ANSI escape code

ANSI escape sequences are characters embedded in the text used to control formatting, color, and other output options on video text terminals. Almost all terminal emulators designed to show text output from a remote computer, and (except for Microsoft Windows) to show text output from local software, interpret at least some of the ANSI escape sequences.

History

Almost all manufacturers of video terminals added vendor-specific escape sequences to do operations such as placing the cursor at arbitrary positions on the screen. As these sequences were all different, elaborate libraries such as termcap had to be created so programs could use the same API for all of them. In addition, most designs required sending numbers (such as row & column) as the binary values of the characters; for some programming languages and for systems that did not use ASCII internally it was often difficult or impossible to turn a number into the correct character.

The first standard for **ANSI escape sequences** was ECMA-48, adopted in 1976. It was a continuation of a series of character coding standards, the first one being ECMA-6 from 1961, a 7-bit standard from which ASCII originates. ECMA-48 has been updated several times and the current edition is the 5th from 1991. It is also adopted by ISO and IEC as standard **ISO/IEC 6429**. The name "ANSI escape sequence" dates from 1981 when ANSI adopted ECMA-48 as the standard **ANSI X3.64** (and later, in 1997, withdrew it).[1]

The first popular video terminal to support these sequences was the Digital VT100 introduced in 1978,[2] which sparked a variety of "clones", among the earliest and most popular of which was the much more affordable Zenith Z-19 in 1979.[3] The popularity of these gradually led to more and more software (especially bulletin board systems) assuming the escape sequences worked, leading to almost all new terminals and emulator programs supporting them.

Support

Most terminal emulators running on Unix-like systems (such as xterm and the OS X Terminal) interpret ANSI escape sequences. The Linux console (the text seen when X is not running) also interprets them. Terminal programs for Microsoft Windows designed to show text from an outside source (a serial port, modem, or socket) also interpret them. Some support for text from local programs on Windows is offered through alternate command processors such as JP Software's TCC (formerly 4NT), Michael J. Mefford's ANSI.COM, and Jason Hood's ansicon.

Many Unix console applications (e.g., ls, grep, Vim, and Emacs) can generate them. Utility programs such as tput output them, as well as in low-level programming libraries, such as termcap or terminfo, or a higher-level library such as curses.

Windows and DOS

MS-DOS 1.0 did not support the ANSI or any other escape sequences. Only a few control characters (CR, LF, BS) were interpreted, making it impossible to do any kind of full-screen application. Any display effects had to be done with BIOS calls (or far more often by directly manipulating the IBM PC hardware).

MS-DOS 2.0 introduced the ability to add a device driver for the ANSI escape sequences – the *de facto* standard being `ANSI.SYS`, but others like `ANSI.COM`,[4] and `NANSI.SYS`[5] are used as well. Extreme slowness and the fact that it was not installed by default made usage by software almost non-existent; software continued to directly manipulate the hardware to get the text display needed. ANSI.SYS and similar drivers continued to work in through Windows 98, and even in Windows Me, when they were set to native DOS mode (the characters taking the whole screen). ANSI.SYS also works in NT-derived systems for 16-bit legacy programs executing under the NTVDM.

The Win32 console does not support ANSI escape sequences at all. Software can manipulate the console with the ioctl-like Console API [6] interlaced with the text output. Some software internally interprets ANSI escape sequences in text being printing and translates them to these calls.

Sequence elements

Escape sequences start with the character **ESC** (ASCII decimal 27/hex 0x1B/octal 033). For two character sequences, the second character is in the range ASCII 64 to 95 (@ to _). However, most of the sequences are more than two characters, and start with the characters **ESC** and **[** (left bracket). This sequence is called **CSI** for **Control Sequence Introducer** (or Control Sequence Initiator). The final character of these sequences is in the range ASCII 64 to 126 (@ to ~).

