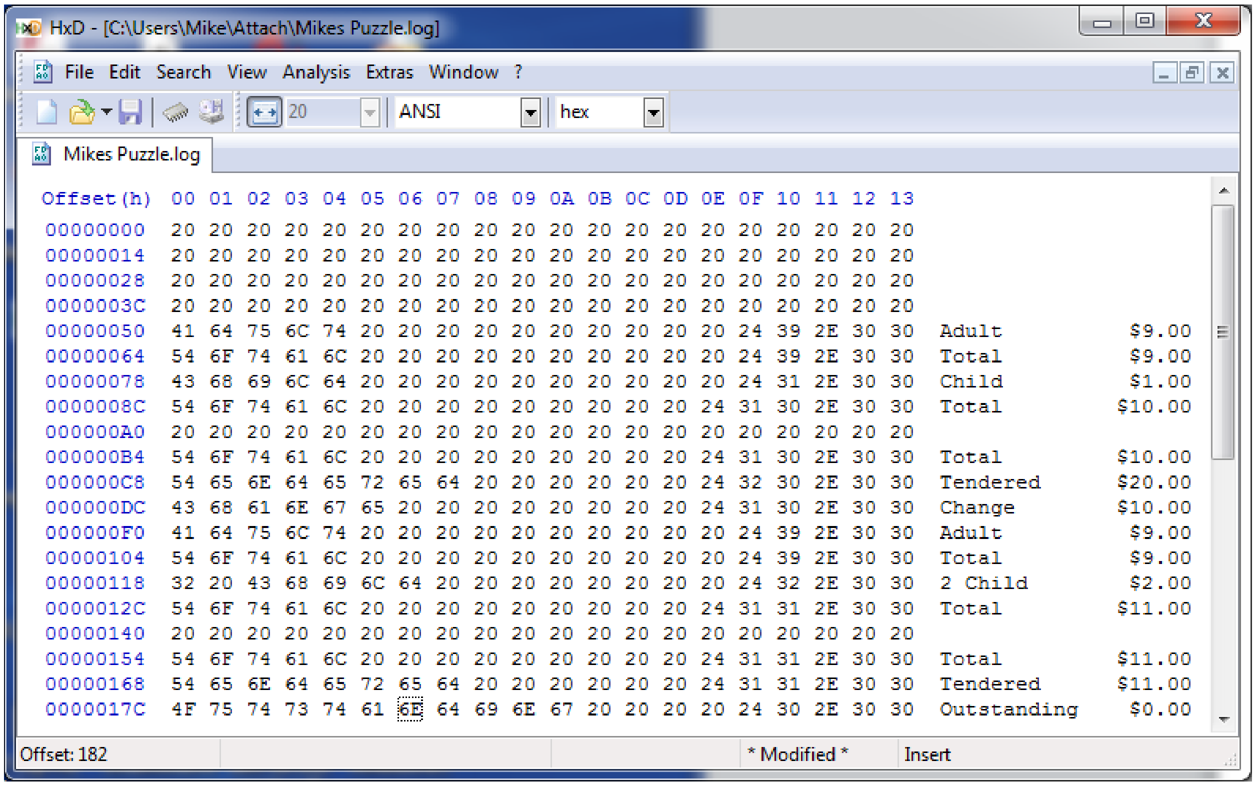

HxD - [C:\Users\Mike\Attach\Mikes Puzzle.log]
File Edit Search View Analysis Extras Window ?
16
ANSI
hex
Mikes Puzzle.log

