Users's Manual for ANSI Terminal Interface

Brent Seidel
Phoenix, AZ

March 30, 2025

Note that this is a draft version and not the final version for publication.

Contents

Chapter 1

Introduction

1.1 About the Project

The intent of this project is to provide assistance in generating ANSI escape sequences for enhancing terminal interfaces. Most of these are done using string constants, but a few functions and procedures are also defined. In most cases the lowest common denominator (*VT100*) is targeted, but the VT100 does not support colors. These routines have been testing using the MacOS terminal program that reports as *VT100* and xTerm from the XQuartz package, which reports as *VT420*.

1.2 License

Chapter 2

How to Obtain

This package is currently available on GitHub at `https://github.com/BrentSeidel/ANSITerm`

2.1 Dependencies

The only dependencies for this project are the following standard Ada Libraries.

2.1.1 Ada Libraries

The following Ada libraries are used:

- `Ada.Calendar`
- `Ada.Strings.Unbounded`
- `Ada.Text_IO`

Chapter 3

Usage Instructions

3.1 Using Alire

This package has not yet been submitted to alire. Until then, use the `gprbuild` instructions.

Alire automatically handles dependencies. To use this in your project, just issue the command "`alr with ansiterm`" in your project directory. To build the standalone CLI program, first obtain the cli using "`alr get ansiterm`". Change to the appropriate directory and use "`alr build`" and "`alr run`".

3.2 Using `gprbuild`

This is a library of routines intended to be used by some program. To use these in your program, edit your `*.gpr` file to include a line to `with` the path to `ansiterm_noalr.gpr`. Then in your Ada code `with` in the package(s) you need and use the routines.

Chapter 4

API Description

There are some functions and procedures, but most of the contents is in the form of constant strings that can either be output or concatenated with other strings to achieve the desired effect.

4.0.1 Datatypes

There is a routine that sends escape sequences to the terminal to try and identify what kind of terminal it is. The result is a member of the `term_type` enumeration. The possible results are:

- *`unknown`* – Terminal type could not be determined
- *VT100*
- *VT101*
- *VT102*
- *VT125*
- *VT131*
- *VT132*
- *VT220*
- *VT240*
- *VT320*
- *VT330*
- *VT340*
- *VT382*
- *VT420*
- *VT510*

- *VT520*
- *VT525*

4.1 Constant Strings

4.1.1 Single Character Control Characters

Constants have been defined for some of the single character control characters:

- *bell* – Ctrl-G, ring terminal bell
- *bs* – Ctrl-H, backspace
- *tab* – Ctrl-I, tab
- *lf* – Ctrl-J, line feed
- *cr* – Ctrl-M, carriage return
- *so* – Ctrl-N, shift out, used to select the G1 character set
- *si* – Ctrl-O, shift in, used to select the G0 character set

4.1.2 Escape and Escape Sequence Prefixes

- *esc* – The escape character
- *csi* – Control Sequence Introducer(ESC [)
- *dcs* – Device Control String (ESC P)
- *osc* – Operating System Command (ESC])

4.1.3 Select Character Set

There are several more that are not supported by the VT100. Some later terminals also support G2 and G3 character sets.

- *g0_uk* – Select the UK character set for G0
- *g0_us* – Select the US character set for G0
- *g0_sym* – Select the symbol and line drawing character set for G0
- *g1_uk* – Select the UK character set for G1
- *g1_us* – Select the US character set for G1
- *g1_sym* – Select the symbol and line drawing character set for G1

4.1.4 VT100 Symbols and Line Drawing Characters

There are a few symbols that are not included in this list. They may be added later.

- *symHoriz* – Horizontal line
- *symVert* – Vertical line
- *symLeftT* – Left side T intersection
- *symRightT* – Right side T intersection
- *symUpperT* – Upper T intersection
- *symLowerT* – Lower T intersection
- *symCornerUL* – Upper left corner
- *symCornerUR* – Upper right corner
- *symCornerLL* – Lower left corner
- *symCornerLR* – Lower right corner
- *symCross* – Horizontal and vertical line crossing
- *symHT* – Symbol for horizontal tab (HT)
- *symFF* – Symbol for form feed (FF)
- *symCR* – Symbol for carriage return (CR)
- *symLF* – Symbol for line feed (LF)
- *symDegree* – Degree symbol (°)
- *symPlusMin* – Plus or minus symbol (±)
- *symNL* – Symbol for null (NL)
- *symVT* – Symbol for vertical tab (VT)
- *symLE* – Symbol for less than or equals (≤)
- *symGE* – Symbol for greater than or equals (≥)
- *symPi* – Symbol for Pi (π)
- *symNE* – Symbol for not equals (≠)

4.1.5 Character Formatting Codes

These constants will need to be concatenated together to create an escape sequence for the desired format. The format of a character mode escape sequence is: *csi* format1 ; format2 ; ... ; formatn *chMode*.

