ANSI escape code

From Wikipedia, the free encyclopedia

In computing, **ANSI escape codes** (or **escape sequences**) are a method using in-band signaling to control the formatting, color, and other output options on video text terminals. To encode this formatting information, certain sequences of bytes are embedded into the text, which the terminal looks for and interprets as commands, not as character codes.

ANSI codes were introduced in the 1970s and became widespread in the minicomputer/mainframe market by the early 1980s. They were used by the nascent bulletin board system market to offer improved displays compared to earlier systems lacking cursor movement, leading to even more widespread use.

Although hardware text terminals have become increasingly rare in the 21st century, the relevance of the ANSI standard persists because most terminal emulators interpret at least some of the ANSI escape sequences in the output text. One notable exception was the Win32 console component of Microsoft Windows before Windows 10 update TH2.[1]

Contents

History

Almost all manufacturers of video terminals added vendor-specific escape sequences to perform operations such as placing the cursor at arbitrary positions on the screen. One example is the VT52 terminal, which allowed the cursor to be placed at an x,y location on the screen by sending the ESC character, a Y character, and then two characters representing with numerical values equal to the x,y location plus 32 (thus starting at the ASCII space character and avoiding the control characters).

As these sequences were different for different platforms, elaborate libraries such as termcap had to be created so programs could use the same API to work with any terminal. Most of these systems required sending numbers (such as row and 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 ANSI standard attempted to address these problems by making a command set that all terminals would use and requiring all numeric information to be transmitted as ASCII numbers. The first standard in the series was ECMA-48, adopted in 1976. It was a continuation of a series of character coding standards, the first one being ECMA-6 from 1965, a 7-bit standard from which ISO 646 originates. The name "ANSI escape sequence" dates from 1979 when ANSI adopted ANSI X3.64. The ANSI X3L2 committee collaborated with the ECMA committee TC 1 to produce nearly identical standards. These two standards were merged into an international standard, ISO 6429.[2] In 1994, ANSI withdrew its standard in favor of the international standard.

The first popular video terminal to support these sequences was the Digital VT100, introduced in 1978.[3] This model was very successful in the market, which sparked a variety of VT100 clones, among the earliest and most popular of which was the much more affordable Zenith Z-19 in 1979.[4] Others included the Qume QVT-108, Televideo TVI-970, Wyse WY-99GT as well as optional "VT100" or "VT103" or "ANSI" modes with varying degrees of compatibility on many other brands. 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.

In 1981, ANSI X3.64 was adopted for use in the US government by FIPS publication 86. Later, the US government stopped duplicating industry standards, so FIPS pub. 86 was withdrawn.[5]

ECMA-48 has been updated several times and is currently at its 5th edition, from 1991. It is also adopted by ISO and IEC as standard **ISO/IEC 6429**.

Platform support

The widespread use of ANSI by bulletin boards and online services led to almost universal platform support by the mid 1980s. In most cases this took the form of a terminal emulator (such as xterm on Unix or the OS X Terminal or ZTerm on MacOS and many communication programs for the IBM PC), although there was increasing support in the standard text output of many operating systems.

Unix and the AmigaOS all included some ANSI support in the OS, which led to widespread use of ANSI by programs running on those platforms. Unix-like operating systems could produce ANSI codes through libraries such as termcap and curses used by many pieces of software to update the display. These libraries are supposed to support non-ANSI terminals as well, but this is so rarely tested nowadays that they are unlikely to work. Many games and shell scripts (such as colored prompts) directly write the ANSI sequences and thus cannot be used on a terminal that does not interpret them.

AmigaOS not only interprets ANSI code sequences for text output to the screen, the AmigaOS printer driver also interprets them (with extensions proprietary to AmigaOS) and translates them into the codes required for the particular printer that is actually attached.[6]

In spite of its popularity, ANSI codes were not universally supported. Support was not built-in on the original "classic" Mac OS, while the Atari ST used the command system adapted from the VT52 with some expansions for color support.[7]

Windows and DOS

MS-DOS 1.x did not support the ANSI or any other escape sequences. Only a few control characters (BEL, CR, LF, BS) were interpreted by the underlying BIOS, making it almost[8] impossible to do any kind of full-screen application. Any display effects had to be done with BIOS calls, which were notoriously slow, or by directly manipulating the IBM PC hardware.