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000010  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000020  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000030  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000040  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000050  41 64 75 6C 74 20 20 20 20 20 20 20 20 20 20 24  Adult          $
00000060  39 2E 30 30 54 6F 74 61 6C 20 20 20 20 20 20 20  9.00Total
00000070  20 20 20 24 39 2E 30 30 43 68 69 6C 64 20 20 20     $9.00Child
00000080  20 20 20 20 20 20 20 24 31 2E 30 30 54 6F 74 61         $1.00Tota
00000090  6C 20 20 20 20 20 20 20 20 20 24 31 30 2E 30 30  l         $10.00
000000A0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000B0  20 20 20 20 54 6F 74 61 6C 20 20 20 20 20 20 20      Total
000000C0  20 20 24 31 30 2E 30 30 54 65 6E 64 65 72 65 64    $10.00Tendered
000000D0  20 20 20 20 20 20 24 32 30 2E 30 30 43 68 61 6E        $20.00Chan
000000E0  67 65 20 20 20 20 20 20 20 20 24 31 30 2E 30 30  ge        $10.00
000000F0  41 64 75 6C 74 20 20 20 20 20 20 20 20 20 20 24  Adult          $
00000100  39 2E 30 30 54 6F 74 61 6C 20 20 20 20 20 20 20  9.00Total
00000110  20 20 20 24 39 2E 30 30 32 20 43 68 69 6C 64 20     $9.002 Child
```

Offset: 182 * Modified * Insert

We do observe numerous 20 codes however, which are space characters, so it's clear that this data source relies on literal spaces for print formatting. XBOB-4 will have to be configured for print width identical to that expected by the POS terminal. HxD has a nifty tool for discovering print width: a button just below the **View** menu that makes displayed data adapt to window width [⟵⟶]. Here's what happens when we click that button and play with window width:



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13

00000000  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000014  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000028  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0000003C  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000050  41 64 75 6C 74 20 20 20 20 20 20 20 20 20 20 24 39 2E 30 30  Adult          $9.00
00000064  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 20 24 39 2E 30 30  Total          $9.00
00000078  43 68 69 6C 64 20 20 20 20 20 20 20 20 20 20 24 31 2E 30 30  Child          $1.00
0000008C  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 24 31 30 2E 30 30  Total         $10.00
000000A0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000B4  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 24 31 30 2E 30 30  Total         $10.00
000000C8  54 65 6E 64 65 72 65 64 20 20 20 20 20 20 24 32 30 2E 30 30  Tendered      $20.00
000000DC  43 68 61 6E 67 65 20 20 20 20 20 20 20 20 24 31 30 2E 30 30  Change        $10.00
000000F0  41 64 75 6C 74 20 20 20 20 20 20 20 20 20 20 24 39 2E 30 30  Adult          $9.00
00000104  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 20 24 39 2E 30 30  Total          $9.00
00000118  32 20 43 68 69 6C 64 20 20 20 20 20 20 20 20 24 32 2E 30 30  2 Child        $2.00
0000012C  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 24 31 31 2E 30 30  Total         $11.00
00000140  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000154  54 6F 74 61 6C 20 20 20 20 20 20 20 20 20 24 31 31 2E 30 30  Total         $11.00
00000168  54 65 6E 64 65 72 65 64 20 20 20 20 20 20 24 31 31 2E 30 30  Tendered      $11.00
0000017C  4F 75 74 73 74 61 6E 64 69 6E 67 20 20 20 20 24 30 2E 30 30  Outstanding    $0.00
```

Hey voilà, everything falls into place as we stretch the window to 20 columns! So how can XBOB-4 be configured to behave like a 20-column printer? The solution is a powerful command that customers often overlook; the **q** command. See the XBOB-4 App Guide for details. Here's an example that specifies a printable area of 20 columns and five rows centered at screen bottom, which might be a good solution for this customer:

```
$1b[175;65;120;240.q
```

To understand this, know that default character size is 12x13 pixels (HxW) and default screen size is 480x240 pixels (NTSC, area constraint defeated). 20 characters are 20 * 12 = 240 pixels wide. Five rows are 5 * 13 = 65 pixels tall. The left edge of the printable area is located at (480 – 240) / 2 = 120 pixels from left screen edge. And the top of the printable area is located at 240 – 65 = 175 pixels from screen top.

Because the **q** command doesn't affect configuration memory like **v** commands do, it must be invoked every time XBOB-4 boots up. We make that happen automatically by storing it in a bootscript:

```
$1bX  $1b[175;65;120;240.q  $1b\  $1b[8v  $1b[1v
```

`$1bX` and `$1b\` sequester the **q** command string in a buffer, preventing immediate execution. `$1b[8v` captures that buffer's contents as a bootscript. `$1b[1v` then stores the entire configuration, including bootscript, into flash memory. Spaces have been added for clarity. It's best to remove them in the final application because, for instance, the ones inside the bootscript would print at boot time, possibly causing an undesired print position offset.

A couple of additional details: Because modern TV monitors don't underscan like CRT monitors, it's generally wise to defeat XBOB-4's display area constraint. The **v** command with **n** = 21 and **m** = 0 handles this chore. Our **q** command discussion above assumes that this has been done:

```
$1b[21;0v
```

