XTerm Control Sequences

Edward Moy

University of California, Berkeley

Revised by

Stephen Gildea

X Consortium (1994)

Thomas Dickey

XFree86 Project (1996-2006)
invisible-island.net (2006-2025)
updated for XTerm Patch #401 (2025/06/22)

Definitions

Many controls use parameters, shown in italics. If a control uses a single parameter, only one parameter name is listed. Some parameters (along with separating $\boxed{\ ;\ }$ characters) may be optional. Other characters in the control are required.

C A single (required) character.

P_s A single (usually optional) numeric parameter, composed of one or more digits.

P_m Any number of single numeric parameters, separated by $\boxed{\ ;\ }$ character(s). Individual values for the parameters are listed with P_s .

P_t A text parameter composed of printable characters.

Control Bytes, Characters, and Sequences

ECMA-48 (aka "ISO 6429") documents C1 (8-bit) and C0 (7-bit) codes. Those are respectively codes 128 to 159 and 0 to 31. ECMA-48 avoids referring to these codes as characters, because that term is associated with *graphic characters*. Instead, it uses "bytes" and "codes", with occasional lapses to "characters" where the meaning cannot be mistaken.

Controls (including the escape code 27) are processed once:

- This means that a C1 control can be mistaken for badly-formed UTF-8 when the terminal runs in UTF-8 mode because C1 controls are valid *continuation bytes* of a UTF-8 encoded (multibyte) value.
- It is not possible to use a C1 control obtained from decoding the UTF-8 text, because that would require reprocessing the data. Consequently there is no ambiguity in the way this document uses the term "character" to refer to bytes in a control sequence.

The order of processing is a necessary consequence of the way ECMA-48 is designed:

- Each byte sent to the terminal can be unambiguously determined to fall into one of a few categories (C0, C1 and graphic characters).
- ECMA-48 is *modal*; once it starts processing a control sequence, the terminal continues until the sequence is complete, or some byte is found which is not allowed in the sequence.
- Intermediate, parameter and final bytes may use the same codes as graphic characters, but they are processed as part of a control sequence and are not actually graphic characters.

- Eight-bit controls can have intermediate, etc., bytes in the range 160 to 255. Those can be treated as their counterparts in the range 32 to 127.
- Single-byte controls can be handled separately from multi-byte control sequences because ECMA-48's rules are unambiguous.

  As a special case, ECMA-48 (section 9) mentions that the control functions shift-in and shift-out are allowed to occur within a 7-bit multibyte control sequence because those cannot alter the meaning of the control sequence.
- Some controls (such as OSC) introduce a string mode, which is ended on a ST (string terminator).

  Section 9 of ECMA-48, like DEC STD 070, chapter 3, goes into detail to explain that when processing 8-bit controls, the eighth bit of each byte is ignored. This applies to the content of APC, DCS, OSC, and PM strings, as well as to the terminating bytes such as the two-byte string terminator. Quoting from the latter, 3.5.4.5 *GR Graphic Characters within Control Strings*:

  ```
  GR (8-bit) graphic characters in APC, OSC, and PM control strings will be
  treated as their 7-bit equivalent (the eighth bit will be ignored).

  GR (8-bit) graphic characters are permitted within Device Control
  Strings, and the graphic character's interpretation will be dependent on
  the internal control string format. When they occur in the introducer
  sequence to a Device Control String, the eighth bit will be ignored, and
  they will be treated as their 7-bit equivalent. (Note that this is the
  same way 8-bit graphic characters are handled within control sequences.)
  ```

  The reason for that is because ECMA-48 presents 7-bit controls as an alternative to 8-bit controls. It says this:

  ```
  The control functions defined in this Standard can be coded in a 7-bit
  code as well as in an 8-bit code; both forms of coded representation are
  equivalent and in accordance with Standard ECMA-35.
  ```

  and in turn, ECMA-35 9.1 says

  ```
  A 7-bit code shall have a structure which is based on a 7-bit code table
  arranged in separate areas as follows (see figure 7):
  ```

  In short, a standard-compliant implementation of ECMA-48 ignores the eighth bit of bytes in control strings other than the C1 controls. *XTerm* does this.

  ECMA-48 describes only correct behavior, telling what types of characters are expected at each stage of the control sequences. It says that the action taken in error recovery is implementation-dependent. *XTerm* decodes control sequences using a state machine. It handles errors in decoding i.e., unexpected characters, by resetting to the initial (ground) state. That is different from the treatment of unimplemented (but correctly formatted) features.

  If an application does not send the string terminator, that is also an error from the standpoint of a user. To accommodate users of those applications, *xterm* has resource settings which allow workarounds:

  - The Linux console's palette sequences do not use a string terminator. The **brokenLinuxOSC** resource setting tells *xterm* to ignore those particular sequences.
  - The terminal should accept single-byte controls within the string. But some applications omit a string terminator, like the Linux console. The **brokenStringTerm** resource setting tells *xterm* to exit string mode if it decodes a common control character such as carriage return before the string terminator.

C1 (8-Bit) Control Characters

The *xterm* program recognizes both 8-bit and 7-bit control characters. It generates 7-bit controls (by default) or 8-bit if S8C1T is enabled. The following pairs of 7-bit and 8-bit control characters are equivalent:

ESC D
: Index (IND is 0x84).

ESC E
: Next Line (NEL is 0x85).

ESC H
: Tab Set (HTS is 0x88).

ESC M
: Reverse Index (RI is 0x8d).

ESC N
: Single Shift Select of G2 Character Set (SS2 is 0x8e), VT220. This affects next character only.

ESC O
: Single Shift Select of G3 Character Set (SS3 is 0x8f), VT220. This affects next character only.

ESC P
: Device Control String (DCS is 0x90).

ESC V
: Start of Guarded Area (SPA is 0x96).

ESC W
: End of Guarded Area (EPA is 0x97).

ESC X
: Start of String (SOS is 0x98).

ESC Z
: Return Terminal ID (DECID is 0x9a). Obsolete form of CSI c (DA).

ESC [
: Control Sequence Introducer (CSI is 0x9b).

ESC \
: String Terminator (ST is 0x9c).

ESC]
: Operating System Command (OSC is 0x9d).

ESC ^
: Privacy Message (PM is 0x9e).

ESC _
: Application Program Command (APC is 0x9f).

These control characters are used in the vtXXX emulation.

VT100-related terminals

In this document, “VT100” refers not only to VT100/VT102, but also to the succession of upward-compatible terminals produced by DEC (Digital Equipment Corporation) from the mid-1970s for about twenty years. For brevity, the document refers to the related models:
“VT200” as VT220/VT240,
“VT300” as VT320/VT340,
“VT400” as VT420, and
“VT500” as VT510/VT520/VT525.

Most of these control sequences are standard VT102 control sequences, but there is support for later DEC VT terminals (i.e., VT220, VT320, VT420, VT510), as well as ECMA-48 and *aixterm* color controls. The only VT102 feature not supported is auto-repeat, since the only way X provides for this will affect all windows.

There are additional control sequences to provide *xterm*-dependent functions, such as the scrollbar or window size. Where the function is specified by DEC or ECMA-48, the mnemonic assigned to it is given in parentheses.

The escape codes to designate and invoke character sets are specified by ISO 2022 (see that document for a discussion of character sets).

Many of the features are optional; *xterm* can be configured and built without support for them.

VT100 Mode

Single-character functions

BEL — Bell (BEL is Ctrl-G).

BS — Backspace (BS is Ctrl-H).

CR — Carriage Return (CR is Ctrl-M).

ENQ — Return Terminal Status (ENQ is Ctrl-E). Default response is an empty string, but may be overridden by a resource **answerbackString**.

FF — Form Feed or New Page (NP). (FF is Ctrl-L). FF is treated the same as LF.

LF — Line Feed or New Line (NL). (LF is Ctrl-J).

SI — Switch to *Standard Character Set* (Ctrl-O is Shift In or LS0). This invokes the G0 character set (the default) as GL.

VT200 and up implement LS0.

SO — Switch to *Alternate Character Set* (Ctrl-N is Shift Out or LS1). This invokes the G1 character set as GL.

VT200 and up implement LS1.

SP — Space.

TAB — Horizontal Tab (HTS is Ctrl-I).

VT — Vertical Tab (VT is Ctrl-K). This is treated the same as LF.

Controls beginning with ESC

This excludes controls where ESC is part of a 7-bit equivalent to 8-bit C1 controls, ordered by the final character(s).

ESC SP F — 7-bit controls (S7C1T), VT220. This tells the terminal to send C1 control characters as 7-bit sequences, e.g., its responses to queries. DEC VT200 and up always accept 8-bit control sequences except when configured for VT100 mode.

ESC SP G — 8-bit controls (S8C1T), VT220. This tells the terminal to send C1 control characters as 8-bit sequences, e.g., its responses to queries. DEC VT200 and up always accept 8-bit control sequences except when configured for VT100 mode.

ESC SP L — Set ANSI conformance level 1, ECMA-43.

ESC SP M — Set ANSI conformance level 2, ECMA-43.

ESC SP N — Set ANSI conformance level 3, ECMA-43.

ESC # 3 — DEC double-height line, top half (DECDHL), VT100.

ESC # 4 — DEC double-height line, bottom half (DECDHL), VT100.

ESC # 5 DEC single-width line (DECSWL), VT100.

ESC # 6 DEC double-width line (DECDWL), VT100.

ESC # 8 DEC Screen Alignment Test (DECALN), VT100.

ESC % @ Select default character set. That is ISO 8859-1 (ISO 2022).

ESC % G Select UTF-8 character set, ISO 2022.

ESC (*C* Designate G0 Character Set, VT100, ISO 2022.

Final character *C* for designating 94-character sets. In this list,

- 0 , A and B were introduced in the VT100,
- most were introduced in the VT200 series,
- a few were introduced in the VT300 series, and
- a few more were introduced in the VT500 series.

The VT220 character sets, together with a few others (such as Portuguese) are activated by the National Replacement Character Set (NRCS) controls. The term “replacement” says that the character set is formed by replacing some of the characters in a set (termed the *Multinational Character Set*) with more useful ones for a given language. The ASCII and DEC Supplemental character sets make up the two halves of the Multinational Character set, initially mapped to GL and GR.

The valid final characters *C* for this control are:

C = A → United Kingdom (UK), VT100.
C = B → United States (USASCII), VT100.
C = C or 5 → Finnish, VT200.
C = H or 7 → Swedish, VT200.
C = K → German, VT200.
C = Q or 9 → French Canadian, VT200.
C = R or f → French, VT200.
C = Y → Italian, VT200.
C = Z → Spanish, VT200.
C = 4 → Dutch, VT200.
C = " > → Greek, VT500.
C = % 2 → Turkish, VT500.
C = % 6 → Portuguese, VT300.
C = % = → Hebrew, VT500.
C = = → Swiss, VT200.
C = ` , E or 6 → Norwegian/Danish, VT200.

The final character A is a special case, since the same final character is used by the VT300-control for the 96-character British Latin-1.

