TN-191

Use a Video Display Terminal for Enhanced Program Debugging and Operation

Mark Durgin - SCIDYNE® Corporation

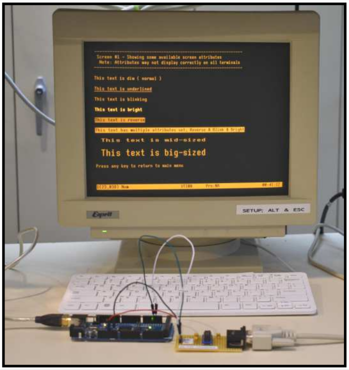
Introduction

This Tech Note describes how a Video Display Terminal (a.k.a. "VDT" or just "Terminal") can be a useful tool when working with low-cost development boards and embedded systems. Control Codes and Escape Sequences are explained and the concepts demonstrated using an Arduino MEGA microcontroller board and VT100 compatible terminal.

Why use a Terminal?

Many of the inexpensive entry-level development systems popular today offer only rudimentary abilities to debug and monitor their operation. For example, the Arduino IDE includes a Serial Monitor for observing running programs. It's simple and convenient but is less practical when trying to display lots of information in an organized and visually pleasing manner. Another drawback is that the development computer must remain connected and running the IDE. This can be a disadvantage during long-term tests or if the computer is needed for other purposes. Using a Video Display Terminal addresses these and several other issues.

For the purpose of this Tech Note an Esprit Model #5000/6102 (circa 1997) Video Display Terminal is used. Like most terminals, it can be configured to emulate the characteristics of several other popular terminal brands and models. Emulation for the Digital Equipment Corporation VT100 is selected as this is a de facto industry standard and suitable for our basic needs. In lieu of an actual Terminal, a spare computer or Raspberry Pi can be used by running any one of a number of popular terminal emulation programs such as PuTTY, Hyper Terminal, Tera Term, or ProComm.



```
Screen #1 - Showing some available screen attributes
 Note: Attributes may not display correctly on all terminals

This text is dim ( normal )
This text is underlined
This text is blinking
This text is bright
This text is reverse
This text has multiple attributes set; Reverse & Blink & Bright
  This text is mid-sized
  This text is big-sized
Press any key to return to main menu
```

Basic Terminal Operation

A Video Display Terminal displays the ASCII characters it receives over a serial communication connection. For example, send it a byte with the value 0x39 (hexadecimal) and the number 9 appears on the screen. Likewise, a capital letter Q will appear when a 0x51 is received. The characters are displayed as persistent dot-matrix graphics managed entirely by the VDT hardware.

Unless configured otherwise, each character received is displayed at the current cursor location and the cursor automatically advances one column to the right of the current row. Reaching the end of a row causes the cursor to reposition to the leftmost column on the next row down. The top-most row disappears as the screen begins scrolling up once the bottom row is filled with characters.

In addition, a terminal can serve as an alpha-numeric keyboard input. Keystrokes entered at the terminal are transmitted to the attached device as ASCII encoded bytes.

Control Codes and Escape Sequences

Besides displaying text characters a terminal will respond to several non-displayable bytes called Control Codes. These special bytes, used alone or in conjunction with other characters and byte values, perform useful operations like instructing the terminal to emit a beep sound, or moving the cursor down one row. For reference the standard ASCII table including Control Codes is shown in Appendix-A.

A string of bytes that begins with an ASCII Escape code (0x1B) is commonly referred to as an "Escape Sequence". When the terminal receives an Escape code the subsequent byte or bytes are not displayed but instead interpreted as parameters belonging to the Escape Sequence. The bytes are grouped as one contiguous uninterrupted block and do not have a Carriage Return or Line-feed termination at the end.

This Escape Sequence instructs	**ASCII Character:**	Esc	[	2	J
the terminal to clear its entire display	**Byte Hex Value:**	**1B**	**5B**	**32**	**4A**

After executing an Escape Sequence the terminal automatically resumes displaying received text characters until another Escape Sequences is detected.

The table below summarizes some of the more useful Control Codes and Escape Sequences. These codes are standard for VT100 terminals but should also work on most other terminals as well. Many more Escape Sequences exist beside the basic ones shown. The reader is encouraged to seek a good VT100 terminal reference to learn more. Appendix-B provides a basic summary of common VT100 Escape Sequences.

Control Codes	Description
BEL	**Bell:** Causes the terminal to emit a momentary beep sound.
LF	**Line Feed:** Moves the cursor from its current position vertically down one row.
CR	**Carriage Return:** Moves the cursor to the first column (leftmost position) of its current row. Note: Some terminals are configured so that a CR will actually behave like a CR LF.
Escape Sequence	**Description**
Esc[2J	**Erase Screen:** Clears the entire terminal screen. Does not move cursor to home upper-left position.
Esc[{row};{column}H	**Locate Cursor:** Sets the cursor position where subsequent text will begin. If no row/column parameters are provided (i.e. Esc[H), the cursor will move to the home upper-left position.
Set a single attribute: Esc[{attr}m **Set multiple attributes:** Esc[{attr};...;{attr}m	**Set Attribute Mode:** Sets a single or multiple display attribute. Use a semicolon to separate multiple attributes set within the same sequence. If only one attribute is being set the semicolon should be omitted.