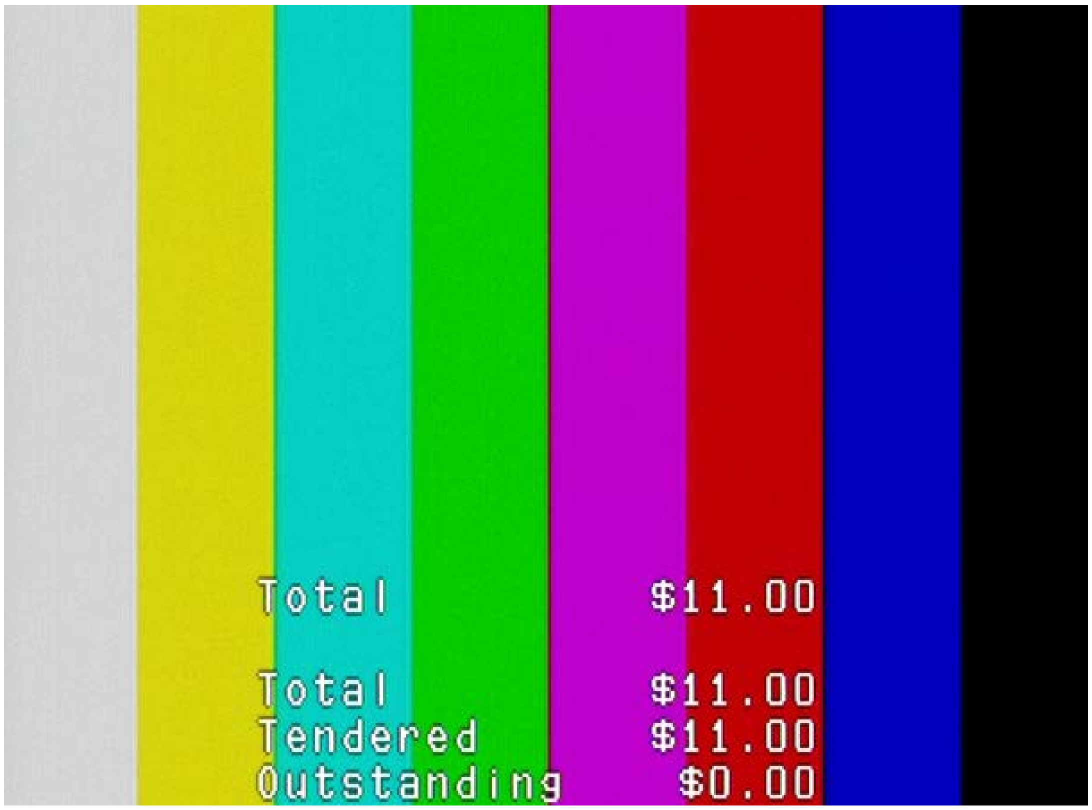
It's often helpful in POS application to clear the text overlay after a while, say 20 seconds. This feature is tucked away under Miscellaneous commands, invoked with the vertical bar character:

```
$1b[2;20|
```

It has to go inside a bootscript for the same reason as the **q** command. So now, by assembling all this in one place, we obtain the following object of wonderment:

```
$1b[21;0v $1bX $1b[2;20| $1b[175;65;120;240.q $1b\ $1b[8v $1b[1v
```

After transmitting the command salvo to XBOB-4 (without spaces) via Terminal's **Send** edit box, and cycling the power, it's time to test the results. To make this possible, we use HxD to save raw print data into a new file after stripping out the header and footer. We can now use the **Send File** button in Terminal to transmit the customer's data to XBOB-4 just as it was originally emitted by the POS terminal. Here's the result:



```
Total             $11.00

Total             $11.00
Tendered          $11.00
Outstanding        $0.00
```

Only the last five lines of data appear at bottom center. All the rest has scrolled off the top of the print window. Under normal circumstances, that data would have been visible for a time between transactions — certainly long enough to be stored in a DVR. So it looks like our work is done here.

But what if you wanted the text to occupy even less screen area? One solution is to use a smaller font. You could select the 8x13 font by inserting a **z** command into the bootscript, for instance. Be aware that smaller fonts can become less legible, especially when text is superimposed over complex and brightly colored areas of the underlying image. It’s possible to fix this issue by using a render mode that makes character cell background dark or black instead of transparent. See **m** command discussion in the XBOB-4 App Guide. Modes 66 and 70 are often useful.

In some cases where text obscures the underlying image too much, a better solution is to make the text semi-transparent. You can do that by backing the front-panel MIX trimmer away from full clockwise. The resulting contrast reduction also diminishes video noise around character edges. Don’t leave the MIX trimmer full counterclockwise and forget about it — you will never see text again!