There are a few other 94-character sets:

C = 0 → DEC Special Character and Line Drawing Set, VT100.
C = < → DEC Supplemental, VT200.
C = < → User Preferred Selection Set, VT300.
C = > → DEC Technical, VT300.

These are documented as 94-character sets (like USASCII) without NRCS:

C = " 4 → DEC Hebrew, VT500.
C = " ? → DEC Greek, VT500.
C = % 0 → DEC Turkish, VT500.
C = % 5 → DEC Supplemental Graphics, VT300.
C = & 4 → DEC Cyrillic, VT500.
C = I → JIS-Katakana, VT382.
C = J → JIS-Roman, VT382.

The VT520 reference manual lists a few more, but no documentation has been found for the mappings:

C = % 3 → SCS NRCS, VT500.
C = & 5 → DEC Russian, VT500.

ESC) C Designate G1 Character Set, ISO 2022, VT100.
The same character sets apply as for ESC (C.

ESC * C Designate G2 Character Set, ISO 2022, VT220.
The same character sets apply as for ESC (C.

ESC + C Designate G3 Character Set, ISO 2022, VT220.
The same character sets apply as for ESC (C.

ESC – C Designate G1 Character Set, VT300.
These controls apply only to 96-character sets. Unlike the 94-character sets, these can have different values than ASCII space and DEL for the mapping of 0x20 and 0x7f. The valid final characters C for this control are:

C = A → ISO Latin-1 Supplemental, VT300.
C = B → ISO Latin-2 Supplemental, VT500.
C = F → ISO Greek Supplemental, VT500.
C = H → ISO Hebrew Supplemental, VT500.
C = L → ISO Latin-Cyrillic, VT500.
C = M → ISO Latin-5 Supplemental, VT500.

ESC . C Designate G2 Character Set, VT300.
The same character sets apply as for ESC – C.

ESC / C Designate G3 Character Set, VT300.
The same character sets apply as for ESC – C.

ESC 6 Back Index (DECBI), VT420 and up.

ESC 7 Save Cursor (DECSC), VT100.

ESC 8 Restore Cursor (DECRC), VT100.

ESC 9 Forward Index (DECFI), VT420 and up.

ESC = Application Keypad (DECKPAM).

ESC > Normal Keypad (DECKPNM), VT100.

ESC F Cursor to lower left corner of screen. This is enabled by the **hpLowerleftBugCompat** resource.

ESC c Full Reset (RIS), VT100.

ESC l Memory Lock (per HP terminals). Locks memory above the cursor.

ESC m — Memory Unlock (per HP terminals).

ESC n — Invoke the G2 Character Set as GL (LS2).

ESC o — Invoke the G3 Character Set as GL (LS3).

ESC | — Invoke the G3 Character Set as GR (LS3R).

ESC } — Invoke the G2 Character Set as GR (LS2R).

ESC ~ — Invoke the G1 Character Set as GR (LS1R), VT100.

Application Program-Command functions

APC P_t ST — None. *xterm* implements no APC functions; P_t is ignored. P_t need not be printable characters.

Device-Control functions

DCS P_s ; P_s | P_t ST

User-Defined Keys (DECUDK), VT220 and up.

The first parameter:

P_s = 0 → Clear all UDK definitions before starting (default).

P_s = 1 → Erase Below (default).

The second parameter:

P_s = 0 ← Lock the keys (default).

P_s = 1 ← Do not lock.

The third parameter is a “;”-separated list of strings denoting the key-code separated by a “/” from the hex-encoded key value. The key codes correspond to the DEC function-key codes (e.g., F6=17).

DCS P_s ! u P_t ST

Assigning User-Preferred Supplemental Sets (DECAUPSS), VT320, VT510. *XTerm* ignores this in UTF-8 mode, and uses the **preferLatin1** resource to choose the default setting.

VT320 provides these:

DCS 0 ! u % 5 ST → DEC Supplemental Graphic

DCS 1 ! u A ST → ISO Latin-1 supplemental

VT510 adds these:

DCS 0 ! u " ? ST → DEC Greek

DCS 0 ! u " 4 ST → DEC Hebrew

DCS 0 ! u % 0 ST → DEC Turkish

DCS 0 ! u & 4 ST → DEC Cyrillic

DCS 1 ! u B ST → ISO Latin-2 Supplemental

DCS 1 ! u F ST → ISO Greek Supplemental

DCS 1 ! u H ST → ISO Hebrew Supplemental

DCS 1 ! u M ST → ISO Latin-5 Supplemental

DCS 1 ! u L ST → ISO Latin-Cyrillic

VT520 accepts a few others (undocumented); xterm adds these:

DCS 0 ! u B ST → United States (USASCII).
DCS 0 ! u 0 ST → DEC Special Character and Line Drawing Set.
DCS 0 ! u > ST → DEC Technical.

DCS $ q P_t ST

Request Status String (DECRQSS), VT420 and up.

The string following the “q” is one of the following:

m → SGR
" p → DECSCL
SP q → DECSCUSR
" q → DECSCA
r → DECSTBM
s → DECSLRM
t → DECSLPP
$ | → DECSCPP
$ } → DECSASD
$ ~ → DECSSDT
) { → DECSTGLT (VT525 only)
* x → DECSACE
* | → DECSNLS
, | → DECAC (VT525 only)
, } → DECATC (VT525 only)
> P_m f → XTQFMTKEYS (xterm)
> P_m m → XTQMODKEYS (xterm)
> P_m t → XTSMTITLE (xterm)

xterm responds with DCS 1 $ r P_t ST for valid requests, replacing the P_t with the corresponding CSI string, or DCS 0 $ r ST for invalid requests.

DCS P_s $ t P_t ST

Restore presentation status (DECRSPS), VT320 and up. The control can be converted from a response from DECCIR or DECTABSR by changing the first “u” to a “t”

P_s = 1 → DECCIR
P_s = 2 → DECTABSR

DCS + Q P_t ST

Request resource values (XTGETXRES), *xterm*. The string following the “Q” is a list of names encoded in hexadecimal (2 digits per character) separated by ; which correspond to *xterm* resource names.

xterm responds with
DCS 1 + R P_t ST for valid requests, adding to P_t an =, and the value of the corresponding *xterm* resource, or
DCS 0 + R P_t ST for invalid requests.
The strings are encoded in hexadecimal (2 digits per character).

Only boolean, numeric and string resources for the VT100 widget are supported by this query. *XTerm* evaluates resources at startup time. Several of *xterm*'s state variables use resources to determine their initial value. Because the resource variable may not reflect the current state, *xterm* provides control sequences for querying the state directly:

- XTQALLOWED
- XTQFMTKEYS
- XTQMODKEYS

DCS + p P_t ST
Set Termcap/Terminfo Data (XTSETTCAP), *xterm*. The string following the "p" is encoded in hexadecimal. After decoding it, *xterm* will use the name to retrieve data from the terminal database. If successful, that overrides the **termName** resource when handling the "tcap" keyboard configuration's function- and special-keys, as well as by the Request Termcap/Terminfo String control.

DCS + q P_t ST
Request Termcap/Terminfo String (XTGETTCAP), *xterm*. The string following the "q" is a list of names encoded in hexadecimal (2 digits per character) separated by ; which correspond to termcap or terminfo capability names for special keyboard keys. A terminal description will include other capabilities, e.g., for cursor movement, which are intentionally not part of this interface.
A few more terminal capabilities are recognized, which are not names of special keys:

- *Co* for termcap colors (or *colors* for terminfo colors), and
- *TN* for termcap name (or *name* for terminfo name).
- *RGB* for the ncurses direct-color extension.
  Only a terminfo name is provided, since termcap applications cannot use this information.

These capabilities fall into two categories:

- Terminal capabilities which may be dynamically adjusted in *xterm* so they do not necessarily match a terminal description.
- The name of the terminal description, which an application can use to obtain the static set of capabilities.

xterm responds with
DCS 1 + r P_t ST for valid requests, adding to P_t an =, and the value of the corresponding string that *xterm* would send, or
DCS 0 + r ST for invalid requests.
The strings are encoded in hexadecimal (2 digits per character). If more than one name is given, *xterm* replies with each name/value pair in the same response. An invalid name (one not found in *xterm*'s tables) ends processing of the list of names.

Functions using CSI, ordered by the final character(s)

CSI P_s @ Insert P_s (Blank) Character(s) (default = 1) (ICH).
CSI P_s SP @ Shift left P_s columns(s) (default = 1) (SL), ECMA-48.
CSI P_s A Cursor Up P_s Times (default = 1) (CUU).
CSI P_s SP A Shift right P_s columns(s) (default = 1) (SR), ECMA-48.
CSI P_s B Cursor Down P_s Times (default = 1) (CUD).
CSI P_s C Cursor Forward P_s Times (default = 1) (CUF).

CSI P_s D — Cursor Backward P_s Times (default = 1) (CUB).

CSI P_s E — Cursor Next Line P_s Times (default = 1) (CNL).

CSI P_s F — Cursor Preceding Line P_s Times (default = 1) (CPL).

CSI P_s G — Cursor Character Absolute [column] (default = [row,1]) (CHA).

CSI P_s ; P_s H

Cursor Position [row;column] (default = [1,1]) (CUP).

CSI P_s I — Cursor Forward Tabulation P_s tab stops (default = 1) (CHT).

CSI P_s J — Erase in Display (ED), VT100.

- P_s = 0 → Erase Below (default).
- P_s = 1 → Erase Above.
- P_s = 2 → Erase All.
- P_s = 3 → Erase Saved Lines, *xterm*.

CSI ? P_s J — Erase in Display (DECSED), VT220.

- P_s = 0 → Selective Erase Below (default).
- P_s = 1 → Selective Erase Above.
- P_s = 2 → Selective Erase All.
- P_s = 3 → Selective Erase Saved Lines, *xterm*.

CSI P_s K — Erase in Line (EL), VT100.

- P_s = 0 → Erase to Right (default).
- P_s = 1 → Erase to Left.
- P_s = 2 → Erase All.

CSI ? P_s K — Erase in Line (DECSEL), VT220.

- P_s = 0 → Selective Erase to Right (default).
- P_s = 1 → Selective Erase to Left.
- P_s = 2 → Selective Erase All.

CSI P_s L — Insert P_s Line(s) (default = 1) (IL).

CSI P_s M — Delete P_s Line(s) (default = 1) (DL).

CSI P_s P — Delete P_s Character(s) (default = 1) (DCH).

CSI # P

CSI P_m # P — Push current dynamic- and ANSI-palette colors onto stack (XTPUSHCOLORS), *xterm*. Parameters (integers in the range 1 through 10, since the default 0 will push) may be used to store the palette into the stack without pushing.

CSI # Q

CSI P_m # Q — Pop stack to set dynamic- and ANSI-palette colors (XTPOPCOLORS), *xterm*. Parameters (integers in the range 1 through 10, since the default 0 will pop) may be used to restore the palette from the stack without popping.

CSI # R — Report the current entry on the palette stack, and the number of palettes stored on the stack, using the same form as XTPOPCOLOR (default = 0) (XTREPORTCOLORS), *xterm*.