Character Attribute	
0	Reset all attributes
1	Bright
2	Dim
4	Underscore
5	Blink
7	Reverse (Inverse)
8	Hidden

Screen Color Attribute*

Color	Foreground	Background
Black	30	40
Red	31	41
Green	32	42
Yellow	33	43
Blue	34	44
Magenta	35	45
Cyan	36	46
White	37	47

*Screen color attributes are not supported by all terminals. Values shown are in decimal notation

Notes:

1. Esc represents the ASCII "escape" character, 27 or 0x1B.
2. Bracketed tags represent modifiable parameters; e.g. {row} would be replaced by a row number, brackets excluded. The values are expressed as single-byte numbers, not two-byte ASCII encoded.

Terminal Serial Communications

The terminal communicates over an asynchronous serial link, typically RS-232. Most any device that supports asynchronous communications can be connected including Personal Computers, Microprocessors, and Programmable-Logic (e.g., CPLD or FPGA). The core requirement is that the device is able to transmit (and receive if communicating bi-directionally) bytes comprising Control Codes, Escape Sequences, and printable ASCII information. For demonstration purposes an Arduino MEGA 2560 is used due to its popularity, low cost, and ease of use.

The MEGA 2560 has four hardware (USART based) asynchronous serial ports as summarized in the following table:

Arduino Mega2560 Serial Ports				
	Serial (Hard-wired to USB)	Serial1	Serial2	Serial3
Tx Output	Pin#1	Pin#18	Pin#16	Pin#14
Rx Input	Pin#0	Pin#19	Pin#17	Pin#15

The first serial port (Serial) is hard-wired to the onboard USB circuitry and is primarily used for communicating to the host computer running the Arduino IDE. For our setup the second serial port (Serial1) is available and connected as shown in figure 1.

A Virtual Communications Port is created on the host computer whenever a MEGA 2560 is connected via USB. This port is associated with Serial on the MEGA 2560 and used by the Arduino IDE to upload and monitor sketches. The same port can also be used by a Terminal Emulator such as PuTTY. However, difficulty can occur when attempting to upload to the MEGA 2560 with both programs running. Be sure to shut down PuTTY before uploading new sketches.

Since Serial1 operates at +5V and RS-232 requires bi-polar signals an intermediate circuit was needed to interface the two voltage levels. The simple circuit consists of a MAX232A chip and related components. Finally, a special cable (a.k.a. Null Modem) was constructed to cross-over the Transmit and Receive signals and also convert between the 9-Pin and 25-Pin Female D-SUB connectors.

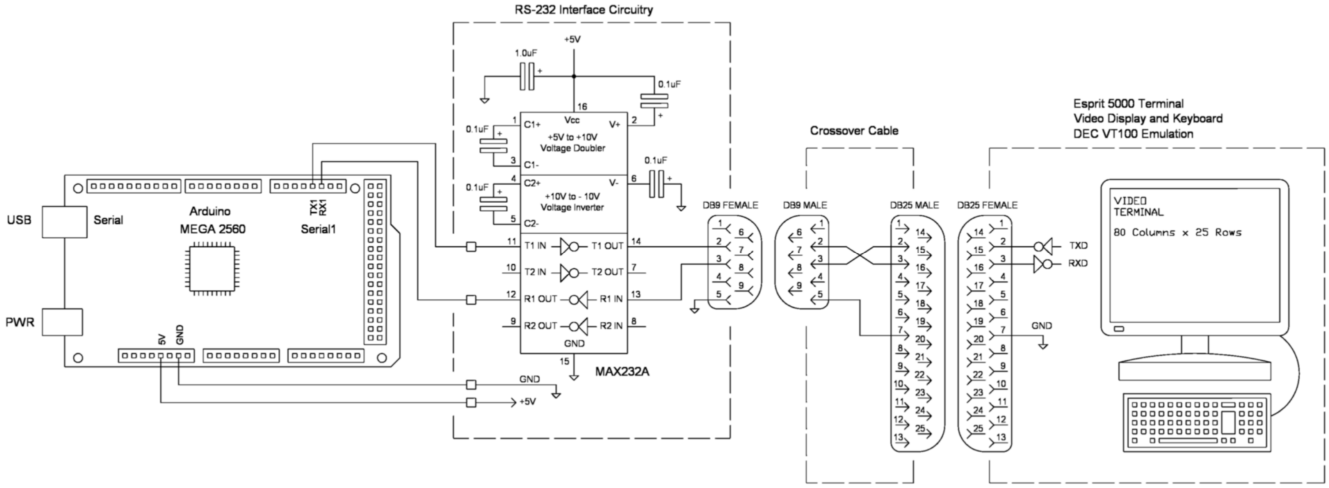


Figure 1 - Wiring Diagram

Arduino Serial Communications

The Arduino environment provides good support for serial communications through several standard library functions. Only the functions used in this Tech Note are discussed but the reader can explore the official Arduino documentation to learn more.