There is a single-character CSI (155/0x9B/0233) as well. The **ESC+[** two-character sequence is more often used than the single-character alternative, for details see C0 and C1 control codes. Only the two-character sequence is recognized by devices that support just ASCII (7-bit bytes) or devices that support 8-bit bytes but use the 0x80–0x9F control character range for other purposes. On terminals that use UTF-8 encoding, both forms take 2 bytes (CSI in UTF-8 is 0xC2, 0x9B)*Talk:ANSI escape code#assertion about C1 controls in UTF-8* but the **ESC+[** sequence is clearer.

Though some encodings use multiple bytes per character, the following discussion is restricted to ASCII characters, and so assumes each character is directly represented by a single byte.

Non-CSI codes

Note: other C0 codes besides ESC — commonly BEL, BS, CR, LF, FF, TAB, VT, SO, and SI — may produce similar or identical effects to some control sequences when output.

ESC N = SS2, ESC O = SS3 — select a single character from one of the alternate character sets.

ESC ^ = PM, ESC _ = APC — these each take a single string of text, terminated by ST (ESC \ ). They are ignored by xterm.

ESC P = DCS: Device control string, ESC] = OSC: Operating system command — these are similar to CSI, but not limited to integer arguments. Because they are frequently used, in many cases BEL is an acceptable alternative to ST. E.g., in xterm, the window title can be set by: "OSC0;this is the window titleBEL"

Note: pressing special keys on the keyboard, as well as outputting many xterm CSI, DCS, or OSC sequences, often produces a CSI, DCS, or OSC sequence.

CSI codes

The general structure of most ANSI escape sequences is `CSI [`*`private mode characters(s?)`*`]` *`n1`* `;` *`n2`*`... [`*`trailing intermediate character(s?)`*`]` *`letter`*. The final byte, modified by private mode characters and trailing intermediate characters, specifies the command. The numbers are optional parameters. The default value used for omitted parameters varies with the command, but is usually 1 or 0. If trailing parameters are omitted, the trailing semicolons may also be omitted.

The final byte is technically any character in the range 64 to 126 (hex 0x40 to 0x7e, ASCII @ to ~), and may be modified with leading intermediate bytes in the range 32 to 47 (hex 0x20 to 0x2f).

The colon (0x3a) is the only character not a part of the general sequence; it was left for future standardization, so any sequence containing it should be ignored. Although multiple private mode characters or trailing intermediates are permitted, there are no such known usages.

If there are any leading private mode characters, the main body of the sequence could theoretically contain any order of characters 0x30 – 0x3f instead of a well-formed semicolon-separated list of numbers, but all known terminals are nice and just use them as a flag. Sequences are also private if the final byte is in the range 112 to 126 (hex 0x70 to 0x7e, ASCII *p* to ~).

Examples of private escape codes include the **DECTCEM** (DEC text cursor enable mode) shown below. It was first introduced for the VT-300 series of video terminals.

The existence of a C0 control, DEL (0x7f), or a high characters is undefined. Typically, implementations will either cancel the sequence or execute the control and then continue parsing the CSI sequence.

Some ANSI escape sequences (not a complete list)