CSI P_s S — Scroll up P_s lines (default = 1) (SU), VT420, ECMA-48.

CSI # S — Report position on title-stack (XTTITLEPOS), *xterm*.

Response is the same format, with parameters:

CSI P_n ; P_m # S

where

P_n is the current index into the title stack

P_m is the maximum index for the title stack

CSI ? P_i ; P_a ; P_v S

Set or request graphics attribute (XTSMGRAPHICS), *xterm*. If configured to support either **Sixel Graphics** or **ReGIS Graphics**, *xterm* accepts a three-parameter control sequence, where P_i, P_a and P_v are the *item*, *action* and *value*:

P_i = 1 → item is number of color registers.
P_i = 2 → item is Sixel graphics geometry (in pixels).
P_i = 3 → item is ReGIS graphics geometry (in pixels).

P_a = 1 → read attribute.
P_a = 2 → reset to default.
P_a = 3 → set to value in P_v.
P_a = 4 → read the maximum allowed value.

P_v is ignored by *xterm* except when setting (P_a == 3).
P_v = *n* ← A single integer is used for color registers.
P_v = *width* ; *height* ← Two integers for graphics geometry.

xterm replies with a control sequence of the same form:

CSI ? P_i ; P_s ; P_v S

where P_s is the status:
P_s = 0 ← success.
P_s = 1 ← error in P_i.
P_s = 2 ← error in P_a.
P_s = 3 ← failure.

On success, P_v represents the value read or set.

Notes:

- The current implementation allows reading the graphics sizes, but disallows modifying those sizes because that is done once, using resource-values.
- Graphics geometry is not necessarily the same as “window size” (see the **XTWINOPS** window manipulation extensions). *XTerm* limits the maximum graphics geometry according to the **maxGraphicSize** resource.
  The **maxGraphicSize** resource can be either an explicit *height*x*width* (default: 1000x1000 as of version 328) or the word “auto” (telling *XTerm* to use limits the **decGraphicsID** or **decTerminalID** resource to determine the limits).
- *XTerm* uses the minimum of the window size and the graphic size to obtain the maximum geometry.

- While resizing a window will always change the current graphics geometry, the reverse is not true. Setting graphics geometry does not affect the window size.
- If *xterm* is able to support graphics (compile-time), but is not configured (runtime) for graphics, these responses will indicate a failure. Other implementations which do not use the maximum graphics dimensions but are configured for graphics should report zeroes for the maximum geometry rather than a failure.

CSI P_s T
Scroll down P_s lines (default = 1) (SD), VT420.

CSI P_s ; P_s ; P_s ; P_s ; P_s T
Initiate highlight mouse tracking (XTHIMOUSE), xterm. Parameters are [func;startx;starty;firstrow;lastrow]. See the section **Mouse Tracking**.

CSI > P_m T
Reset title mode features to default value (XTRMTITLE), *xterm*. Normally, “reset” disables the feature. It is possible to disable the ability to reset features by compiling a different default for the title modes into *xterm*.

If no parameters are given, all title mode features are reset to the initial (compiled-in) default.

P_s = 0 → Do not set window/icon labels using hexadecimal.
P_s = 1 → Do not query window/icon labels using hexadecimal.
P_s = 2 → Do not set window/icon labels using UTF-8.
P_s = 3 → Do not query window/icon labels using UTF-8.

(See discussion of **Title Modes**).

CSI ? 5 W
Reset tab stops to start with column 9, every 8 columns (DECST8C), VT510.

CSI P_s X
Erase P_s Character(s) (default = 1) (ECH).

CSI P_s Z
Cursor Backward Tabulation P_s tab stops (default = 1) (CBT).

CSI P_s ^
Scroll down P_s lines (default = 1) (SD), ECMA-48.
This was a publication error in the original ECMA-48 5th edition (1991) corrected in 2003.

CSI P_s `
Character Position Absolute [column] (default = [row,1]) (HPA).

CSI P_s a
Character Position Relative [columns] (default = [row,col+1]) (HPR).

CSI P_s b
Repeat the preceding graphic character P_s times (REP).

CSI P_s c
Send Device Attributes (Primary DA).

P_s = 0 or omitted → request attributes from terminal. The response depends on the **decTerminalID** resource setting.

→ CSI ? 1 ; 2 c (“VT100 with Advanced Video Option”)
→ CSI ? 1 ; 0 c (“VT101 with No Options”)
→ CSI ? 4 ; 6 c (“VT132 with Advanced Video and Graphics”)
→ CSI ? 6 c (“VT102”)
→ CSI ? 7 c (“VT131”)
→ CSI ? 1 2 ; P_s c (“VT125”)
→ CSI ? 6 2 ; P_s c (“VT220”)
→ CSI ? 6 3 ; P_s c (“VT320”)
→ CSI ? 6 4 ; P_s c (“VT420”)
→ CSI ? 6 5 ; P_s c (“VT510” to (“VT525”)

The VT100-style response parameters do not mean anything by themselves. VT220 (and higher) parameters do, telling the host what features the terminal supports:

P_s = 1 → 132-columns.
P_s = 2 → Printer.
P_s = 3 → ReGIS graphics.
P_s = 4 → Sixel graphics.
P_s = 6 → Selective erase.
P_s = 8 → User-defined keys.
P_s = 9 → National Replacement Character sets.
P_s = 1 5 → Technical characters.
P_s = 1 6 → Locator port.
P_s = 1 7 → Terminal state interrogation.
P_s = 1 8 → User windows.
P_s = 2 1 → Horizontal scrolling.
P_s = 2 2 → ANSI color, e.g., VT525.
P_s = 2 8 → Rectangular editing.
P_s = 2 9 → ANSI text locator (i.e., DEC Locator mode).

XTerm supports part of the *User windows* feature, providing a single page (which corresponds to its visible window). Rather than resizing the font to change the number of lines/columns in a fixed-size display, *xterm* uses the window extension controls (DECSNLS, DECSCPP, DECSLPP) to adjust its visible window's size. The "cursor coupling" controls (DECHCCM, DECPCCM, DECVCCM) are ignored.

CSI = P_s c Send Device Attributes (Tertiary DA).

P_s = 0 → report Terminal Unit ID (default), VT400. XTerm uses zeros for the site code and serial number in its DECRPTUI response.

CSI > P_s c Send Device Attributes (Secondary DA).

P_s = 0 or omitted → request the terminal's identification code. The response depends on the **decTerminalID** resource setting. It should apply only to VT220 and up, but *xterm* extends this to VT100.

→ CSI > P_p ; P_v ; P_c c

where P_p denotes the terminal type

P_p = 0 → "VT100".
P_p = 1 → "VT220".
P_p = 2 → "VT240" or "VT241".
P_p = 1 8 → "VT330".
P_p = 1 9 → "VT340".
P_p = 2 4 → "VT320".
P_p = 3 2 → "VT382".
P_p = 4 1 → "VT420".
P_p = 6 1 → "VT510".
P_p = 6 4 → "VT520".

P_p = 6 5 → "VT525".

and P_v is the firmware version (for *xterm*, this was originally the XFree86 patch number, starting with 95). In a DEC terminal, P_c indicates the ROM cartridge registration number and is always zero.

CSI P_s d Line Position Absolute [row] (default = [1,column]) (VPA).

CSI P_s e Line Position Relative [rows] (default = [row+1,column]) (VPR).

CSI P_s ; P_s f

Horizontal and Vertical Position [row;column] (default = [1,1]) (HVP).

CSI > P_p ; P_v f

CSI > P_p f Set/reset key format options (XTFMTKEYS), xterm. Set or reset resource-values used by *xterm* to decide how to format escape sequences holding information about the modifiers pressed with a given key.

The first parameter P_p identifies the resource to set/reset. The second parameter P_v is the value to assign to the resource.

If the second parameter is omitted, the resource is reset to its initial value. Value 5 is reserved for input via the **string** action.

P_p = 0 → **formatKeyboard**.
P_p = 1 → **formatCursorKeys**.
P_p = 2 → **formatFunctionKeys**.
P_p = 3 → **formatKeypadKeys**.
P_p = 4 → **formatOtherKeys**.
P_p = 6 → **formatModifierKeys**.
P_p = 7 → **formatSpecialKeys**.

If no parameters are given, all resources are reset to their initial values. See *Alt and Meta Keys* for more details on the **formatOtherKeys** feature.

CSI P_s g Tab Clear (TBC). ECMA-48 defines additional codes, but the VT100 user manual notes that it ignores other codes. DEC's later terminals (and *xterm*) do the same, for compatibility.

P_s = 0 → Clear Current Column (default).
P_s = 3 → Clear All.

CSI ? P_p g Query key modifier options (XTQFMTKEYS), *xterm*.

The parameter P_p identifies the resource to query.

P_p = 0 → **formatKeyboard**.
P_p = 1 → **formatCursorKeys**.
P_p = 2 → **formatFunctionKeys**.
P_p = 3 → **formatKeypadKeys**.
P_p = 4 → **formatOtherKeys**.
P_p = 6 → **formatModifierKeys**.
P_p = 7 → **formatSpecialKeys**.

XTerm's response can be used to restore this state, because it is formatted as an XTFMTKEYS control, i.e.,

CSI > P_p f

where
P_p = 0 → **formatKeyboard**.
P_p = 1 → **formatCursorKeys**.
P_p = 2 → **formatFunctionKeys**.
P_p = 3 → **formatKeypadKeys**.
P_p = 4 → **formatOtherKeys**.
P_p = 6 → **formatModifierKeys**.
P_p = 7 → **formatSpecialKeys**.

CSI P_m h Set Mode (SM).
P_s = 2 → Keyboard Action Mode (KAM).
P_s = 4 → Insert Mode (IRM).
P_s = 1 2 → Send/receive (SRM).
P_s = 2 0 → Automatic Newline (LNM).

CSI ? P_m h DEC Private Mode Set (DECSET).
P_s = 1 → Application Cursor Keys (DECCKM), VT100.
P_s = 2 → Designate USASCII for character sets G0-G3 (DECANM), VT100, and set VT100 mode.
P_s = 3 → 132 Column Mode (DECCOLM), VT100.
P_s = 4 → Smooth (Slow) Scroll (DECSCLM), VT100.
P_s = 5 → Reverse Video (DECSCNM), VT100.
P_s = 6 → Origin Mode (DECOM), VT100.
P_s = 7 → Auto-Wrap Mode (DECAWM), VT100.
P_s = 8 → Auto-Repeat Keys (DECARM), VT100.
P_s = 9 → Send Mouse X & Y on button press. See the section **Mouse Tracking**. This is the X10 *xterm* mouse protocol.
P_s = 1 0 → Show toolbar (rxvt).
P_s = 1 2 → Start blinking cursor (AT&T 610).
P_s = 1 3 → Start blinking cursor (set only via resource or menu).
P_s = 1 4 → Enable XOR of blinking cursor control sequence and menu.
P_s = 1 8 → Print Form Feed (DECPFF), VT220.
P_s = 1 9 → Set print extent to full screen (DECPEX), VT220.
P_s = 2 5 → Show cursor (DECTCEM), VT220.
P_s = 3 0 → Show scrollbar (rxvt).
P_s = 3 5 → Enable font-shifting functions (rxvt).
P_s = 3 8 → Enter Tektronix mode (DECTEK), VT240, *xterm*.
P_s = 4 0 → Allow 80 ↔ 132 mode, *xterm*.
P_s = 4 1 → **more**(1) fix (see **curses** resource).
P_s = 4 2 → Enable National Replacement Character sets (DECNRCM), VT220.
P_s = 4 3 → Enable Graphic Expanded Print Mode (DECGEPM), VT340.
P_s = 4 4 → Turn on margin bell, *xterm*.
P_s = 4 4 → Enable Graphic Print Color Mode (DECGPCM), VT340.