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`,[9] `NANSI.SYS`[10] and `ANSIPLUS.EXE` are used as well (these are considerably faster as they bypass the BIOS). Slowness and the fact that it was not installed by default made software rarely take advantage of it; instead, applications continued to directly manipulate the hardware to get the text display needed. ANSI.SYS and similar drivers continued to work in Windows 9x up to Windows Me, and in NT-derived systems for 16-bit legacy programs executing under the NTVDM.

The Win32 console did not support ANSI escape sequences at all until Windows 10 "Threshold 2".[1] Some replacements or additions for the console window such as JP Software's TCC (formerly 4NT), Michael J. Mefford's ANSI.COM, Jason Hood's `ANSICON`[11] and Maximus5's ConEmu do interpret ANSI escape sequences printed by programs.

Some software internally interprets ANSI escape sequences in text being printed and translates them to calls to manipulate the color and cursor position in the command output window,[12] to make it easier to port software using ANSI to Windows.

Sequence elements

Sequences have different length. All sequences start with the character **`ESC`** (ASCII decimal `27`/hex `0x1B`/octal `033`) followed by a second character in the range ASCII `64–95` (`@` to _/hex `0x40–0x5F`).[13]:5.3.a

In 8-bit environments, these two character sequences can be merged into single byte, using the `0x80–0x9F` control character range.[13]:5.3.b Devices that support just ASCII (7-bit bytes) recognize only the two-character sequence. The same is true for 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 (the Control Sequence Introducer (CSI) in UTF-8 is `0xC2`, `0x9B`) but the `ESC[` sequence is clearer.

However, most of the sequences are more than two characters and start with the characters `ESC` and `[` (left bracket/`0x5B`). On 8-bit systems there is a single-character (`155`/`0x9B` /`0233`) as well, still the `ESC[` two-character sequence is more often used than the single-character alternative (for details see C0 and C1 control codes).[13]:5.4.a This sequence is called **CSI** for **Control Sequence Introducer** (or Control Sequence Initiator). The final character

of these sequences is in the range ASCII `64–126` (`@` to ~/hex `0x40–0x7E`).[13]:5.4 Sequences that begin with the escape character are named escape sequences. Sequences beginning with the CSI are named control sequences. A sequence may be both an escape sequence and a control sequence.

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

Non-CSI codes

The following describe some of the control sequences that do not begin with the CSI sequence. Note that 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.

Some non-CSI escape sequences (not a complete list)

Escape Sequence	Single-Byte Equivalent	Name	Effect
ESC N	0x8e	SS2 – Single Shift Two	Select a single character from one of the alternate character sets. In xterm, SS2 selects the G2 character set, and SS3 selects the G3 character set.[14]
ESC O	0x8f	SS3 – Single Shift Three	
ESC P	0x90	DCS – Device Control String	Controls a device. In xterm, uses of this sequence include defining User-Defined Keys, and requesting or setting Termcap/Terminfo data.[14]
ESC \	0x9c	ST – String Terminator	Terminates strings in other controls (including APC, DCS, OSC, PM, and SOS).[13]:8.3.143
ESC]	0x9d	OSC – Operating System Command	Starts a control string for the operating system to use. OSC sequences are similar to CSI sequences, but are not limited to integer arguments. In general, these control sequences are terminated by ST.[13]:8.3.89 In xterm, they may also be terminated by BEL.[14] For example, in xterm, the window title can be set by `OSC 0;this is the window title BEL`.
ESC X	0x98	SOS – Start of String	Takes an argument of a string of text, terminated by ST. The uses for these string control sequences are defined by the application[13]:8.3.2,8.3.128 or privacy discipline.[13]:8.3.94 These functions are not implemented and the arguments are ignored by xterm.[14]
ESC ^	0x9e	PM – Privacy Message	
ESC _	0x9f	APC – Application Program Command	
ESC c		RIS – Reset to Intitial State	Resets the device to its original state. This may include (if applicable): reset graphic rendition, clear tabulation stops, reset to default font, and more.

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 character(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 `64–126` (hex `0x40–0x7E`, ASCII `@` to `~`), and may be modified with leading intermediate bytes in the range `32` to `47` (hex `0x20–0x2F`, ASCII `space` to `/`).