Code	Name	Effect
CSI *n* A	CUU – Cursor Up	Moves the cursor *n* (default 1) cells in the given direction. If the cursor is already at the edge of the screen, this has no effect.
CSI *n* B	CUD – Cursor Down	
CSI *n* C	CUF – Cursor Forward	
CSI *n* D	CUB – Cursor Back	
CSI *n* E	CNL – Cursor Next Line	Moves cursor to beginning of the line *n* (default 1) lines down.
CSI *n* F	CPL – Cursor Previous Line	Moves cursor to beginning of the line *n* (default 1) lines up.
CSI *n* G	CHA – Cursor Horizontal Absolute	Moves the cursor to column *n*.
CSI *n* ; *m* H	CUP – Cursor Position	Moves the cursor to row *n*, column *m*. The values are 1-based, and default to 1 (top left corner) if omitted. A sequence such as CSI ;5H is a synonym for CSI 1;5H as well as CSI 17;H is the same as CSI 17H and CSI 17;1H
CSI *n* J	ED – Erase Data	Clears part of the screen. If *n* is zero (or missing), clear from cursor to end of screen. If *n* is one, clear from cursor to beginning of the screen. If *n* is two, clear entire screen (and moves cursor to upper left on MS-DOS ANSI.SYS).
CSI *n* K	EL – Erase in Line	Erases part of the line. If *n* is zero (or missing), clear from cursor to the end of the line. If *n* is one, clear from cursor to beginning of the line. If *n* is two, clear entire line. Cursor position does not change.
CSI *n* S	SU – Scroll Up	Scroll whole page up by *n* (default 1) lines. New lines are added at the bottom. (not ANSI.SYS)
CSI *n* T	SD – Scroll Down	Scroll whole page down by *n* (default 1) lines. New lines are added at the top. (not ANSI.SYS)
CSI *n* ; *m* f	HVP – Horizontal and Vertical Position	Moves the cursor to row *n*, column *m*. Both default to 1 if omitted. Same as CUP
CSI *n* [;*k*] m	SGR – Select Graphic Rendition	Sets SGR parameters, including text color. After CSI can be zero or more parameters separated with ;. With no parameters, CSI m is treated as CSI 0 m (reset / normal), which is typical of most of the ANSI escape sequences.
CSI 6 n	DSR – Device Status Report	Reports the cursor position to the application as (as though typed at the keyboard) ESC[*n*;*m*R, where *n* is the row and *m* is the column. (May not work on MS-DOS.)
CSI s	SCP – Save Cursor Position	Saves the cursor position.
CSI u	RCP – Restore Cursor Position	Restores the cursor position.
CSI ?25l	DECTCEM	Hides the cursor. (Note: the trailing character is lowercase L.)
CSI ?25h	DECTCEM	Shows the cursor.

SGR (Select Graphic Rendition) parameters

Code	Effect	Note
0	Reset / Normal	all attributes off
1	Bold or increased intensity	
2	Faint (decreased intensity)	not widely supported
3	Italic: on	not widely supported. Sometimes treated as inverse.
4	Underline: Single	
5	Blink: Slow	less than 150 per minute
6	Blink: Rapid	MS-DOS ANSI.SYS; 150 per minute or more; not widely supported
7	Image: Negative	inverse or reverse; swap foreground and background (reverse video)
8	Conceal	not widely supported
9	Crossed-out	Characters legible, but marked for deletion. Not widely supported.
10	Primary(default) font	
11–19	*n*-th alternate font	Select the *n*-th alternate font. 14 being the fourth alternate font, up to 19 being the 9th alternate font.
20	Fraktur	hardly ever supported
21	Bold: off or Underline: Double	bold off not widely supported, double underline hardly ever
22	Normal color or intensity	neither bold nor faint
23	Not italic, not Fraktur	
24	Underline: None	not singly or doubly underlined
25	Blink: off	
26	Reserved	
27	Image: Positive	
28	Reveal	conceal off
29	Not crossed out	
30–37	Set text color	30 + x, where x is from the color table below
38	Set xterm-256 text color	next arguments are 5;x where x is color index (0..255)
39	Default text color	implementation defined (according to standard)
40–47	Set background color	40 + x, where x is from the color table below
48	Set xterm-256 background color	next arguments are 5;x where x is color index (0..255)
49	Default background color	implementation defined (according to standard)
50	Reserved	
51	Framed	
52	Encircled	
53	Overlined	
54	Not framed or encircled	
55	Not overlined	
56–59	Reserved	
60	ideogram underline or right side line	hardly ever supported

61	ideogram double underline or double line on the right side	hardly ever supported
62	ideogram overline or left side line	hardly ever supported
63	ideogram double overline or double line on the left side	hardly ever supported
64	ideogram stress marking	hardly ever supported
90–99	Set foreground color, high intensity	aixterm (not in standard)
100–109	Set background color, high intensity	aixterm (not in standard)

Colors

Text color is manipulated using CSI *n* [;*k*] m sequence. From the section above, you use CSI codes `30+n` to indicate color, where `n` is in the table below. Black would be `\x1b[30m`, red would be `\x1b[31m`, and utilizing the "bold" code, gray would be `\x1b[30;1m` and bold red could be made by printing `\x1b[31;1m` to the terminal. You can cancel the color using `\x1b[0m`.