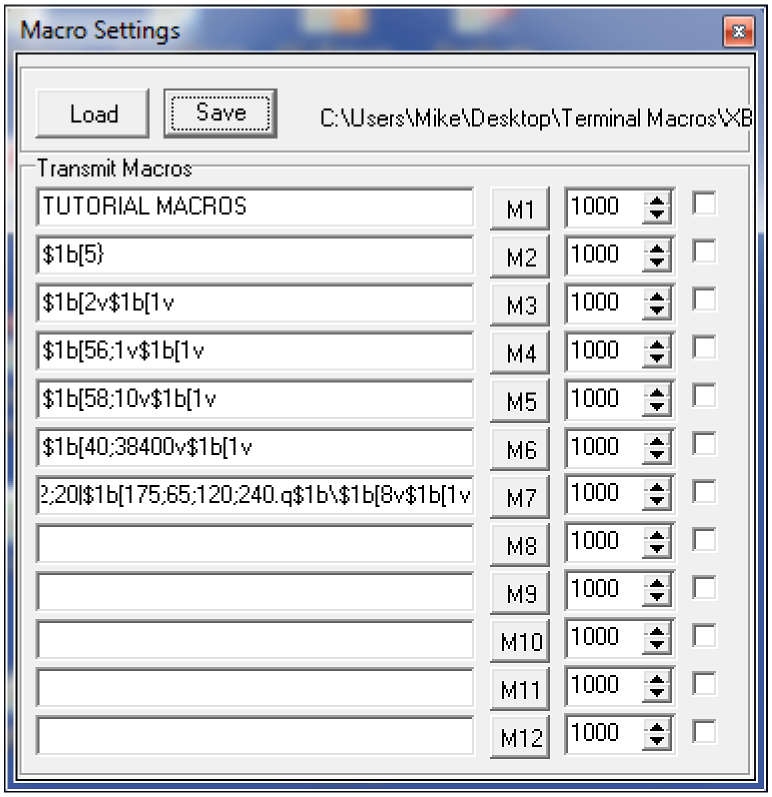
About Carriage Returns & Line Feeds

By default, XBOB-4 handles <CR> and <LF> codes literally. That is, <CR> moves print position to the beginning of the current line, and <LF> moves print position down by one line. If you encounter data sources that emit only one of these codes, expecting the printer to act as if both were present, then just configure XBOB-4 to behave like the target printer. See **v** command; **n** = 11. This setting often does the job:

```
$1b[11;3v
```

Multiple Installations

There are at least three ways to avoid retyping long strings of cryptic commands. If your command string isn't much bigger than the example above, then you could just paste it into a Terminal macro edit box. Macro strings can exceed box length, as shown in the M7 box below:



TUTORIAL MACROS	M1	1000	☐
$1b[5}	M2	1000	☐
$1b[2v$1b[1v	M3	1000	☐
$1b[56;1v$1b[1v	M4	1000	☐
$1b[58;10v$1b[1v	M5	1000	☐
$1b[40;38400v$1b[1v	M6	1000	☐
2;20I$1b[175;65;120;240.q$1b\$1b[8v$1b[1v	M7	1000	☐
	M8	1000	☐
	M9	1000	☐
	M10	1000	☐
	M11	1000	☐
	M12	1000	☐

The **Set Macro** button opens this window. Once Macro Settings have been saved, it's only necessary to smack the M7 button in order to configure another XBOB-4 in the same way. Here's what the other example macros do: M1 simply prints the text; M2 summons the configuration report (to the terminal); M3 restores default configuration; M4 enables SPI slave device 0 (extended font memory); M5 enables the distance encoder interface (only in XBOB-4E); M6 sets the communication rate to 38,400 bps. Note that M6 puts the kibosh on communication after reboot, until Terminal's baud rate is set correspondingly.

Another method for configuring multiple devices, which is better if you have a lot of commands and/or printable text in a bootscript, is to put that stuff into a plain ASCII file and then transmit it to XBOB-4 with Terminal's **Send File** capability. The only hitch is that many text editors don't handle <ESC> properly. Instead of "$1b" for <ESC>, you must include literal control code bytes in these files. Programmer's Notepad is one editor with a good solution. It displays inline <ESC> codes with a special symbol and allows the user to insert them at will, by holding down the **Alt** key while typing 027 on the number pad ($27_{10} = 1B_{16}$). It's also possible to insert control codes with a hex editor such as HxD.

Finally, be aware that Decade Engineering offers the free BOB-4 Conscriptor utility program, which provides GUIs for custom font management as well as configuration and bootscript editing. The **Setup Editor** in Conscriptor makes it possible to load XBOB-4 with custom fonts, bootscript and configuration data by clicking one button. Bootscript and configuration files saved by Conscriptor are readable with ordinary text editors, assuming that they do something sensible with embedded <ESC> codes. Windows Notepad displays a short left arrow.