P_s = 4 5 → Reverse-wraparound mode (XTREVWRAP), *xterm*.

P_s = 4 5 → Enable Graphic Print Color Syntax (DECGPCS), VT340.

P_s = 4 6 → Start logging (XTLOGGING), *xterm*. This is normally disabled by a compile-time option.

P_s = 4 6 → Graphic Print Background Mode, VT340.

P_s = 4 7 → Use *Alternate Screen Buffer*, *xterm*. This may be disabled by the **titeInhibit** resource.

P_s = 4 7 → Enable Graphic Rotated Print Mode (DECGRPM), VT340.

P_s = 6 6 → Application keypad mode (DECNKM), VT320.

P_s = 6 7 → Backarrow key sends backspace (DECBKM), VT340, VT420. This sets the **backarrowKey** resource to "true".

P_s = 6 9 → Enable left and right margin mode (DECLRMM), VT420 and up.

P_s = 8 0 → Enable *Sixel Display Mode* (DECSDM), VT330, VT340, VT382.

P_s = 9 5 → Do not clear screen when DECCOLM is set/reset (DECNCSM), VT510 and up.

P_s = 1 0 0 0 → Send Mouse X & Y on button press and release. See the section **Mouse Tracking**. This is the X11 *xterm* mouse protocol.

P_s = 1 0 0 1 → Use Hilite Mouse Tracking, *xterm*.

P_s = 1 0 0 2 → Use Cell Motion Mouse Tracking, *xterm*. See the section **Button-event tracking**.

P_s = 1 0 0 3 → Use All Motion Mouse Tracking, *xterm*. See the section **Any-event tracking**.

P_s = 1 0 0 4 → Send **FocusIn/FocusOut** events, *xterm*.

P_s = 1 0 0 5 → Enable UTF-8 Mouse Mode, *xterm*.

P_s = 1 0 0 6 → Enable SGR Mouse Mode, *xterm*.

P_s = 1 0 0 7 → Enable *Alternate Scroll Mode*, *xterm*. This corresponds to the **alternateScroll** resource.

P_s = 1 0 1 0 → Scroll to bottom on tty output (rxvt). This sets the **scrollTtyOutput** resource to "true".

P_s = 1 0 1 1 → Scroll to bottom on key press (rxvt). This sets the **scrollKey** resource to "true".

P_s = 1 0 1 4 → Enable **fastScroll** resource, *xterm*.

P_s = 1 0 1 5 → Enable urxvt Mouse Mode.

P_s = 1 0 1 6 → Enable SGR Mouse PixelMode, *xterm*.

P_s = 1 0 3 4 → Interpret "meta" key, *xterm*. This sets the eighth bit of keyboard input (and enables the **eightBitInput** resource).

P_s = 1 0 3 5 → Enable special modifiers for Alt and NumLock keys, *xterm*. This enables the **numLock** resource.

P_s = 1 0 3 6 → Send ESC when Meta modifies a key, *xterm*. This enables the **metaSendsEscape** resource.

P_s = 1 0 3 7 → Send DEL from the editing-keypad Delete key, *xterm*.

P_s = 1 0 3 9 → Send ESC when Alt modifies a key, *xterm*. This enables the

altSendsEscape resource, *xterm*.

P_s = 1 0 4 0 → Keep selection even if not highlighted, *xterm*. This enables the **keepSelection** resource.

P_s = 1 0 4 1 → Use the CLIPBOARD selection, *xterm*. This enables the **selectToClipboard** resource.

P_s = 1 0 4 2 → Enable Urgency window manager hint when Control-G is received, *xterm*. This enables the **bellIsUrgent** resource.

P_s = 1 0 4 3 → Enable raising of the window when Control-G is received, *xterm*. This enables the **popOnBell** resource.

P_s = 1 0 4 4 → Reuse the most recent data copied to CLIPBOARD, *xterm*. This enables the **keepClipboard** resource.

P_s = 1 0 4 5 → Extended Reverse-wraparound mode (XTREVWRAP2), *xterm*.

P_s = 1 0 4 6 → Enable switching to/from *Alternate Screen Buffer*, *xterm*. This works for terminfo-based systems, updating the **titeInhibit** resource.

P_s = 1 0 4 7 → Use *Alternate Screen Buffer*, *xterm*. This may be disabled by the **titeInhibit** resource.

P_s = 1 0 4 8 → Save cursor as in DECSC, *xterm*. This may be disabled by the **titeInhibit** resource.

P_s = 1 0 4 9 → Save cursor as in DECSC, *xterm*. After saving the cursor, switch to the *Alternate Screen Buffer*, clearing it first. This may be disabled by the **titeInhibit** resource. This control combines the effects of the 1 0 4 7 and 1 0 4 8 modes. Use this with terminfo-based applications rather than the 4 7 mode.

P_s = 1 0 5 0 → Set terminfo/termcap function-key mode, *xterm*.

P_s = 1 0 5 1 → Set Sun function-key mode, *xterm*.

P_s = 1 0 5 2 → Set HP function-key mode, *xterm*.

P_s = 1 0 5 3 → Set SCO function-key mode, *xterm*.

P_s = 1 0 6 0 → Set legacy keyboard emulation, i.e, X11R6, *xterm*.

P_s = 1 0 6 1 → Set VT220 keyboard emulation, *xterm*.

P_s = 2 0 0 1 → Enable readline mouse button-1, *xterm*.

P_s = 2 0 0 2 → Enable readline mouse button-2, *xterm*.

P_s = 2 0 0 3 → Enable readline mouse button-3, *xterm*.

P_s = 2 0 0 4 → Set bracketed paste mode, *xterm*.

P_s = 2 0 0 5 → Enable readline character-quoting, *xterm*.

P_s = 2 0 0 6 → Enable readline newline pasting, *xterm*.

CSI P_s i Media Copy (MC).

P_s = 0 → Print screen (default).

P_s = 4 → Turn off printer controller mode.

P_s = 5 → Turn on printer controller mode.

P_s = 1 0 → HTML screen dump, *xterm*.

P_s = 1 1 → SVG screen dump, *xterm*.

CSI ? P_s i Media Copy (MC), DEC-specific.
P_s = 1 → Print line containing cursor.
P_s = 4 → Turn off autoprint mode.
P_s = 5 → Turn on autoprint mode.
P_s = 1 0 → Print composed display, ignores DECPEX.
P_s = 1 1 → Print all pages.

CSI P_m l Reset Mode (RM).
P_s = 2 → Keyboard Action Mode (KAM).
P_s = 4 → Replace Mode (IRM).
P_s = 1 2 → Send/receive (SRM).
P_s = 2 0 → Normal Linefeed (LNM).

CSI ? P_m l DEC Private Mode Reset (DECRST).
P_s = 1 → Normal Cursor Keys (DECCKM), VT100.
P_s = 2 → Designate VT52 mode (DECANM), VT100.
P_s = 3 → 80 Column Mode (DECCOLM), VT100.
P_s = 4 → Jump (Fast) Scroll (DECSCLM), VT100.
P_s = 5 → Normal Video (DECSCNM), VT100.
P_s = 6 → Normal Cursor Mode (DECOM), VT100.
P_s = 7 → No Auto-Wrap Mode (DECAWM), VT100.
P_s = 8 → No Auto-Repeat Keys (DECARM), VT100.
P_s = 9 → Don’t send Mouse X & Y on button press, *xterm*.
P_s = 1 0 → Hide toolbar (rxvt).
P_s = 1 2 → Stop blinking cursor (AT&T 610).
P_s = 1 3 → Disable blinking cursor (reset only via resource or menu).
P_s = 1 4 → Disable XOR of blinking cursor control sequence and menu.
P_s = 1 8 → Don’t Print Form Feed (DECPFF), VT220.
P_s = 1 9 → Limit print to scrolling region (DECPEX), VT220.
P_s = 2 5 → Hide cursor (DECTCEM), VT220.
P_s = 3 0 → Don’t show scrollbar (rxvt).
P_s = 3 5 → Disable font-shifting functions (rxvt).
P_s = 4 0 → Disallow 80 ↔ 132 mode, *xterm*.
P_s = 4 1 → No **more**(1) fix (see **curses** resource).
P_s = 4 2 → Disable National Replacement Character sets (DECNRCM), VT220.
P_s = 4 3 → Disable Graphic Expanded Print Mode (DECGEPM), VT340.
P_s = 4 4 → Turn off margin bell, *xterm*.
P_s = 4 4 → Disable Graphic Print Color Mode (DECGPCM), VT340.
P_s = 4 5 → No Reverse-wraparound mode (XTREVWRAP), *xterm*.
P_s = 4 5 → Disable Graphic Print Color Syntax (DECGPCS), VT340.
P_s = 4 6 → Stop logging (XTLOGGING), *xterm*. This is normally disabled by a compile-time option.
P_s = 4 7 → Use *Normal Screen Buffer*, *xterm*.

P_s = 4 7 → Disable Graphic Rotated Print Mode (DECGRPM), VT340.

P_s = 6 6 → Numeric keypad mode (DECNKM), VT320.

P_s = 6 7 → Backarrow key sends delete (DECBKM), VT340, VT420. This sets the **backarrowKey** resource to “false”.

P_s = 6 9 → Disable left and right margin mode (DECLRMM), VT420 and up.

P_s = 8 0 → Disable *Sixel Display Mode* (DECSDM), VT330, VT340, VT382. Turns on “Sixel Scrolling”. See the section **Sixel Graphics** and mode 8 4 5 2 .

P_s = 9 5 → Clear screen when DECCOLM is set/reset (DECNCSM), VT510 and up.

P_s = 1 0 0 0 → Don’t send Mouse X & Y on button press and release. See the section **Mouse Tracking**.

P_s = 1 0 0 1 → Don’t use Hilite Mouse Tracking, *xterm*.

P_s = 1 0 0 2 → Don’t use Cell Motion Mouse Tracking, *xterm*. See the section **Button-event tracking**.

P_s = 1 0 0 3 → Don’t use All Motion Mouse Tracking, *xterm*. See the section **Any-event tracking**.

P_s = 1 0 0 4 → Don’t send **FocusIn/FocusOut** events, *xterm*.