The colon (`58`, hex `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 in the range `48–63` (hex `0x30–0x3F`, ASCII `0` to `?`) instead of a well-formed semicolon-separated list of numbers, but all known terminals use the non-digit characters in this range as flags. Sequences are also private if the final byte is in the range `112–126` (hex `0x70–0x7E`, ASCII `p–~`).

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 behavior of the terminal is undefined in the case where a CSI sequence contains any character outside of the range `32` to `126` (hex `0x20–0x7E`, ASCII `space–~`). These illegal characters are either C0 control characters (the range `0–31`, hex `0x00–0x1F`), character `127` (hex `0x7F`, ASCII `DEL`), or high-ASCII characters (the range `128–255`, hex `0x80–0xFF`).

Possibilities for handling illegal characters in a CSI sequence include:

1. Assuming the end of the CSI sequence, ignoring it and treating further characters as data;

2. Ignoring this sequence including all future characters through the next character that would normally end a CSI sequence (anything in the range `64–126` (hex `0x40–0x7E`, ASCII`@–~`)); or

3. Processing any control code as the terminal normally would outside of a CSI sequence before continuing to parse the rest of the sequence.

Some ANSI control codes (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	
	CUF –	

CSI `n` C	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. (not ANSI.SYS)
CSI `n` F	CPL – Cursor Previous Line	Moves cursor to beginning of the line ***n*** (default `1`) lines up. (not ANSI.SYS)
CSI `n` G	CHA – Cursor Horizontal Absolute	Moves the cursor to column ***n*** (default `1`). (not ANSI.SYS)
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 in Display	Clears part of the screen. If ***n*** is `0` (or missing), clear from cursor to end of screen. If ***n*** is `1`, clear from cursor to beginning of the screen. If ***n*** is `2`, clear entire screen (and moves cursor to upper left on DOS ANSI.SYS). If ***n*** is `3`, clear entire screen and delete all lines saved in the scrollback buffer (this feature was added for xterm and is supported by other terminal applications).
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` m	SGR – Select Graphic	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),

	Rendition	which is typical of most of the ANSI escape sequences.
CSI 5i	AUX Port On	Enable aux serial port usually for local serial printer
CSI 4i	AUX Port Off	Disable aux serial port usually for local serial printer
CSI 6n	DSR – Device Status Report	Reports the cursor position (CPR) to the application as (as though typed at the keyboard) `ESC[n;mR`, where n is the row and m is the column.)
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; 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 supported.
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 (foreground)	`30` + n, where n is from the color table below
38	Reserved for extended set foreground color	typical supported next arguments are `5;n` where n is color index (`0..255`) or `2;r;g;b` where r, g, b are red, green and blue color channels (out of `255`)
39	Default text color (foreground)	implementation defined (according to standard)
40–47	Set background color	`40` + n, where n is from the color table below
48	Reserved for extended set background color	typical supported next arguments are `5;n` where n is color index (`0..255`) or `2;r;g;b` where r, g, b are red, green and blue color channels (out of `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
65	ideogram attributes off	hardly ever supported, reset the effects of all of `60–64`
90–	Set foreground text color,	

97	high intensity	aixterm (not in standard)
100–107	Set background color, high intensity	aixterm (not in standard)

Colors

The original specification only had 8 colors, and just gave them names. The SGR parameters 30-37 selected the foreground color, while 40-47 selected the background. Quite a few terminals implemented "bold" (SGR code 1) as a brighter color rather than a different font, thus providing 8 additional foreground colors. Usually you could not get these as background colors, though sometimes inverse video (SGR code 7) would allow that. Examples: to get black letters on white background use `ESC[30;47m`, to get red use `ESC[31m`, to get bright red use `ESC[31;1m`. To reset colors to their defaults, use `ESC[39;49m` (not supported on some terminals), or reset all attributes with `ESC[0m`.

Color table[15]

Intensity	0	1	2	3	4	5	6	7
Normal	Black	Red	Green	Yellow[16]	Blue	Magenta	Cyan	White
Bright	Black	Red	Green	Yellow	Blue	Magenta	Cyan	White

When hardware started using 8-bit DACs several pieces of software assigned 24-bit color numbers to these names. The chart below shows default RGB assignments for some common terminal programs, together with the CSS and the X Window System colors for these color names.