- *chMode* – Terminates a format escape sequence
- *chReset* – Resets all formatting to default
- *chBold* – Set boldface formatting
- *chDim* – Set dim formatting
- *chItalic* – Set italic formatting
- *chUnderline* – Set underline formatting
- *chBlink* – Set blink formatting
- *chInverse* – Set inverse video formatting
- *chHidden* – Set hidden formatting
- *chStrike* – Set strikethrough formatting
- *chNoBold* – Clears bold (and probably dim) formatting
- *chNoDim* – Clears dim (and probably bold) formatting
- *chNoItalic* – Clears italic formatting
- *chNoUnderline* – Clears underline formatting
- *chNoBlink* – Clears blinking
- *chNoInverse* – Clears inverse video
- *chNoHidden* – Clears hidden formatting
- *chNoStrike* – Clears strikethrough formatting
- *fgBlack* – Set foreground color to Black
- *fgRed* – Set foreground color to Red
- *fgGreen* – Set foreground color to Green
- *fgYellow* – Set foreground color to Yellow
- *fgBlue* – Set foreground color to Blue
- *fgMagenta* – Set foreground color to Magenta
- *fgCyan* – Set foreground color to Cyan

- *fgWhite* – Set foreground color to White
- *fgDefault* – Set foreground color to the default color
- *bgBlack* – Set the background color to Black
- *bgRed* – Set the background color to Red
- *bgGreen* – Set the background color to Green
- *bgYellow* – Set the background color to Yellow
- *bgBlue* – Set the background color to Blue
- *bgMagenta* – Set the background color to Magenta
- *bgCyan* – Set the background color to Cyan
- *bgWhite* – Set the background color to White
- *bgDefault* – Set the background color to the default color

There are additional color sequences for bright colors, 256 color and RBG color sequences that have not been added here. They may be added at some point. It may also be simpler to use a function to generate some of these color sequences.

4.1.6 Useful Sequences

These are complete escape sequences and do not need to combined to build an escape sequence.

- *red* – Set text foreground color to bold red
- *blue* – Set text foreground color to bold blue
- *green* – Set text foreground color to bold green
- *yellow* – Set text foreground color to bold yellow
- *cyan* – Set text foreground color to bold cyan
- *magenta* – Set text foreground color to bold magenta
- *white* – Set text foreground color to bold white
- *rst* – Reset all text formatting to default
- *cls* – Move the cursor to the home (top left) position and clear the screen
- *reqPos* – Request the current cursor position (intended for use in a function)
- *reqAttr* – Request the primary device attributes (intended for use in a function)

4.2 Functions and Procedures

Returns a `String` that contains an escape sequence to position the cursor to the specified line (or row) and column. The top left corner is position (1,1) and rows increase downwards and columns increase rightward.

```
function posCursor(Line, Column : Natural) return String;
```

Returns a `String` that will draw a box on the screen. If the parameter `line` is true, the box will use the DEC line drawing characters, otherwise dashes, vertical bars, and plus signs will be used.

```
function drawBox(row, col, height, width : Natural; line : Boolean) return String;
```

Returns a `String` that will fill a box with the character specified in `c`. The box parameters are the same for the `drawBox` function.

```
function fillBox(row, col, height, width : Natural; c : Character) return String;
```

Returns a `String` containing the entered character or escape sequence. The parameter `d` contains a duration. If the duration expires with no characters entered, an empty string is entered.

```
function getCharOrEscape(d : Duration) return String;
```

Attempts to determine the size of the terminal window (or actual terminal if connected to a physical terminal). This is done by setting the cursor to row and column 9999 and requesting the current cursor position. If no position can be determined, `rows` and `cols` will be set to 0.

```
procedure getSize(rows : out Natural; cols : out Natural);
```

Attempts to identify the terminal type. Returns a value of `term_type`. If no type can be determined, ***`unknown`*** is returned.

```
function identify return term_type;
```

Chapter 5

Other Stuff

If there is anything else that should be added, additional chapters may be added as needed.

Bibliography

[1] J.G.P. Barnes. *Programming in Ada plus Language Reference Manual*. Addison-Wesley Publishing Company, 1991.

[2] John Barnes. *Programming in Ada 2012*. Cambridge University Press, 2014.

[3] John Barnes. *Programming in Ada 2022*. Cambridge University Press, 2024.

[4] XFree86 Project. Xterm control sequences, 12 2024.