P_s = 1 0 0 5 → Disable UTF-8 Mouse Mode, *xterm*.

P_s = 1 0 0 6 → Disable SGR Mouse Mode, *xterm*.

P_s = 1 0 0 7 → Disable *Alternate Scroll Mode*, *xterm*. This corresponds to the **alternateScroll** resource.

P_s = 1 0 1 0 → Don’t scroll to bottom on tty output (rxvt). This sets the **scrollTtyOutput** resource to “false”.

P_s = 1 0 1 1 → Don’t scroll to bottom on key press (rxvt). This sets the **scrollKey** resource to “false”.

P_s = 1 0 1 4 → Disable **fastScroll** resource, *xterm*.

P_s = 1 0 1 5 → Disable urxvt Mouse Mode.

P_s = 1 0 1 6 → Disable SGR Mouse Pixel-Mode, *xterm*.

P_s = 1 0 3 4 → Don’t interpret “meta” key, *xterm*. This disables the **eightBitInput** resource.

P_s = 1 0 3 5 → Disable special modifiers for Alt and NumLock keys, *xterm*. This disables the **numLock** resource.

P_s = 1 0 3 6 → Don’t send ESC when Meta modifies a key, *xterm*. This disables the **metaSendsEscape** resource.

P_s = 1 0 3 7 → Send VT220 Remove from the editing-keypad *Delete* key, *xterm*.

P_s = 1 0 3 9 → Don’t send ESC when Alt modifies a key, *xterm*. This disables the **altSendsEscape** resource.

P_s = 1 0 4 0 → Do not keep selection when not highlighted, *xterm*. This disables the **keepSelection** resource.

P_s = 1 0 4 1 → Use the PRIMARY selection, *xterm*. This disables the **selectToClipboard** resource.

P_s = 1 0 4 2 → Disable Urgency window manager hint when Control-G is received, *xterm*. This disables the **bellIsUrgent** resource.

P_s = 1 0 4 3 → Disable raising of the window when Control-G is received, *xterm*. This disables the **popOnBell** resource.

P_s = 1 0 4 5 → No Extended Reverse-wraparound mode (XTREVWRAP2), *xterm*.

P_s = 1 0 4 6 → Disable switching to/from *Alternate Screen Buffer*, *xterm*. This works for terminfo-based systems, updating the **titeInhibit** resource. If currently using the *Alternate Screen Buffer*, *xterm* switches to the Normal Screen Buffer.

P_s = 1 0 4 7 → Use Normal Screen Buffer, *xterm*. Clear the screen first if in the *Alternate Screen Buffer*. This may be disabled by the **titeInhibit** resource.

P_s = 1 0 4 8 → Restore cursor as in DECRC, *xterm*. This may be disabled by the **titeInhibit** resource.

P_s = 1 0 4 9 → Use Normal Screen Buffer and restore cursor as in DECRC, *xterm*. This may be disabled by the **titeInhibit** resource. This combines the effects of the 1 0 4 7 and 1 0 4 8 modes. Use this with terminfo-based applications rather than the 4 7 mode.

P_s = 1 0 5 0 → Reset terminfo/termcap function-key mode, *xterm*.

P_s = 1 0 5 1 → Reset Sun function-key mode, *xterm*.

P_s = 1 0 5 2 → Reset HP function-key mode, *xterm*.

P_s = 1 0 5 3 → Reset SCO function-key mode, *xterm*.

P_s = 1 0 6 0 → Reset legacy keyboard emulation, i.e, X11R6, *xterm*.

P_s = 1 0 6 1 → Reset keyboard emulation to Sun/PC style, *xterm*.

P_s = 2 0 0 1 → Disable readline mouse button-1, *xterm*.

P_s = 2 0 0 2 → Disable readline mouse button-2, *xterm*.

P_s = 2 0 0 3 → Disable readline mouse button-3, *xterm*.

P_s = 2 0 0 4 → Reset bracketed paste mode, *xterm*.

P_s = 2 0 0 5 → Disable readline character-quoting, *xterm*.

P_s = 2 0 0 6 → Disable readline newline pasting, *xterm*.

CSI P_m m Character Attributes (SGR).

P_s = 0 → Normal (default), VT100.

P_s = 1 → Bold, VT100.

P_s = 2 → Faint, decreased intensity, ECMA-48 2nd.

P_s = 3 → Italicized, ECMA-48 2nd.

P_s = 4 → Underlined, VT100.

P_s = 5 → Blink, VT100.

This appears as Bold in X11R6 xterm.

P_s = 7 → Inverse, VT100.

P_s = 8 → Invisible, i.e., hidden, ECMA-48 2nd, VT300.

P_s = 9 → Crossed-out characters, ECMA-48 3rd.

P_s = 2 1 → Doubly-underlined, ECMA-48 3rd.

P_s = 2 2 → Normal (neither bold nor faint), ECMA-48 3rd.

P_s = 2 3 → Not italicized, ECMA-48 3rd.

P_s = 2 4 → Not underlined, ECMA-48 3rd.
P_s = 2 5 → Steady (not blinking), ECMA-48 3rd.
P_s = 2 7 → Positive (not inverse), ECMA-48 3rd.
P_s = 2 8 → Visible, i.e., not hidden, ECMA-48 3rd, VT300.
P_s = 2 9 → Not crossed-out, ECMA-48 3rd.
P_s = 3 0 → Set foreground color to Black.
P_s = 3 1 → Set foreground color to Red.
P_s = 3 2 → Set foreground color to Green.
P_s = 3 3 → Set foreground color to Yellow.
P_s = 3 4 → Set foreground color to Blue.
P_s = 3 5 → Set foreground color to Magenta.
P_s = 3 6 → Set foreground color to Cyan.
P_s = 3 7 → Set foreground color to White.
P_s = 3 9 → Set foreground color to default, ECMA-48 3rd.
P_s = 4 0 → Set background color to Black.
P_s = 4 1 → Set background color to Red.
P_s = 4 2 → Set background color to Green.
P_s = 4 3 → Set background color to Yellow.
P_s = 4 4 → Set background color to Blue.
P_s = 4 5 → Set background color to Magenta.
P_s = 4 6 → Set background color to Cyan.
P_s = 4 7 → Set background color to White.
P_s = 4 9 → Set background color to default, ECMA-48 3rd.

Some of the above note the edition of ECMA-48 which first describes a feature. In its successive editions from 1979 to 1991 (*2nd* 1979, *3rd* 1984, *4th* 1986, and *5th* 1991), ECMA-48 listed codes through 6 5 (skipping several toward the end of the range). Most of the ECMA-48 codes not implemented in *xterm* were never implemented in a hardware terminal. Several (such as 3 9 and 4 9) are either noted in ECMA-48 as implementation defined, or described in vague terms.

The successive editions of ECMA-48 give little attention to changes from one edition to the next, except to comment on features which have become obsolete. ECMA-48 1st (1976) is unavailable; there is no reliable source of information which states whether "ANSI" color was defined in that edition, or later (1979). The VT100 (1978) implemented the most commonly used non-color video attributes which are given in the 2nd edition.

While 8-color support is described in ECMA-48 2nd edition, the VT500 series (introduced in 1993) were the first DEC terminals implementing "ANSI" color. The DEC terminal's use of color is known to differ from *xterm*; useful documentation on this series became available too late to influence *xterm*.

If 16-color support is compiled, the following *aixterm* controls apply. Assume that *xterm*'s resources are set so that the ISO color codes are the first 8 of a set of 16. Then the *aixterm* colors are the bright versions of the ISO colors:

P_s = 9 0 → Set foreground color to Black.
P_s = 9 1 → Set foreground color to Red.
P_s = 9 2 → Set foreground color to Green.
P_s = 9 3 → Set foreground color to Yellow.
P_s = 9 4 → Set foreground color to Blue.
P_s = 9 5 → Set foreground color to Magenta.
P_s = 9 6 → Set foreground color to Cyan.
P_s = 9 7 → Set foreground color to White.
P_s = 1 0 0 → Set background color to Black.
P_s = 1 0 1 → Set background color to Red.
P_s = 1 0 2 → Set background color to Green.
P_s = 1 0 3 → Set background color to Yellow.
P_s = 1 0 4 → Set background color to Blue.
P_s = 1 0 5 → Set background color to Magenta.
P_s = 1 0 6 → Set background color to Cyan.
P_s = 1 0 7 → Set background color to White.

If *xterm* is compiled with the 16-color support disabled, it supports the following, from *rxvt*:

P_s = 1 0 0 → Set foreground and background color to default.

XTerm maintains a color palette whose entries are identified by an index beginning with zero. If 88- or 256-color support is compiled, the following apply:

- All parameters are decimal integers.
- RGB values range from zero (0) to 255.
- The 88- and 256-color support uses *subparameters* described in ISO-8613-6 for *indexed* color. ISO-8613-6 also mentions *direct color*, using a similar scheme. *xterm* supports that, too.
- *xterm* allows either colons (standard) or semicolons (legacy) to separate the subparameters (but after the first colon, colons must be used).

The indexed- and direct-color features are summarized in the FAQ, which explains why semicolon is accepted as a subparameter delimiter:

Can I set a color by its number?

These ISO-8613-6 controls (marked in ECMA-48 5th edition as “reserved for future standardization”) are supported by *xterm*:

P_s = 3 8 : 2 : P_i : P_r : P_g : P_b → Set foreground color using RGB values. If *xterm* is not compiled with direct-color support, it uses the closest match in its palette for the given RGB P_r/P_g/P_b. The color space identifier P_i is ignored.

P_s = 3 8 : 5 : P_s → Set foreground color to P_s, using indexed color.

P_s = 4 8 : 2 : P_i : P_r : P_g : P_b → Set background color using RGB values. If *xterm* is not compiled with direct-color support, it uses the closest match in its palette for the given RGB $P_r/P_g/P_b$. The color space identifier P_i is ignored.

P_s = 4 8 : 5 : P_s → Set background color to P_s, using indexed color.

This variation on ISO-8613-6 is supported for compatibility with KDE konsole:

P_s = 3 8 ; 2 ; P_r ; P_g ; P_b → Set foreground color using RGB values. If *xterm* is not compiled with direct-color support, it uses the closest match in its palette for the given RGB $P_r/P_g/P_b$.

P_s = 4 8 ; 2 ; P_r ; P_g ; P_b → Set background color using RGB values. If *xterm* is not compiled with direct-color support, it uses the closest match in its palette for the given RGB $P_r/P_g/P_b$.

In each case, if *xterm* is compiled with direct-color support, and the resource **directColor** is true, then rather than choosing the closest match, *xterm* asks the X server to directly render a given color.

CSI > P_p ; P_v m

CSI > P_p m Set/reset key modifier options (XTMODKEYS), *xterm*. Set or reset resource-values used by *xterm* to decide whether to construct escape sequences holding information about the modifiers pressed with a given key.

The first parameter P_p identifies the resource to set/reset. The second parameter P_v is the value to assign to the resource.

If the second parameter is omitted, the resource is reset to its initial value. Value 5 is reserved for input via the **string** action.