Function	Description
Serial1.begin(baud, settings)	This must be called once to initializes the serial port for asynchronous serial communications. All parameters on both devices need to match exactly in order for communications to work properly. The Baud Rate (bits-per-second) parameter is required; standard rates are: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. The optional second parameter sets data, parity, and stop bits. If not specified the default of 8 data bits, no parity, one stop bit is used. Example: `Serial1.begin(9600, SERIAL_8N1);`
Serial1.write()	Writes binary data to the serial port as a single byte or a series of bytes. Example: `Serial1.write(0x07);`
Serial1.print()	Sends ASCII encoded characters and strings to the serial port. Example: `Serial1.print("Hello World");`
Serial1.println()	Same as Serial1.print but appends control codes Carriage Return (0x0D) and Line Feed (0x0A) to the outgoing string. These two control codes cause the cursor to be placed at the leftmost column on the next row down.

Serial1.available()	Checks to see if any characters have been received on the serial port. Returns binary TRUE (logic 1) if characters are present and ready to be read otherwise returns binary FALSE (logic 0). Does not remove received characters from buffer.
	Example: `if (Serial1.available()) {`
	`... run this code ...`
	`}`

Serial1.read()	Reads available characters from the serial port in to a variable or buffer. Example: `buf[] = Serial1.read();`

Notes:

1) The "1" appearing in the serial communication function names denotes Serial port 1. If using the second serial port use Serial2 instead, and likewise for Serial3. To communicate to the Arduino IDE serial monitor or a terminal emulation program sharing the same host virtual communications port simply use Serial without any number suffix.

2) For enhanced readability and easier future program maintenance the accompanying demonstration software uses a #define to denote the serial port connected to the terminal. During compilation the text Serial1 will be substituted for each occurrence of the text TERMPORT. This allows all references to the MEGA serial port connected to the terminal to be set by one assignment prior to compiling.

3) The serial library must be included in the Arduino sketch to gain access to the functions.

Terminal Demonstration Software

The accompanying demonstration software **tn191.ino** is well commented and self-explanatory. It provides a good overview of how to communicate with Video Display Terminal. Even though it has been written for the Arduino MEGA 2560 the concepts are generic enough for other devices and computer languages. Appendix-C provides a listing of the demonstration software. Always check the SCIDYNE website for the most current version.

Helpful Tips

When applying the software for your own use the following tips and code snippets will aid in creating code that is fast, efficient, and easy to maintain:

Assign the serial port associated with the terminal using a #define

As described earlier the MEGA 2560 has four independent asynchronous serial ports; Serial - Serial3. The serial port designated to communicate with the terminal is referenced repeatedly throughout the software. If there is ever a need to switch ports sizeable effort would be required to correctly locate and change all the references. Although it's possible to do this with the Find-And-Replace feature of the Arduino IDE editor the preferred method is to use a single #define at the top of the program (or an included .h file) and have the compiler do all the work by means of text substitution. The demonstration software uses the mnemonic TERMPORT to denote which serial port will be associated with the terminal. This technique also greatly improves the readability of the software for other programmers by indicating exactly what the port is used for.

```
//-- Designate which serial port will communicate with the terminal
#define TERMPORT Serial1  // Valid choices are: Serial, Serial1, Serial2, or Serial3
```

Save SRAM with the F() macro

The default action of the Arduino compiler is to place literal strings in Flash memory, then copy them to SRAM at startup as static SRAM variables. Programs with many string can quickly consume available SRAM. Using the F() macro instructs the compiler to keep strings in Flash PROGMEM memory space. To use it simply wrap the literal string in the F() macro. The macro is part of the Arduino IDE.

```
//-- Wrap strings with F() macro to save SRAM
TERMPORT.println(F("This text stays in Flash memory"));   // Print text from Flash
```

Use a SRAM buffer and sprintf() to create strings

When creating text strings it is handy to use the sprintf() function. This built-in C language function has the same functionality and follows similar formatting rules as the printf() function. The difference is that the resulting string is placed in a SRAM buffer rather than being immediately outputted. This allows great flexibility on how a string is constructed. For instance, additional operations can be performed on the buffer before outputting such as including formatted numbers, string concatenation, and character replacement.

```
//-- General purpose variables
char buf[80];       // Reserve a SRAM buffer where strings will be created
int valn = 12345;  // A preloaded variable

//-- Use Sprintf
sprintf(buf,"Dec: %d = Hex: %X", valn)); // Create a text string and place in buffer
TERMPORT.println(buf);  // Send it to the terminal
```

Speed-up things by splitting the screen between Static and Live Data

A screen will generally consist of both static (non-changing) text such as labels and prompts along with live data updated as the program runs. Since the static items never change it makes sense to send them only once. The live data can be placed on the screen where needed using the locate cursor escape sequence. The result is the program will not be wasting time re-sending non-changing data which in turn increases overall program execution speed.

```
// Place static (non-changing) information on screen first
cls();              // Clear the terminal screen
locate (0, 0);      // Start in upper-left corner
TERMPORT.println(F("---------------------------"));
TERMPORT.println(F(" Show static and live data"));
TERMPORT.println(F("---------------------------"));

locate(5,0);  // Locate cursor at row 5 column 0
TERMPORT.print(F("This program has been running for       Milli-Seconds"));

// Now add live data.  Repeat this loop until a terminal key is pressed
do {
  locate(5, 35);                       // Locate cursor with-in pre-placed text
  sprintf(buf,"%5u", millis());        // Create a string of system milli-seconds
  TERMPORT.print(buf);                 //  and print it at current cursor location
} while ( !(TERMPORT.available()));    // Repeat live data loop until a terminal key is pressed

// Clean-up before exit
TERMPORT.read();  // Read and discard the keypress entered on the terminal by user
```