Color table[7]

Intensity	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Normal	Black	Red	Green	Yellow[8]	Blue	Magenta	Cyan	White
Bright	Black	Red	Green	Yellow	Blue	Magenta	Cyan	White

There are two other color standards CSS/HTML standard colors and X Window colors which standardize both the color names and associated RGB color values, but the escape sequence standard only specifies the color names, not RGB values. The chart below shows default RGB assignments for some common terminal programs, together with the CSS and the X-Window colors for these color names.

	Color name	**Standard VGA colors**	**Windows XP command prompt**	**Terminal.app**	**PuTTY**	**xterm**	**CSS/HTML**	**X Window**
Normal	**Black**	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	Red	170, 0, 0	128, 0, 0	194, 54, 33	187, 0, 0	205, 0, 0	255, 0, 0	255, 0, 0
	Green	0, 170, 0	0, 128, 0	37, 188, 36	0, 187, 0	0, 205, 0	0, 255, 0	0, 128, 0
	Brown/yellow	170, 85, 0	128, 128, 0	173, 173, 39	187, 187, 0	205, 205, 0	255, 255, 0	255, 255, 0
	Blue	0, 0, 170	0, 0, 128	73, 46, 225	0, 0, 187	0, 0, 238	0, 0, 255	0, 0, 255
	Magenta	170, 0, 170	128, 0, 128	211, 56, 211	187, 0, 187	205, 0, 205	255, 0, 255	255, 0, 255
	Cyan	0, 170, 170	0, 128, 128	51, 187, 200	0, 187, 187	0, 205, 205	0, 255, 255	0, 255, 255
	Gray	170, 170, 170	192, 192, 192	203, 204, 205	187, 187, 187	229, 229, 229	255, 255, 255	255, 255, 255
Bright/light	**Darkgray**	85, 85, 85	128, 128, 128	129, 131, 131	85, 85, 85	127, 127, 127		
	Red	255, 85, 85	255, 0, 0	252,57,31	255, 85, 85	255, 0, 0		
	Green	85, 255, 85	0, 255, 0	49, 231, 34	85, 255, 85	0, 255, 0	144, 238, 144	144, 238, 144
	Yellow	255, 255, 85	255, 255, 0	234, 236, 35	255, 255, 85	255, 255, 0	255, 255, 224	225, 255, 224

	Blue	85, 85, 255	0, 0, 255	88, 51, 255	85, 85, 255	92, 92, 255	173, 216, 230	173, 216, 230
	Magenta	255, 85, 255	255, 0, 255	249, 53, 248	255, 85, 255	255, 0, 255		
	Cyan	85, 255, 255	0, 255, 255	20, 240, 240	85, 255, 255	0, 255, 255	224, 255, 255	224, 255, 255
	White	255, 255, 255	255, 255, 255	233, 235, 235	255, 255, 255	255, 255, 255		

The VGA column denotes the typical colors that are used when booting PCs and leaving them in their classical 80×25 text mode. The colors are different in the EGA/VGA graphic modes.

In July 2004, the blue colors of xterm changed,[9] RGB (0,0,205) → (0,0,238) for normal and (0,0,255) → (92,92,255) for bright. As of 2010, old xterm versions still linger on many computers though.

Recent-enough versions of KDE's Konsole program support 24-bit foreground and background color setting:

```
https://github.com/robertknight/konsole/blob/master/tests/color-spaces.pl
Quoting <https://github.com/robertknight/konsole/blob/master/user-doc/README.moreColors>:
  ESC[ … 38;2;<r>;<g>;<b> … m Select RGB foreground color
  ESC[ … 48;2;<r>;<g>;<b> … m Select RGB background color
```

Examples

`CSI 2 J` — This clears the screen and, on some devices, locates the cursor to the y,x position 1,1 (upper left corner).