P_p = 0 → **modifyKeyboard**.
P_p = 1 → **modifyCursorKeys**.
P_p = 2 → **modifyFunctionKeys**.
P_p = 3 → **modifyKeypadKeys**.
P_p = 4 → **modifyOtherKeys**.
P_p = 6 → **modifyModifierKeys**.
P_p = 7 → **modifySpecialKeys**.

If no parameters are given, all resources are reset to their initial values. See *Alt and Meta Keys* for more details on the **modifyOtherKeys** feature.

CSI ? P_p m Query key modifier options (XTQMODKEYS), *xterm*.

The parameter P_p identifies the resource to query.

P_p = 0 → **modifyKeyboard**.
P_p = 1 → **modifyCursorKeys**.
P_p = 2 → **modifyFunctionKeys**.
P_p = 3 → **modifyKeypadKeys**.
P_p = 4 → **modifyOtherKeys**.
P_p = 6 → **modifyModifierKeys**.
P_p = 7 → **modifySpecialKeys**.

XTerm's response can be used to restore this state, because it is formatted as an XTMODKEYS control, i.e.,

CSI > P_p m

where

P_p = 0 → **modifyKeyboard**.
P_p = 1 → **modifyCursorKeys**.
P_p = 2 → **modifyFunctionKeys**.
P_p = 3 → **modifyKeypadKeys**.
P_p = 4 → **modifyOtherKeys**.
P_p = 6 → **modifyModifierKeys**.
P_p = 7 → **modifySpecialKeys**.

CSI P_s n — Device Status Report (DSR).

P_s = 5 → Status Report.
Result ("OK") is CSI 0 n
P_s = 6 → Report Cursor Position (CPR) [row;column].
Result is CSI r ; c R

Note: it is possible for this sequence to be sent by a function key. For example, with the default keyboard configuration the shifted F3 key may send (with shift-, control-, alt-modifiers)

CSI 1 ; 2 R , or
CSI 1 ; 5 R , or
CSI 1 ; 6 R , etc.

The second parameter encodes the modifiers; values range from 2 to 16. See the section **PC-Style Function Keys** for the codes. The **modifyFunctionKeys** and **modifyKeyboard** resources can change the form of the string sent from the modified F3 key.

CSI > P_s n — Disable key modifier options, *xterm*. These modifiers may be enabled via the CSI > P_m m sequence. This control sequence corresponds to a resource value of "−1", which cannot be set with the other sequence.

The parameter identifies the resource to be disabled:

P_s = 0 → **modifyKeyboard**.
P_s = 1 → **modifyCursorKeys**.
P_s = 2 → **modifyFunctionKeys**.
P_s = 3 → **modifyKeypadKeys**.
P_s = 4 → **modifyOtherKeys**.
P_s = 6 → **modifyModifierKeys**.
P_s = 7 → **modifySpecialKeys**.

If the parameter is omitted, **modifyFunctionKeys** is disabled. When **modifyFunctionKeys** is disabled, *xterm* uses the modifier keys to make an extended sequence of function keys rather than adding a parameter to each function key to denote the modifiers.

CSI ? P_s n — Device Status Report (DSR, DEC-specific).

P_s = 6 → Report Cursor Position (DECXCPR). The response [row;column] is returned as
CSI ? r ; c R

(assumes the default page, i.e., “1”).

P_s = 1 5 → Report Printer status. The response is
CSI ? 1 0 n (ready). or
CSI ? 1 1 n (not ready).

P_s = 2 5 → Report UDK status. The response is
CSI ? 2 0 n (unlocked)
or
CSI ? 2 1 n (locked).

P_s = 2 6 → Report Keyboard status. The response is
CSI ? 2 7 ; 1 ; 0 ; 0 n (North American).
The last two parameters apply to VT300 & up (keyboard ready) and VT400 & up (LK01) respectively.

P_s = 5 5 → Report Locator status. The response is CSI ? 5 0 n Locator available, if compiled-in, or CSI ? 5 3 n No Locator, if not.

P_s = 5 6 → Report Locator type. The response is CSI ? 5 7 ; 1 n Mouse, if compiled-in, or CSI ? 5 7 ; 0 n Cannot identify, if not.

P_s = 6 2 → Report macro space (DECMSR). The response is CSI P_n * {.

P_s = 6 3 → Report memory checksum (DECCKSR), VT420 and up. The response is DCS P_t ! ~ x x x x ST.
P_t is the request id (from an optional parameter to the request).
The x’s are hexadecimal digits 0-9 and A-F.

P_s = 7 5 → Report data integrity. The response is CSI ? 7 0 n (ready, no errors).

P_s = 8 5 → Report multi-session configuration. The response is CSI ? 8 3 n (not configured for multiple-session operation).

CSI > P_s p
Set resource value **pointerMode** (XTSMPOINTER), xterm. This is used by *xterm* to decide whether to hide the pointer cursor as the user types.
Valid values for the parameter:
P_s = 0 → never hide the pointer.
P_s = 1 → hide if the mouse tracking mode is not enabled.
P_s = 2 → always hide the pointer, except when leaving the window.
P_s = 3 → always hide the pointer, even if leaving/entering the window.
If no parameter is given, *xterm* uses the default, which is 1.

CSI ! p
Soft terminal reset (DECSTR), VT220 and up.

CSI P_l ; P_c " p
Set conformance level (DECSCL), VT220 and up.
The first parameter selects the conformance level. Valid values are:
P_l = 6 1 → level 1, e.g., VT100.
P_l = 6 2 → level 2, e.g., VT200.
P_l = 6 3 → level 3, e.g., VT300.
P_l = 6 4 → level 4, e.g., VT400.
P_l = 6 5 → level 5, e.g., VT500.

The second parameter selects the C1 control transmission mode. This is an optional parameter, ignored in conformance level 1. Valid values are:
P_c = 0 → 8-bit controls.
P_c = 1 → 7-bit controls (DEC factory default).
P_c = 2 → 8-bit controls.
The 7-bit and 8-bit control modes can also be set by S7C1T and S8C1T, but DECSCL is preferred.

CSI P_s $ p
Request ANSI mode (DECRQM). For VT300 and up, reply DECRPM is
CSI P_s ; P_m $ y
where P_s is the mode number as in SM/RM, and P_m is the mode value:
0 - not recognized
1 - set
2 - reset
3 - permanently set
4 - permanently reset

CSI ? P_s $ p
Request DEC private mode (DECRQM). For VT300 and up, reply DECRPM is
CSI ? P_s ; P_m $ y
where P_s is the mode number as in DECSET/DECSET, P_m is the mode value as in the ANSI DECRQM.
Two private modes are read-only (i.e., 1 3 and 1 4), provided only for reporting their values using this control sequence. They correspond to the resources **cursorBlink** and **cursorBlinkXOR**.

CSI # p
CSI P_m # p Push video attributes onto stack (XTPUSHSGR), *xterm*. This is an alias for CSI # { , used to work around language limitations of C#.

CSI > P_s q
P_s = 0 → Report *xterm* name and version (XTVERSION).
The response is a DSR sequence identifying the version:
DCS > | text ST

CSI P_s q Load LEDs (DECLL), VT100.
P_s = 0 → Clear all LEDS (default).
P_s = 1 → Light Num Lock.
P_s = 2 → Light Caps Lock.
P_s = 3 → Light Scroll Lock.
P_s = 2 1 → Extinguish Num Lock.
P_s = 2 2 → Extinguish Caps Lock.
P_s = 2 3 → Extinguish Scroll Lock.

CSI P_s SP q Set cursor style (DECSCUSR), VT520.
P_s = 0 → blinking block.
P_s = 1 → blinking block (default).

P_s = 2 → steady block.
P_s = 3 → blinking underline.
P_s = 4 → steady underline.
P_s = 5 → blinking bar, *xterm*.
P_s = 6 → steady bar, *xterm*.

CSI P_s " q Select character protection attribute (DECSCA), VT220. Valid values for the parameter:
P_s = 0 → DECSED and DECSEL can erase (default).
P_s = 1 → DECSED and DECSEL cannot erase.
P_s = 2 → DECSED and DECSEL can erase.

CSI # q Pop video attributes from stack (XTPOPSGR), *xterm*. This is an alias for CSI # }, used to work around language limitations of C#.

CSI P_s ; P_s r
Set Scrolling Region [top;bottom] (default = full size of window) (DECSTBM), VT100.

CSI ? P_m r Restore DEC Private Mode Values (XTRESTORE), xterm. The value of P_s previously saved is restored. P_s values are the same as for DECSET.
Like Restore Cursor (DECRC), this uses a one-level cache. Unlike Restore Cursor, specific settings can be saved and restored independently. Only those modes listed as parameters are restored.

CSI P_t ; P_l ; P_b ; P_r ; P_m $ r
Change Attributes in Rectangular Area (DECCARA), VT400 and up.
P_t ; P_l ; P_b ; P_r denotes the rectangle.
P_m denotes the SGR attributes to change: 0 , 1 , 4 , 5 , 7 , 8 . Setting SGR 0 resets modes 1 , 4 , 5 , 7 . Those modes can be individually reset with SGR 2 2 , 2 4 , 2 5 and 2 7 . Setting SGR 8 is an *xterm* extension; it may be reset with SGR 2 8 . See DECSACE.

CSI s Save cursor, available only when DECLRMM is disabled (SCOSC, also ANSI.SYS).

CSI P_l ; P_r s
Set left and right margins (DECSLRM), VT420 and up. This is available only when DECLRMM is enabled.

CSI > P_s s Set/reset shift-escape options (XTSHIFTESCAPE), *xterm*. This corresponds to the **shiftEscape** resource.
Valid values for the parameter:
P_s = 0 → allow shift-key to override mouse protocol.
P_s = 1 → conditionally allow shift-key as modifier in mouse protocol.
These resource values are disallowed in the control sequence:
P_s = 2 → always allow shift-key as modifier in mouse protocol.
P_s = 3 → never allow shift-key as modifier in mouse protocol.
If no parameter is given, *xterm* uses the default, which is 0 .

CSI ? P_m s Save DEC Private Mode Values (XTSAVE), xterm. P_s values are the same as for DECSET.
Like Save Cursor (DECSC), this uses a one-level cache. Unlike Save Cursor, specific settings can be saved and restored independently. Only those modes listed as parameters are saved.

CSI P_s ; P_s ; P_s t

Window manipulation (XTWINOPS), *dtterm*, extended by *xterm*. These controls may be disabled using the **allowWindowOps** resource.

xterm uses *Extended Window Manager Hints* (EWMH) to maximize the window. Some window managers have incomplete support for EWMH. For instance, *fvwm*, *flwm* and *quartz-wm* advertise support for maximizing windows horizontally or vertically, but in fact equate those to the maximize operation.

Valid values for the first (and any additional parameters) are:

P_s = 1 → De-iconify window.

P_s = 2 → Iconify window.

P_s = 3 ; *x* ; *y* → Move window to [x, y].