References

http://ascii-table.com/ansi-escape-sequences-vt-100.php

https://en.wikipedia.org/wiki/Escape_sequence

http://www.termsys.demon.co.uk/vtansi.htm

Appendix - A

US ASCII, ANSI X3.4-1986 (ISO 646 International Reference Version)

Dec	Hex	Keyboard	Name	Description	Dec	Hex	Char	Description	Dec	Hex	Char	Description	Dec	Hex	Char	Description
0	00	CTRL-@	NUL	Null	32	20		Space	64	40	@	Commercial At Sign	96	60	`	Grave Accent
1	01	CTRL-A	SOH	Start Of Heading	33	21	!	Exclamation Mark	65	41	A	Capital Letter A	97	61	a	Small letter a
2	02	CTRL-B	SOT	Start Of Text	34	22	"	Quotation Mark	66	42	B	Capital Letter B	98	62	b	Small letter b
3	03	CTRL-C	ETX	End Of Text	35	23	#	Number Sign, Hash Tag	67	43	C	Capital Letter C	99	63	c	Small letter c
4	04	CTRL-D	EOT	End Of Transmission	36	24	$	Dollar Sign	68	44	D	Capital Letter D	100	64	d	Small letter d
5	05	CTRL-E	ENQ	Enquiry	37	25	%	Percent Sign	69	45	E	Capital Letter E	101	65	e	Small letter e
6	06	CTRL-F	ACK	Acknowledge	38	26	&	Ampersand	70	46	F	Capital Letter F	102	66	f	Small letter f
7	07	CTRL-G	BEL	Bell (Beep)	39	27	'	Apostrophe	71	47	G	Capital Letter G	103	67	g	Small letter g
8	08	CTRL-H	BS	Backspace	40	28	(	Right Parenthesis	72	48	H	Capital Letter H	104	68	h	Small letter h
9	09	CTRL-I	HT	Horizontal Tab	41	29	)	Left Parenthesis	73	49	I	Capital Letter I	105	69	i	Small letter i
10	0A	CTRL-J	LF	Line Feed	42	2A	*	Asterisk	74	4A	J	Capital Letter J	106	6A	j	Small letter j
11	0B	CTRL-K	VT	Vertical Tab	43	2B	+	Plus Sign	75	4B	K	Capital Letter K	107	6B	k	Small letter k
12	0C	CTRL-L	FF	Form Feed	44	2C	,	Comma	76	4C	L	Capital Letter L	108	6C	l	Small letter l
13	0D	CTRL-M	CR	Carriage Return	45	2D	-	Minus Sign, Hyphen	77	4D	M	Capital Letter M	109	6D	m	Small letter m
14	0E	CTRL-N	SO	Shift Out	46	2E	.	Period	78	4E	N	Capital Letter N	110	6E	n	Small letter n
15	0F	CTRL-O	SI	Shift In	47	2F	/	Forward. Slash	79	4F	O	Capital Letter O	111	6F	o	Small letter o
16	10	CTRL-P	DLE	Data Link Escape	48	30	0	Digit 0	80	50	P	Capital Letter P	112	70	p	Small letter p
17	11	CTRL-Q	DC1	Device Control 1 (XON)	49	31	1	Digit 1	81	51	Q	Capital Letter Q	113	71	q	Small letter q
18	12	CTRL-R	DC2	Device Control 2	50	32	2	Digit 2	82	52	R	Capital Letter R	114	72	r	Small letter r
19	13	CTRL-S	DC3	Device Control 3 (XOFF)	51	33	3	Digit 3	83	53	S	Capital Letter S	115	73	s	Small letter s
20	14	CTRL-T	DC4	Device Control 4	52	34	4	Digit 4	84	54	T	Capital Letter T	116	74	t	Small letter t
21	15	CTRL-U	NAK	Negative Acknowledge	53	35	5	Digit 5	85	55	U	Capital Letter U	117	75	u	Small letter u
22	16	CTRL-V	SYN	Synchronous Idle	54	36	6	Digit 6	86	56	V	Capital Letter V	118	76	v	Small letter v
23	17	CTRL-W	ETB	End Of Trans. Block	55	37	7	Digit 7	87	57	W	Capital Letter W	119	77	w	Small letter w
24	18	CTRL-X	CAN	Cancel	56	38	8	Digit 8	88	58	X	Capital Letter X	120	78	x	Small letter x
25	19	CTRL-Y	EM	End Of Medium	57	39	9	Digit 9	89	59	Y	Capital Letter Y	121	79	y	Small letter y
26	1A	CTRL-Z	SUB	Substitute	58	3A	:	Colon	90	5A	Z	Capital Letter Z	122	7A	z	Small letter z
27	1B	CTRL-[	ESC	Escape	59	3B	;	Semicolon	91	5B	[	Left Square Bracket	123	7B	{	Left curly bracket
28	1C	CTRL-\	FS	File Separator	60	3C	<	Less-Than Sign	92	5C	\	Back Slash	124	7C	|	Vertical Bar
29	1D	CTRL-]	GS	Group Separator	61	3D	=	Equal Sign	93	5D	]	Right Square Bracket	125	7D	}	Right curly Bracket
30	1E	CRTL-^	RS	Record Separator	62	3E	>	Greater-Than Sign	94	5E	^	Circumflex Accent	126	7E	~	Tilde
31	1F	CTRL-_	US	Unit Separator	63	3F	?	Question Mark	95	5F	_	Underscore	127	7F		Delete (CTRL-?)