`CSI 32 m` — This makes text green. On MS-DOS, normally the green would be dark, dull green, so you may wish to enable Bold with the sequence `CSI 1 m` which would make it bright green, or combined as `CSI 32 ; 1 m`. MS-DOS ANSI.SYS uses the Bold state to make the character Bright; also the Blink state can be set (via `INT 10, AX 1003h, BL 00h`) to render the Background in the Bright mode. MS-DOS ANSI.SYS does not support SGR codes 90–97 and 100–107 directly.

`CSI 0 ; 6 8 ; "DIR" ; 13 p` — This re-assigns the key F10 to send to the keyboard buffer the string "DIR" and ENTER, which in the DOS command line would display the contents of the current directory. (MS-DOS ANSI.SYS only) This is a private-use code (as indicated by the letter p), using a non-standard extension to include a string-valued parameter. Following the letter of the standard would consider the sequence to end at the letter D.

`CSI s` — This saves the cursor position. Using the sequence `CSI u` will restore it to the position. Say the current cursor position is 7(y) and 10(x). The sequence `CSI s` will save those two numbers. Now you can move to a different cursor position, such as 20(y) and 3(x), using the sequence `CSI 20 ; 3 H` or `CSI 20 ; 3 f`. Now if you use the sequence CSI u the cursor position will return to 7(y) and 10(x). Some terminals require the DEC sequences `ESC 7` / `ESC 8` instead which is more widely supported.

Example of use in shell scripting

ANSI escape codes are often used in UNIX and UNIX-like terminals to provide syntax highlighting. For example, on compatible terminals, the following *list* command color-codes file and directory names by type.

```
ls --color
```

Users can employ escape codes in their scripts by including them as part of *standard output* or *standard error*. For example, the following *sed* command embellishes the output of the *make* command by displaying lines containing words starting with "ERR" in , and words starting with "WARN" in **bold** (letter case is ignored).

```
make 2>&1 | sed -e 's/.*\bERR.*/\x1b[7m&\x1b[0m/i' -e 's/.*\bWARN.*/\x1b[1m&\x1b[0m/i'
```

The representations of the codes are highlighted.[10]

Invalid and ambiguous sequences in use

- The Linux console uses `OSC P n rr gg bb` to change the palette, which, if hard-coded into an application, may hang other terminals. However, appending `ST` will be ignored by Linux and form a proper, ignorable sequence for other terminals.
- On the Linux console, certain function keys generate sequences of the form `CSI [ char`. The CSI sequence should terminate on the [.
- Old versions of Terminator generates `SS3 1; modifiers char` when F1–F4 are pressed with modifiers. The faulty behavior was copied from GNOME Terminal.
- xterm replies `CSI row ; column R` if asked for cursor position and `CSI 1 ; modifiers R` if the F3 key is pressed with modifiers, which collide in the case of *row* == 1. This can be avoided by using the *?* private modifier, which will be reflected in the response.
- many terminals prepend `ESC` to any character that is typed with the alt key down. This creates ambiguity for uppercase letters and symbols @[\]^_, which would form C1 codes.
- Konsole generates `SS3 modifiers char` when F1–F4 are pressed with modifiers.