P_s = 4 ; *height* ; *width* → Resize the *xterm* window to given height and width in pixels. Omitted parameters reuse the current height or width. Zero parameters use the display's height or width.

P_s = 5 → Raise the *xterm* window to the front of the stacking order.

P_s = 6 → Lower the *xterm* window to the bottom of the stacking order.

P_s = 7 → Refresh the *xterm* window.

P_s = 8 ; *height* ; *width* → Resize the text area to given height and width in characters. Omitted parameters reuse the current height or width. Zero parameters use the display's height or width.

P_s = 9 ; 0 → Restore maximized window.

P_s = 9 ; 1 → Maximize window (i.e., resize to screen size).

P_s = 9 ; 2 → Maximize window vertically.

P_s = 9 ; 3 → Maximize window horizontally.

P_s = 1 0 ; 0 → Undo full-screen mode.

P_s = 1 0 ; 1 → Change to full-screen.

P_s = 1 0 ; 2 → Toggle full-screen.

P_s = 1 1 → Report *xterm* window state.

If the *xterm* window is non-iconified, it returns CSI 1 t .

If the *xterm* window is iconified, it returns CSI 2 t .

P_s = 1 3 → Report *xterm* window position.

Note: X Toolkit positions can be negative, but the reported values are unsigned, in the range 0-65535. Negative values correspond to 32768-65535.

Result is CSI 3 ; *x* ; *y* t

P_s = 1 3 ; 2 → Report *xterm* text-area position.

Result is CSI 3 ; *x* ; *y* t

P_s = 1 4 → Report *xterm* text area size in pixels.

Result is CSI 4 ; *height* ; *width* t

P_s = 1 4 ; 2 → Report *xterm* window size in pixels.

Normally *xterm*'s *window* is larger than its *text area*, since it includes the frame (or decoration) applied by the window manager, as well as the area used by a scroll-bar.

Result is CSI 4 ; *height* ; *width* t

P_s = 1 5 → Report size of the screen in pixels.

Result is CSI 5 ; *height* ; *width* t

P_s = 1 6 → Report *xterm* character cell size in pixels.

Result is CSI 6 ; *height* ; *width* t

P_s = 1 8 → Report the size of the text area in characters.

Result is CSI 8 ; *height* ; *width* t

P_s = 1 9 → Report the size of the screen in characters.

Result is CSI 9 ; *height* ; *width* t

P_s = 2 0 → Report *xterm* window's icon label.

Result is OSC L *label* ST

P_s = 2 1 → Report *xterm* window's title.

Result is OSC 1 *label* ST

P_s = 2 2 ; 0 → Save *xterm* icon and window title on stack.

P_s = 2 2 ; 1 → Save *xterm* icon title on stack.

P_s = 2 2 ; 2 → Save *xterm* window title on stack.

P_s = 2 3 ; 0 → Restore *xterm* icon and window title from stack.

P_s = 2 3 ; 1 → Restore *xterm* icon title from stack.

P_s = 2 3 ; 2 → Restore *xterm* window title from stack.

P_s >= 2 4 → Resize to P_s lines (DECSLPP), VT340 and VT420.

xterm adapts this by resizing its window.

XTWINOPS 2 2 (save/push title) and 2 3 (restore/pop title) accept an optional third parameter for direct access to the stack. Parameters in the range 1 through 10, may be used to store the title into the stack or retrieve the title from the stack without pushing/popping.

CSI > P_m t This *xterm* control sets one or more features of the title modes (XTSMTITLE), xterm. Each parameter enables a single feature.

P_s = 0 → Set window/icon labels using hexadecimal.

P_s = 1 → Query window/icon labels using hexadecimal.

P_s = 2 → Set window/icon labels using UTF-8.

P_s = 3 → Query window/icon labels using UTF-8. (See discussion of **Title Modes**)

If no parameters are given, title mode features are set to the initial (compiled-in) default.

CSI P_s SP t Set warning-bell volume (DECSWBV), VT520.

P_s = 0 or 1 → off.

P_s = 2 , 3 or 4 → low.

P_s = 5 , 6 , 7 , or 8 → high.

CSI P_t ; P_l ; P_b ; P_r ; P_m $ t

Reverse Attributes in Rectangular Area (DECRARA), VT400 and up.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

P_m denotes the attributes to reverse, i.e., 0, 1, 4, 5, 7, 8. Reversing SGR 0 reverses modes 1, 4, 5, 7. Reversing SGR 8 is an *xterm* extension. See DECSACE.

CSI u — Restore cursor (SCORC, also ANSI.SYS).

CSI & u — User-Preferred Supplemental Set (DECRQUPSS), VT320, VT510. Response is DECAUPSS.

CSI P_s SP u — Set margin-bell volume (DECSMBV), VT520.
P_s = 0, 5, 6, 7, or 8 → high.
P_s = 1 → off.
P_s = 2, 3 or 4 → low.

CSI " v — Request Displayed Extent (DECRQDE), VT340, VT420.
Response is
CSI P_h ; P_w ; P_c ; P_r ; P_p " w
where
P_h is the number of lines of the current page
P_w is the number of columns of the current page
P_c is the column number at the top-left of the window
P_r is the row number at the top-left of the window
P_p is the current page number

CSI P_t ; P_l ; P_b ; P_r ; P_p ; P_t ; P_l ; P_p $ v
Copy Rectangular Area (DECCRA), VT400 and up.
P_t ; P_l ; P_b ; P_r denotes the rectangle.
P_p denotes the source page.
P_t ; P_l denotes the target location.
P_p denotes the target page.

CSI P_s $ w — Request presentation state report (DECRQPSR), VT320 and up.
P_s = 0 → error.
P_s = 1 → cursor information report (DECCIR).
Response is
DCS 1 $ u P_t ST
Refer to the VT420 programming manual, which requires six pages to document the data string P_t,
P_s = 2 → tab stop report (DECTABSR).
Response is
DCS 2 $ u P_t ST
The data string P_t is a list of the tab-stops, separated by "/" characters.

CSI P_t ; P_l ; P_b ; P_r ' w
Enable Filter Rectangle (DECEFR), VT420 and up.
Parameters are [top;left;bottom;right].
Defines the coordinates of a filter rectangle and activates it. Anytime the locator is detected outside of the filter rectangle, an outside rectangle event is generated and the rectangle is disabled. Filter rectangles are always treated as "one-shot" events. Any parameters that are omitted default to the current locator position. If all parameters are omitted, any locator motion will be reported. DECELR always cancels any previous rectangle definition.

CSI P_s x — Request Terminal Parameters (DECREQTPARM).
if P_s is a "0" (default) or "1", and *xterm* is emulating VT100, the control sequence elicits a

response of the same form whose parameters describe the terminal:

P_s → the given P_s incremented by 2.

P_n = 1 ← no parity.

P_n = 1 ← eight bits.

P_n = 1 ← 2 8 transmit 38.4k baud.

P_n = 1 ← 2 8 receive 38.4k baud.

P_n = 1 ← clock multiplier.

P_n = 0 ← STP flags.

CSI P_s * x Select Attribute Change Extent (DECSACE), VT420 and up.

P_s = 0 → from start to end position, wrapped.

P_s = 1 → from start to end position, wrapped.

P_s = 2 → rectangle (exact).

Modes 0 and 1 are the *stream* modes of the DECCARA and DECRARA controls. There are several aspects to *stream* versus *rectangle* modes:

1) In both *stream* and *rectangle* modes, the row- and column-positions are affected by Origin Mode.

2) In *rectangle* mode, cells outside the row- and column-positions are unaffected. In *stream* mode, the row- and column-positions are the starting and ending cells, with wrapping which ignores Origin Mode.

3) In *stream* mode, those controls affect only cells where a character was drawn. In *rectangle* mode, cells where no character was drawn are first filled in with a space.

CSI P_c ; P_t ; P_l ; P_b ; P_r $ x

Fill Rectangular Area (DECFRA), VT420 and up.

P_c is the character to use.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

CSI P_s # y Select checksum extension (XTCHECKSUM), *xterm*. The bits of P_s modify the calculation of the checksum returned by DECRQCRA:

0 → do not negate the result.

1 → do not report the VT100 video attributes.

2 → do not omit checksum for blanks.

3 → omit checksum for cells not explicitly initialized.

4 → do not mask cell value to 8 bits or ignore combining characters.

CSI P_i ; P_g ; P_t ; P_l ; P_b ; P_r * y

Request Checksum of Rectangular Area (DECRQCRA), VT420 and up. Response is

DCS P_i ! ~ x x x x ST

P_i is the request id.

P_g is the page number.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

The x's are hexadecimal digits 0-9 and A-F.

CSI P_s ; P_u ' z

Enable Locator Reporting (DECELR).

Valid values for the first parameter:

P_s = 0 → Locator disabled (default).

P_s = 1 → Locator enabled.

P_s = 2 → Locator enabled for one report, then disabled.

The second parameter specifies the coordinate unit for locator reports.

Valid values for the second parameter:

P_u = 0 or omitted → default to character cells.

P_u = 1 ← device physical pixels.

P_u = 2 ← character cells.

CSI P_t ; P_l ; P_b ; P_r $ z

Erase Rectangular Area (DECERA), VT400 and up.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

CSI P_m ' { Select Locator Events (DECSLE).

Valid values for the first (and any additional parameters) are:

P_s = 0 → only respond to explicit host requests (DECRQLP). This is default. It also cancels any filter rectangle.

P_s = 1 → report button down transitions.

P_s = 2 → do not report button down transitions.

P_s = 3 → report button up transitions.

P_s = 4 → do not report button up transitions.

CSI # {

CSI P_m # { Push video attributes onto stack (XTPUSHSGR), *xterm*. The optional parameters correspond to the SGR encoding for video attributes, except for colors (which do not have a unique SGR code):

P_s = 1 → Bold.

P_s = 2 → Faint.

P_s = 3 → Italicized.

P_s = 4 → Underlined.

P_s = 5 → Blink.

P_s = 7 → Inverse.

P_s = 8 → Invisible.

P_s = 9 → Crossed-out characters.

P_s = 2 1 → Doubly-underlined.

P_s = 3 0 → Foreground color.

P_s = 3 1 → Background color.

If no parameters are given, all of the video attributes are saved. The stack is limited to 10 levels.

CSI P_t ; P_l ; P_b ; P_r $ {

Selective Erase Rectangular Area (DECSERA), VT400 and up.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

CSI P_t ; P_l ; P_b ; P_r # |

Report selected graphic rendition (XTREPORTSGR), *xterm*. The response is an SGR sequence which contains the attributes which are common to all cells in a rectangle.

P_t ; P_l ; P_b ; P_r denotes the rectangle.

CSI P_s $ | Select columns per page (DECSCPP), VT340.

P_s = 0 → 80 columns, default if P_s omitted.

P_s = 8 0 → 80 columns.

P_s = 1 3 2 → 132 columns.

CSI P_s ' | Request Locator Position (DECRQLP).

Valid values for the parameter are:

P_s = 0 , 1 or omitted → transmit a single DECLRP locator report.