Notes:
0x00 - 0x1F are control codes
0x20 - 0x7E are printable characters

The ASCII standard defines byte values of 0x00 through 0x1F and 0x7F as non-displayable control codes. Bytes with a value between 0x20 and 0x7E are used to denote displayable ASCII text covering the English alphabet, decimal digits, and common punctuation symbols. The remaining values, 0x80 through 0xFF (not shown), are unassigned but often used (although not universally) for special purposes such box drawing graphics, special typographic symbols, or to display alternate language characters.

Appendix - B

Summary of VT-100 Escape Sequences

Sequence	Description
Esc[20h	Set new line mode
Esc[?1h	Set cursor key to application
Esc[?3h	Set number of columns to 132
Esc[?4h	Set smooth scrolling
Esc[?5h	Set reverse video on screen
Esc[?6h	Set origin to relative
Esc[?7h	Set auto-wrap mode
Esc[?8h	Set auto-repeat mode
Esc[?9h	Set interlacing mode
Esc[20l	Set line feed mode
Esc[?1l	Set cursor key to cursor
Esc[?2l	Set VT52 (versus ANSI)
Esc[?3l	Set number of columns to 80
Esc[?4l	Set jump scrolling
Esc[?5l	Set normal video on screen
Esc[?6l	Set origin to absolute
Esc[?7l	Reset auto-wrap mode
Esc[?8l	Reset auto-repeat mode
Esc[?9l	Reset interlacing mode
Esc=	Set alternate keypad mode
Esc>	Set numeric keypad mode
Esc(A	Set United Kingdom G0 character set
Esc)A	Set United Kingdom G1 character set
Esc(B	Set United States G0 character set
Esc)B	Set United States G1 character set
Esc(0	Set G0 special chars. & line set
Esc)0	Set G1 special chars. & line set
Esc(1	Set G0 alternate character ROM
Esc)1	Set G1 alternate character ROM
Esc(2	Set G0 alt char ROM and spec. graphics
Esc)2	Set G1 alt char ROM and spec. graphics

Sequence	Description
EscN	Set single shift 2
EscO	Set single shift 3
Esc[m	Turn off character attributes
Esc[0m	Turn off character attributes
Esc[1m	Turn bold mode on
Esc[2m	Turn low intensity mode on
Esc[4m	Turn underline mode on
Esc[5m	Turn blinking mode on
Esc[7m	Turn reverse video on
Esc[8m	Turn invisible text mode on
Esc[Line;Liner	Set top and bottom lines of a window
Esc[ValueA	Move cursor up n lines
Esc[ValueB	Move cursor down n lines
Esc[ValueC	Move cursor right n lines
Esc[ValueD	Move cursor left n lines
Esc[H	Move cursor to upper left corner
Esc[;H	Move cursor to upper left corner
Esc[Line;ColumnH	Move cursor to screen location v,h
Esc[f	Move cursor to upper left corner
Esc[;f	Move cursor to upper left corner
Esc[Line;Columnf	Move cursor to screen location v,h
EscD	Move/scroll window up one line
EscM	Move/scroll window down one line
EscE	Move to next line
Esc7	Save cursor position and attributes
Esc8	Restore cursor position and attributes
EscH	Set a tab at the current column
Esc[g	Clear a tab at the current column
Esc[0g	Clear a tab at the current column
Esc[3g	Clear all tabs

Sequence	Description
Esc#3	Double-height letters, top half
Esc#4	Double-height letters, bottom half
Esc#5	Single width, single height letters
Esc#6	Double width, single height letters
Esc[K	Clear line from cursor right
Esc[0K	Clear line from cursor right
Esc[1K	Clear line from cursor left
Esc[2K	Clear entire line
Esc[J	Clear screen from cursor down
Esc[0J	Clear screen from cursor down
Esc[1J	Clear screen from cursor up
Esc[2J	Clear entire screen
Esc5n	Device status report
Esc0n	Response: terminal is OK
Esc3n	Response: terminal is not OK
Esc6n	Get cursor position
EscLine;ColumnR	Response: cursor is at v,h
Esc[c	Identify what terminal type
Esc[0c	Identify what terminal type (another)
Esc[?1;Value0c	Response: terminal type code n
Escc	Reset terminal to initial state
Esc#8	Screen alignment display
Esc[2;1y	Confidence power up test
Esc[2;2y	Confidence loopback test
Esc[2;9y	Repeat power up test
Esc[2;10y	Repeat loopback test
Esc[0q	Turn off all four leds
Esc[1q	Turn on LED #1
Esc[2q	Turn on LED #2
Esc[3q	Turn on LED #3
Esc[4q	Turn on LED #4

Notes:
Esc = ASCII "Escape" character decimal 27, Hexadecimal 0x1b.