	Color name	Standard VGA colors	Windows XP CMD	Terminal.app	PuTTY	mIRC	xterm	CSS/HTML	X
Normal	**Black**	0, 0, 0	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	127, 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, 147, 0	0, 205, 0	0, 255, 0	0, 128, 0
	Brown/yellow	170, 85, 0	128, 128, 0	173, 173, 39	187, 187, 0	252, 127, 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, 127	0, 0, 238	0, 0, 255	0, 0, 255
	Magenta	170, 0, 170	128, 0, 128	211, 56, 211	187, 0, 187	156, 0, 156	205, 0, 205	255, 0, 255	255, 0, 255
		0, 170,			0, 187,	0,	0,		0,

	Cyan	170	0, 128, 128	51, 187, 200	187	147, 147	205, 205	0, 255, 255	255, 255
	Gray	170, 170, 170	192, 192, 192	203, 204, 205	187, 187, 187	210, 210, 210	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	127, 127, 127		
	Red	255, 85, 85	255, 0, 0	252,57,31	255, 85, 85	255, 0, 0	255, 0, 0		
	Green	85, 255, 85	0, 255, 0	49, 231, 34	85, 255, 85	0, 252, 0	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, 0	255, 255, 224	225, 255, 224
	Blue	85, 85, 255	0, 0, 255	88, 51, 255	85, 85, 255	0, 0, 252	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	255, 0, 255		
	Cyan	85, 255, 255	0, 255, 255	20, 240, 240	85, 255, 255	0, 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	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,[17] 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.

Xterm,[14] KDE's Konsole,[18] as well as all libvte based terminals[19] (including GNOME Terminal) support ISO-8613-3 24-bit foreground and background color setting Quoting one of the text-files in its source-tree:[20]

```
ESC[ … 38;2;<r>;<g>;<b> … m Select RGB foreground color
ESC[ … 48;2;<r>;<g>;<b> … m Select RGB background color
```

The ITU's "T.416 Information technology - Open Document Architecture (ODA) and interchange format: Character content architectures". which was adopted as ISO/IEC International Standard 8613-6 gives more detail:

```
  ESC[ … 38:2:<r>:<g>:<b>:<unused>:<CS tolerance>:<Color-Space: 0="CIELUV"; 1="CIELAB">m Select RGB foreground color
  ESC[ … 48:2:<r>:<g>:<b>:<unused>:<CS tolerance>:<Color-Space: 0="CIELUV"; 1="CIELAB">m Select RGB background color
```

Note that this uses the otherwise reserved ':' character to separate the sub-options which may have been a source of confusion for real-world implementations. It also documents using '3' as the second parameter to specify colors using a Cyan-Magenta-Yellow scheme and '4' for a Cyan-Magenta-Yellow-Black one, the latter using the position marked as "unused" in the above examples for the Black component.

In 256-color mode (ESC[38;5;<*fgcode*>m and ESC[48;5;<*bgcode*>m), the color-codes are the following:

```
  0x00-0x07:  standard colors (as in ESC [ 30–37 m)
  0x08-0x0F:  high intensity colors (as in ESC [ 90–97 m)
  0x10-0xE7:  6 × 6 × 6 = 216 colors: 16 + 36 × r + 6 × g + b (0 ≤ r, g, b ≤ 5)
  0xE8-0xFF:  grayscale from black to white in 24 steps
```

256-color mode — foreground: ESC[38;5;#m   background: ESC[48;5;#m

Standard colors								High-intensity colors							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

216 colors

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231

Grayscale colors

232	**233**	**234**	**235**	**236**	**237**	**238**	**239**	**240**	**241**	**242**	**243**	**244**	**245**	**246**	**247**	**248**	**249**	**250**	**251**	**252**	**253**	**254**	**255**

Xterm allows also to set the default foreground and background colors using[14]

```
  ESC]10;<foreground>BEL
  ESC]11;<background>BEL
```

where `<foreground>` and `<background>` are X color specifications, and BEL is the ASCII BEL character (code 7). The closing bracket instead of an opening bracket reveals that it belongs to the operating system control commands.