Notes

[1] See this NIST list of withdrawn standards (http://www.itl.nist.gov/fipspubs/withdraw.htm)
[2] Paul Williams (2006). "Digital's Video Terminals" (http://vt100.net/vt_history). VT100.net. . Retrieved 17 August 2011.
[3] Heathkit Company (1979). "Heathkit Catalog 1979" (http://www.pestingers.net/Computer_history/Computers_79.htm). Heathkit Company. . Retrieved 4 November 2011.
[4] Michael Mefford (7 February 1989). "ANSI.com: Download It Here" (http://www.pcmag.com/article2/0,2817,5343,00.asp). PC Magazine. . Retrieved 10 August 2011.
[5] Dan Kegel, Eric Auer (28 February 1999). "Nansi and NNansi - ANSI Drivers for MS-DOS" (http://www.kegel.com/nansi/). Dan Kegel's Web Hostel. . Retrieved 10 August 2011.
[6] http://msdn.microsoft.com/en-us/library/ms682087.aspx
[7] The names are standard, however the exact shade/hue/value of colors are not standardized and will depend on the device used to display them.
[8] On terminals based on CGA compatible hardware, such as ANSI.SYS running on DOS, this normal intensity foreground color is rendered as Orange. CGA RGBI monitors contained hardware to modify the dark yellow color to an orange/brown color by reducing the green component. See this ansi art (http://sixteencolors.net/pack/ciapak26/DH-JNS11.CIA) as an example.
[9] "Patch #192 - 2004/7/12 - XFree86 4.4.99.9" (http://invisible-island.net/xterm/xterm.log.html#xterm_192). .
[10] Colorized shell echo (http://www.debian.org/doc/manuals/debian-reference/ch09.en.html#_colorized_shell_echo)

External links

- Standard ECMA-48, Control Functions For Coded Character Sets (http://www.ecma-international.org/publications/standards/Ecma-048.htm). (*5th edition, June 1991*), European Computer Manufacturers Association, Geneva 1991 (also published by ISO and IEC as standard ISO/IEC 6429)
- vt100.net DEC Documents (http://vt100.net/docs/)
- ANSI.SYS -- ansi terminal emulation escape sequences (http://enterprise.aacc.cc.md.us/~rhs/ansi.html)
- Xterm / Escape Sequences (http://invisible-island.net/xterm/ctlseqs/ctlseqs.html)
- AIXterm / Escape Sequences (http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds1/aixterm.htm)
- A collection of escape sequences for terminals that are vaguely compliant with ECMA-48 and friends. (http://bjh21.me.uk/all-escapes/all-escapes.txt)
- ANSI Escape Sequences (http://ascii-table.com/ansi-escape-sequences.php)

Article Sources and Contributors

ANSI escape code *Source*: http://en.wikipedia.org/w/index.php?oldid=537017219 *Contributors*: Abdull, Adammck, Anders Kaseorg, AnonMoos, BWDuncan, Bgagaga, Bluemoose, Brian Patrie, Changaco, Cmglee, Copyeditor42, Cyrius, DXBari, Damian Yerrick, DanielPharos, David Gerard, Davygrvy, Dcoetzee, Dmeranda, DmitryKo, Dra, Dragice, EdC, Electron9, Emperorbma, Fbriere, Fleminra, Frap, Fromageestciel, Ghettoblaster, Gpvos, GregWooledge, HansM, Hatster301, Heinrichmartin, Henning Makholm, Hlovdal, Homerjay, Incnis Mrsi, Interiot, Intgr, Isheden, Ivan Shmakov, Iwmt, J l bradley, JLaTondre, Jengelh, Jim1138, Jmgonzalez, Jricjr, Jwfearn, Keka, KiloByte, Kirun, Krazymike, Kreline, Kristjan.Jonasson, Locke Cole, MDGx, Matthiaspaul, MegaSloth, Mfwitten, MichaelRWolf, Mild Bill Hiccup, Mindmatrix, Mirokado, Nabla, Nasa-verve, Niceguyjoey, Nickj, Nmagedman, Oerjan, Ohconfucius, Optimix, Psychonaut, R'n'B, R.123, Random832, Reep, ReyBrujo, Riddley, Rwwww, Scandum, Shawnhcorey, SimonOwen, Sperling, Spitzak, Stevage, StuartBrady, Tedickey, The Interior, Tomalak geretkal, Toresbe, Tortillovsky, Towopedia, Tripodics, TuukkaH, Updatebjarni, Viznut, Wapcaplet, Wlindley, ZeroOne, 134 anonymous edits

License