An ANSI Escape Sequence is a series of ASCII characters, the first of which is the ASCII "Escape" character 27 (0x1B). The character or characters following the escape character specify an alphanumeric code that controls a display or keyboard function.

Appendix - C

Demonstration Software Listing
Visit the SCIDYNE website for the most current version.

```
//========================================================================
// SCIDYNE Corporation   649 School Street   Pembroke, MA 02359-3649 USA
// Tel: (781) 293-3059   Fax: (781) 293-4034   URL: www.scidyne.com
//========================================================================
// NOTICE -- This demonstration software is provided "As Is".  It can be
// freely used and modified as described under the terms of the
// Creative Commons Attribution License agreement.
//    For details visit: https://creativecommons.org/licenses/by/4.0/
//========================================================================
// Project  :  Tech Note TN-191 Demonstration Software
// File     :  TN191.ino
// Revision :  1.00
// Date     :  10-22-18
// Author:  :  Mark Durgin
// Target   :  Arduino Mega 2560
// Compiler :  Arduino IDE  V1.8.5
//========================================================================
// Description and usage:
//   This software accompanies Tech Note TN-191 "Driving Video Display Terminals".
//
//   In Arduino IDE under Tools -> Board select Arduino MEGA2560
//    Select the port associated with the MEGA 2560 connected to the computer via USB.
//
//   A single #define TERMPORT configures which Arduino MEGA 2560 serial port will be used:
//
//     Serial - Uses the USB port communicating to the host computer running the Arduino IDE.
//       A terminal emulation program such as PuTTY can also share this port with the Arduino IDE but
//       interference can be expected especially when uploading a new sketch and PuTTY is actively running.
//       Be sure to shut down PuTTY before trying to upload to Arduino.
//
//     Serial1(2,3) - A RS-232 Shield or interface circuit is used on Serial #1 to communicate and display
//       results on an VT100 ASCII Terminal. This avoids the interference issues described above.
//       This also applies to Serial2 and Serial3.
//
//========================================================================
// History:
//  10-17-18  R0.00   - Project started
//========================================================================

// -- Define program constants
#define TERMPORT Serial    // Set which Arduino serial port will be used; Serial, Serial1, Serial2, or Serial3

// -- Define Special ASCII and VT100 Terminal codes
const  char BEL = 0x07;                         // ASCII code for beep
const  char ESC = 0x1b;                         // ASCII code for escape
const  char ATTR_RESET = '0';                   // Code to reset all terminal attributes
const  char ATTR_BRIGHT = '1';                  // Code to have subsequent text print bright "bold"
const  char ATTR_DIM = '2';                     // Code to have subsequent text print dim
const  char ATTR_UNDERSCORE = '4';              // Code to have subsequent text print with underline
const  char ATTR_BLINK = '5';                   // Code to have subsequent text print as blinking
const  char ATTR_REVERSE = '7';                 // Code to have subsequent text print as inverse
const  char ATTR_HIDDEN = '8';                  // Code to have subsequent text print as hidden

// -- Declare prototypes before use
void beep (void);                               // Make terminal emit beep sound
void locate (byte row, byte col);               // locate cursor on terminal screen
void cls (void);                                // Clear the entire terminal screen
void cursormode (bool disp);                    // Control display of cursor; 0 = hide, 1 = display
void terminal_attribute (char attribute);       // Set a terminal attribute
void show_screen1 (void);                       // Show Screen 1
void show_screen2 (void);                       // Show Screen 2
void midtext(byte row, byte col, const char * strpntr);   // Display mid-sized (1-row) text starting at row, column
void bigtext(byte row, byte col, const char * strpntr);   // Display big-sized (2-row) text starting at row, column

//******************************************************************
// Program setup.  Runs once and initializes the program
//******************************************************************
void setup()
{
  // Setup for serial communication to the terminal
  //   IMPORTANT! - MUST set baud rate and other parameters to match terminal parameters
  TERMPORT.begin(9600, SERIAL_8N1);     // Start serial communications
}


//******************************************************************
// This is the main program.
//******************************************************************
void loop()
{
    cls();              //Clear the terminal screen
    locate (0, 0);      //Loacte cursor in upper left corner
    TERMPORT.println(F("------------------------------------------------------"));
    TERMPORT.println(F(" SCIDYNE Corporation   Demonstration Software  TN-191  10/22/18 "));
    TERMPORT.println(F("------------------------------------------------------"));
    TERMPORT.println();

    terminal_attribute(ATTR_UNDERSCORE);  // Subsequent text will be underlined
    TERMPORT.println(F("Main Menu:"));
    terminal_attribute(ATTR_RESET);  // Turn-off underline