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 reassigns 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 was sometimes used for ANSI bombs. 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 GNU *sed* command embellishes the output of the *make* command by displaying lines containing words starting with "WARN" in reverse video and words starting with "ERR" in bright yellow on a dark red background (letter case is ignored). The representations of the codes are highlighted.[21]

```
make 2>&1 | sed -e 's/.*\bWARN.*/\x1b[7m&\x1b[0m/i' -e 's/.*\bERR.*/\x1b[93;41 m&\x1b[0m/i'
```

The following Bash function flashes the terminal (by alternately sending reverse and normal video mode codes) until the user presses a key.[22]

```
flasher () { while true; do printf \\e[?5h; sleep 0.1; printf \\e[?5l; read -s -n1 -t1 && break; done; }
```

This can be used to alert a programmer when a lengthy command terminates, such as with `make ; flasher`.[23]

```
printf \\033c
```

This will reset the console, similar to the command `reset` on modern Linux systems; however it should work even on older Linux systems and on other (non-Linux) UNIX variants.

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 generate `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.

See also

- ANSI art
- Control character
- Advanced Video Attribute Terminal Assembler and Recreator (AVATAR)
- ISO/IEC JTC 1/SC 2

Notes

1. Oisin Grehan (4 February 2016). "Windows 10 TH2 (v1511) Console Host Enhancements". Retrieved 10 February 2016.
2. Historical version of ECMA-48 (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-48,%202nd%20Edition,%20August%201979.pdf)
3. Paul Williams (2006). "Digital's Video Terminals". VT100.net. Retrieved 17 August 2011.
4. Heathkit Company (1979). "Heathkit Catalog 1979". Heathkit Company. Retrieved 4 November 2011.
5. Withdrawn FIPS Listed by Number (http://www.nist.gov/itl/upload/Withdrawn-FIPS-by-Numerical-Order-Index2.pdf)
6. "Amiga Printer Command Definitions". Commodore. Retrieved 10 July 2013.
7. "Using C-Kermit" (https://books.google.ca/books?id=Z0ejBQAAQBAJ&pg=PA88), p. 88.
8. The screen display could be replaced by drawing the entire new screen's contents at the bottom, scrolling the previous screen up sufficiently to erase all the old text. The user would see the scrolling, and the hardware cursor would be left at the very bottom. Some early batch files achieved rudimentary "full screen" displays in this way.
9. Michael Mefford (7 February 1989). "ANSI.com: Download It Here". PC Magazine. Retrieved 10 August 2011.
10. Dan Kegel, Eric Auer (28 February 1999). "Nansi and NNansi – ANSI Drivers for MS-DOS". Dan Kegel's Web Hostel. Retrieved 10 August 2011.
11. Jason Hood (2005). "Process ANSI escape sequences for Windows console programs". Jason Hood's Home page. Retrieved 9 May 2013.
12. "colorama 0.2.5 :". *Python Package Index*. Retrieved 17 August 2013.
13. "Standard ECMA-48: Control Functions for Coded Character Sets" (Fifth ed.). Ecma International. June 1991.
14. "XTerm Control Sequences". *invisible-island.net*. 13 January 2014. Retrieved 13 April 2014.
15. 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.
16. 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.

17. "Patch #192 – 2004/7/12 – XFree86 4.4.99.9".
18. "color-spaces.pl (a copy of 256colors2.pl from xterm dated 11 July 1999)". KDE. 6 December 2006.
19. "libvte's bug report and patches". GNOME Bugzilla. 4 April 2014. Retrieved 5 June 2016.
20. "README.moreColors". KDE. 22 April 2010.
21. "Chapter 9. System tips". *debian.org*.
22. "VT100.net: Digital VT100 User Guide". Retrieved 19 January 2015.
23. "bash – How to get a notification when my commands are done – Ask Different". Retrieved 19 January 2015.

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 (https://web.archive.org/web/20060206022229/http://enterprise.aacc.cc.md.us/~rhs/ansi.html) at the Wayback Machine (archived 6 February 2006)
- 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=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds1%2Faixterm.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)
- ITU-T Rec. T.416 (03/93) Information technology – Open Document Architecture (ODA) and interchange format: Character content architectures (https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.416-199303-I!!PDF-E&type=items)