    TERMPORT.println(F(" 1: Show some available screen attributes"));
    TERMPORT.println(F(" 2: Demonstrate locating the cursor with static and live data"));
    TERMPORT.println(F(" 3: Make the Terminal Beep"));
    TERMPORT.println();

    // Prompt user and wait for a keypress
    TERMPORT.print(F(" Please press 1, 2, or 3 "));
    while ( !(TERMPORT.available()) );   // Wait for a key press

    switch(TERMPORT.read()) {  // Read the key pressed and determine what to do next

      case '1'  : // Key 1 pressed
        show_screen1();
        break;

      case '2'  : // Key 2 pressed
        show_screen2();
        break;

      case '3'  : // Key 3 pressed
        beep();
        break;

      default   :  // An un-supported key was pressed
        break;
    }
}


//******************************************************************************
// Subroutines start here
//******************************************************************************

//************************************************
// Show screen #1
// Display some of the available attributes. Not all
// terminals can display all the attributes
//************************************************
void show_screen1 ( void )
{
    cls();                  //Clear the terminal screen
    locate (0, 0);      //Start in upper left corner
    TERMPORT.println(F("----------------------------------------------------"));
    TERMPORT.println(F(" Screen #1 - Showing some available screen attributes"));
    TERMPORT.println(F("  Note: Attributes may not display correctly on all terminals"));
    TERMPORT.println(F("----------------------------------------------------"));
    TERMPORT.println();

    // Show dim (normal) text
    terminal_attribute(ATTR_DIM);                            // Set the DIM attribute
    TERMPORT.println(F("This text is dim ( normal )"));    // Print the text string
    terminal_attribute(ATTR_RESET);                          // Reset all attributes
    TERMPORT.println();                                      // Move down one row

    // Show underlined text
    terminal_attribute(ATTR_UNDERSCORE);
    TERMPORT.println(F("This text is underlined"));
    terminal_attribute(ATTR_RESET);
    TERMPORT.println();

    // Show blinking text
    terminal_attribute(ATTR_BLINK);
    TERMPORT.println(F("This text is blinking"));
    terminal_attribute(ATTR_RESET);
    TERMPORT.println();

    // Show bright text
    terminal_attribute(ATTR_BRIGHT);
    TERMPORT.println(F("This text is bright"));
    terminal_attribute(ATTR_RESET);
    TERMPORT.println();

    // Show reverse text
    terminal_attribute(ATTR_REVERSE);
    TERMPORT.println(F("This text is reverse"));
    terminal_attribute(ATTR_RESET);
    TERMPORT.println();

    // Show text with multiple attributes
    terminal_attribute(ATTR_REVERSE);
    terminal_attribute(ATTR_BLINK);
    terminal_attribute(ATTR_BRIGHT);
    TERMPORT.println(F("This text has multiple attributes set; Reverse & Blink & Bright"));
    terminal_attribute(ATTR_RESET);

    // Show mid-sized text (double-width single-row)
    midtext(18, 2, "This text is mid-sized");

    // Show big-sized text (double-width double-row)
    bigtext(20, 2, "This text is big-sized");

    // Prompt user and Wait for a keypress
    terminal_attribute(ATTR_RESET);
    TERMPORT.println();
    TERMPORT.println();

    TERMPORT.print(F("Press any key to return to main menu "));

    while ( !(TERMPORT.available()) );       // Wait for a keypress
    TERMPORT.read();                         // Read and discard the keypress entered by user
}



//************************************************************************
// Show screen #2
// This screen shows how to display static (non-changing) information and
// then write live data without disturbing the static information or
// causing the terminal screen to scroll.
//************************************************************************
void show_screen2 ( void )
{
```

```
// Declare local variables
int  i;              // A general purpose variable
int  loopcntr = 0;     // A loop counter for this subroutine
char buf[80];         // A RAM buffer for creating strings, sized for maximum screen width

// Place static (non-changing) information on screen
cls();             // Clear the terminal screen
locate (0, 0);      // Start in upper-left corner
TERMPORT.println(F("----------------------------------"));
TERMPORT.println(F(" Screen #2 - Show static and live data"));
TERMPORT.println(F("----------------------------------"));

locate(20,0);       // Locate cursor
TERMPORT.print(F("Press any key to return to main menu "));

locate(5,0);        // Locate cursor
TERMPORT.print(F("Counter Value [dec]:"));

locate(6,15);       // Locate cursor
TERMPORT.print(F("[hex]:"));

locate(7,15);       // Locate cursor
TERMPORT.print(F("[oct]:"));

locate(8,15);       // Locate cursor
TERMPORT.print(F("[bin]:"));

locate(9,33);       // Locate cursor
TERMPORT.print(F("|"));
locate(10,33);       // Locate cursor
TERMPORT.print(F("|"));
locate(11,33);       // Locate cursor
TERMPORT.print(F("+--- Bit#6 is "));

// Analog input static text
for(i=0; i<5; i++)
{
   locate(13+i,0);
   sprintf(buf,"Analog In-%d:", i);
   TERMPORT.print(buf);
}

cursormode(0);   // Disable the cursor

// Add live data.  Repeat this loop until a key is pressed
do {
  locate(5, 23);                          // Locate cursor beyond pre-placed text
  sprintf(buf,"%5d",loopcntr);     // Create a string to print (5 digit signed decimal)
  TERMPORT.print(buf);            //  and print it at current cursor location

  locate(6, 24);                          // Locate cursor beyond pre-placed text
  sprintf(buf,"%04X",loopcntr);   // Create a string to print (4 digit hexadecimal)
  TERMPORT.print(buf);            //  and print it at current cursor location

  locate(7, 22);                           // Locate cursor beyond pre-placed text
  sprintf(buf,"%06o",loopcntr);     // Create a string to print (6 digit octal)
  TERMPORT.print(buf);              //  and print it at current cursor location

  locate(8, 22);               // Locate cursor beyond pre-placed text
  sprintf(buf,"%c%c%c%c %c%c%c%c %c%c%c%c %c%c%c%c",    // Create 16-bit binary string to print
    ((loopcntr & 0x8000) ? '1' : '0'),  // Test bit-15
    ((loopcntr & 0x4000) ? '1' : '0'),  // Test bit-14
    ((loopcntr & 0x2000) ? '1' : '0'),  // Test bit-13
    ((loopcntr & 0x1000) ? '1' : '0'),  // Test bit-12
    ((loopcntr & 0x0800) ? '1' : '0'),  // Test bit-11
    ((loopcntr & 0x0400) ? '1' : '0'),  // Test bit-10
    ((loopcntr & 0x0200) ? '1' : '0'),  // Test bit-9
    ((loopcntr & 0x0100) ? '1' : '0'),  // Test bit-8
    ((loopcntr & 0x0080) ? '1' : '0'),  // Test bit-7
```

```
        ((loopcntr & 0x0040) ? '1' : '0'),  // Test bit-6
        ((loopcntr & 0x0020) ? '1' : '0'),  // Test bit-5
        ((loopcntr & 0x0010) ? '1' : '0'),  // Test bit-4
        ((loopcntr & 0x0008) ? '1' : '0'),  // Test bit-3
        ((loopcntr & 0x0004) ? '1' : '0'),  // Test bit-2
        ((loopcntr & 0x0002) ? '1' : '0'),  // Test bit-1
        ((loopcntr & 0x0001) ? '1' : '0')); // Test bit-0

      TERMPORT.print(buf);    //  and print it at current cursor location

      // Add live analog input data
      for(i=0; i<5; i++)
      {
        locate(13+i,14);                          // Locate cursor beyond pre-placed text
        sprintf(buf,"%4d", analogRead(i));   // Create a string from Reading an analog input
        TERMPORT.print(buf);                   //  and print it at current cursor location
      }

      // Demonstrate conditional text
      locate(11, 47);                             // Locate cursor beyond pre-placed text
      if (loopcntr & 0x0040)                  // Test bit 6 of counter
       sprintf(buf,"set    ");
      else
       sprintf(buf,"not set");

      TERMPORT.print(buf);

      loopcntr++; // Increment the loop counter

    } while ( !(TERMPORT.available()));      // Repeat live data loop until a key is pressed


    // Clean-up before exit
    TERMPORT.read();       // Read and discard the keypress entered on the terminal by user
    cursormode(1);             // Enable the cursor
}


//****************************************************
// Clear the entire terminal screen
//****************************************************
void cls (void)
{
  TERMPORT.write(ESC);    // Send escape character
  TERMPORT.print("[2J");
}


//********************************************************
// Control display of terminal cursor
// Call with disp as 1 to enable cursor or 0 to hide it
//********************************************************
void cursormode (bool disp)
{
  TERMPORT.write(ESC);    // Send escape character
  TERMPORT.print( (disp ? "[?25h" : "[?25l") );
}


//****************************************************
// locate cursor on terminal screen
//****************************************************
void locate (byte row, byte col)
{
  TERMPORT.write(ESC);
  TERMPORT.print("[");
  TERMPORT.print(row);
  TERMPORT.print(";");
  TERMPORT.print(col);
  TERMPORT.print("H");
```

```
}


//******************************************************
// Set a terminal attribute
//******************************************************
void terminal_attribute (char attribute)
{
   TERMPORT.write(ESC);    // Send escape character
   TERMPORT.print("[");
   TERMPORT.print(attribute);
   TERMPORT.print("m");
}


//******************************************************
// Make terminal emit beep sound
//******************************************************
void beep (void)
{
  TERMPORT.write(BEL);    // Send BELL control code
}


//*************************************************************
// Display double-width, single-height string at row, column
//*************************************************************
void midtext (byte row, byte col, const char * strpntr)
{
  locate(row,col);                   // Locate the cursor
  TERMPORT.write(ESC);    // Send escape sequence
  TERMPORT.print("#6");

  TERMPORT.print(strpntr); //  and print string it at current cursor location

  terminal_attribute(ATTR_RESET);  // Reset all attributes
}

//************************************************
// Display double-height string at row, column
// These strings occupy two rows
//************************************************
void bigtext (byte row, byte col, const char * strpntr)
{
  locate(row,col);                      // Locate the cursor
  TERMPORT.write(ESC);        // Send escape sequence
  TERMPORT.print("#3");          //  to display string top-half
  TERMPORT.print(strpntr);      // Send string to terminal

  locate(row+1,col);                  // Locate the cursor (down one row)
  TERMPORT.write(ESC);       // Send escape sequence
  TERMPORT.print("#4");         //  to display string bottom-half
  TERMPORT.print(strpntr);     // Send string to terminal

  terminal_attribute(ATTR_RESET);  // Reset all attributes
}
```