If Locator Reporting has been enabled by a DECELR, *xterm* will respond with a DECLRP Locator Report. This report is also generated on button up and down events if they have been enabled with a DECSLE, or when the locator is detected outside of a filter rectangle, if filter rectangles have been enabled with a DECEFR.

← CSI P_e ; P_b ; P_r ; P_c ; P_p & w

Parameters are [*event*;*button*;*row*;*column*;*page*].

Valid values for the event:

P_e = 0 ← locator unavailable - no other parameters sent.

P_e = 1 ← request - *xterm* received a DECRQLP.

P_e = 2 ← left button down.

P_e = 3 ← left button up.

P_e = 4 ← middle button down.

P_e = 5 ← middle button up.

P_e = 6 ← right button down.

P_e = 7 ← right button up.

P_e = 8 ← M4 button down.

P_e = 9 ← M4 button up.

P_e = 1 0 ← locator outside filter rectangle.

The "*button*" parameter is a bitmask indicating which buttons are pressed:

P_b = 0 ← no buttons down.

P_b & 1 ← right button down.

P_b & 2 ← middle button down.

P_b & 4 ← left button down.

P_b & 8 ← M4 button down.

The "*row*" and "*column*" parameters are the coordinates of the locator position in the *xterm* window, encoded as ASCII decimal.

The "*page*" parameter is not used by xterm.

CSI P_s * | Select number of lines per screen (DECSNLS), VT420 and up.

CSI # } Pop video attributes from stack (XTPOPSGR), *xterm*. Popping restores the video-attributes which were saved using XTPUSHSGR to their previous state.

CSI P_s ; P_f ; P_b , |
Assign Color (DECAC), VT525 only.
P_s selects the color item
P_f is the foreground color index 0..15
P_b is the background color index 0..15
Color items:
P_s = 1 → normal text
P_s = 2 → window frame
xterm uses the SGR color palette with DECAC color item 1 to update the VT100 window colors, like OSC 1 0 and 1 1 .

CSI P_s ; P_f ; P_b , }
Alternate Text Color (DECATC), VT525 only. This feature specifies the colors to use when DECSTGLT is selected to 1 or 2.
P_s selects attribute combinations
P_f is the foreground color index 0..15
P_b is the background color index 0..15
Attribute combinations:
P_s = 0 → normal text
P_s = 1 → bold
P_s = 2 → reverse
P_s = 3 → underline
P_s = 4 → blink
P_s = 5 → bold reverse
P_s = 6 → bold underline
P_s = 7 → bold blink
P_s = 8 → reverse underline
P_s = 9 → reverse blink
P_s = 1 0 → underline blink
P_s = 1 1 → bold reverse underline
P_s = 1 2 → bold reverse blink
P_s = 1 3 → bold underline blink
P_s = 1 4 → reverse underline blink
P_s = 1 5 → bold reverse underline blink

CSI P_s ' } Insert P_s Column(s) (default = 1) (DECIC), VT420 and up.

CSI P_s $ } Select active status display (DECSASD), VT320 and up.
P_s = 0 → main (default)
P_s = 1 → status line

CSI P_s ' ~ Delete P_s Column(s) (default = 1) (DECDC), VT420 and up.

CSI P_s $ ~ Select status line type (DECSSDT), VT320 and up.
P_s = 0 → none
P_s = 1 → indicator (default)

P_s = 2 → host-writable.

Operating System Commands

OSC P_s ; P_t BEL
OSC P_s ; P_t ST

Set Text Parameters, *xterm*. Some control sequences return information:

- For colors and font, if P_t is a "?", the control sequence elicits a response which consists of the control sequence which would set the corresponding value.
- A few of these control sequences began with *dtterm* (codes 0, 1, and 2). Code 3 in *dtterm* sets the working directory for the next session. *XTerm* does that with the **spawn−new−terminal** action.

XTerm accepts either BEL or ST for terminating OSC sequences, and when returning information, uses the same terminator used in a query. While the latter is preferred, the former is supported for legacy applications:

- Although documented in the changes for X.V10R4 (December 1986), BEL as a string terminator dates from X11R4 (December 1989).
- Since XFree86-3.1.2Ee (August 1996), xterm has accepted ST (the documented string terminator in ECMA-48).

P_s specifies the type of operation to perform:

P_s = 0 → Change Icon Name and Window Title to P_t.
P_s = 1 → Change Icon Name to P_t.
P_s = 2 → Change Window Title to P_t.
P_s = 3 → Set X property on top-level window. P_t should be in the form "*prop=value*", or just "*prop*" to delete the property.
P_s = 4 ; *c* ; *spec* → Change Color Number *c* to the color specified by *spec*.

The *spec* can be a name or RGB specification as per *XParseColor*. Any number of *c/spec* pairs may be given. The color numbers correspond to the ANSI colors 0-7, their bright versions 8-15, and if supported, the remainder of the 88-color or 256-color table.

If a "?" is given rather than a name or RGB specification, *xterm* replies with a control sequence of the same form which can be used to set the corresponding color. Because more than one pair of color number and specification can be given in one control sequence, *xterm* can make more than one reply.

P_s = 5 ; *c* ; *spec* → Change Special Color Number *c* to the color specified by *spec*.

The *spec* parameter can be a name or RGB specification as per *XParseColor*. Any number of *c/spec* pairs may be given. The special colors can also be set by adding the maximum number of colors (e.g., 88 or 256) to these codes in an OSC 4 control:

P_c = 0 ← resource **colorBD** (BOLD).

P_c = 1 ← resource **colorUL** (UNDERLINE).
P_c = 2 ← resource **colorBL** (BLINK).
P_c = 3 ← resource **colorRV** (REVERSE).
P_c = 4 ← resource **colorIT** (ITALIC).

P_s = 6 ; c ; f → Enable/disable Special Color Number c. The second parameter tells *xterm* to enable the corresponding color mode if nonzero, disable it if zero. OSC 6 is the same as OSC 1 0 6.

If no parameters are given, this control has no effect.

The 10 colors (below) which may be set or queried using 1 0 through 1 9 are denoted *dynamic colors*, since the corresponding control sequences were the first means for setting *xterm*'s colors dynamically, i.e., after it was started. They are not the same as the ANSI colors (however, the dynamic text foreground and background colors are used when ANSI colors are reset using SGR 3 9 and 4 9, respectively). These controls may be disabled using the **allowColorOps** resource. At least one parameter is expected for P_t. Each successive parameter changes the next color in the list. The value of P_s tells the starting point in the list. The colors are specified by name or RGB specification as per *XParseColor*.

Resource	Description
foreground	VT100 text foreground color
background	VT100 text background color
cursorColor	text cursor color
pointerColor	pointer foreground color
pointerColorBackground	pointer background
(**foreground**)	Tektronix foreground color
(**background**)	Tektronix background color
highlightColor	highlight background color
(**cursorColor**)	Tektronix cursor color
highlightTextColor	highlight foreground color

The Tektronix colors are initially set from the VT100 colors, but after that can be set independently using these control sequences.

If a "?" is given rather than a name or RGB specification, *xterm* replies with a control sequence of the same form which can be used to set the corresponding dynamic color. Because more than one pair of color number and specification can be given in one control sequence, *xterm* can make more than one reply.

P_s = 1 0 → Change VT100 text foreground color to P_t.
P_s = 1 1 → Change VT100 text background color to P_t.
P_s = 1 2 → Change text cursor color to P_t.
P_s = 1 3 → Change pointer foreground color to P_t.

P_s = 1 4 → Change pointer background color to P_t.
P_s = 1 5 → Change Tektronix foreground color to P_t.
P_s = 1 6 → Change Tektronix background color to P_t.
P_s = 1 7 → Change highlight background color to P_t.
P_s = 1 8 → Change Tektronix cursor color to P_t.
P_s = 1 9 → Change highlight foreground color to P_t.

P_s = 2 2 → Change pointer cursor shape to P_t. The parameter P_t sets the **pointerShape** resource. If P_t is empty, or does not match any of the standard names, *xterm* uses the resource's default "xterm" shape.

P_s = 4 6 → Change Log File to P_t. The parameter P_t sets the **logFile** resource. Logging is normally disabled by a compile-time option.

P_s = 5 0 → Set Font to P_t. These controls may be disabled using the **allowFontOps** resource. If P_t begins with a "#", index in the font menu, relative (if the next character is a plus or minus sign) or absolute. A number is expected but not required after the sign (the default is the current entry for relative, zero for absolute indexing).

The same rule (plus or minus sign, optional number) is used when querying the font. The remainder of P_t is ignored.

A font can be specified after a "#" index expression, by adding a space and then the font specifier.

If the **TrueType Fonts** menu entry is set (the **renderFont** resource), then this control sets/queries the **faceName** resource.

P_s = 5 1 → reserved for Emacs shell.

P_s = 5 2 → Manipulate Selection Data. These controls may be disabled using the **allowWindowOps** resource. The parameter P_t is parsed as

P_c ; P_d

The first, P_c, may contain zero or more characters from the set c , p , q , s , 0 , 1 , 2 , 3 , 4 , 5 , 6 , and 7 . It is used to construct a list of selection parameters for clipboard, primary, secondary, select, or cut-buffers 0 through 7 respectively, in the order given. If the parameter is empty, *xterm* uses s 0 , to specify the configurable primary/clipboard selection and cut-buffer 0.

The second parameter, P_d, gives the selection data. Normally this is a string encoded in base64 (RFC-4648). The data becomes the new selection, which is then available for pasting by other applications.

If the second parameter is a ? , *xterm* replies to the host with the selection data encoded using the same protocol. It uses the first selection found by asking successively for each item from the list of selection parameters.

If the second parameter is neither a base64 string nor ?, then the selection is cleared.

P_s = 6 0 → Query allowed features (XTQALLOWED). *XTerm* replies with

OSC 6 0 ; P_t ST

where P_t is a comma-separated list of the *allowed* optional runtime feature categories, i.e., zero or more of these resource names:

allowColorOps
allowFontOps
allowMouseOps
allowPasteControls
allowTcapOps
allowTitleOps
allowWindowOps

Except for **allowPasteControls**, those resources can be enabled or disabled at runtime using menus. They cannot be changed using control sequences.

P_s = 6 1 → Query disallowed features (XTQDISALLOWED). The second parameter (i.e., the main feature category) must be one of the resource names returned by OSC 6 0 . *XTerm* replies with

OSC 6 1 ; P_t ST

where P_t is a comma-separated list of the optional runtime feature subcategories which would be disallowed if the main feature is disabled.

P_s = 6 2 → Query allowable features (XTQALLOWABLE). The second parameter (i.e., the main feature category) must be one of the resource names that might be returned by OSC 6 0 . *XTerm* replies with

OSC 6 2 ; P_t ST

where P_t is a comma-separated list of the optional runtime feature subcategories which correspond to the main feature category.

P_s = 1 0 4 ; *c* → Reset Color Number *c*. It is reset to the color specified by the corresponding X resource. Any number of *c* parameters may be given. These parameters correspond to the ANSI colors 0-7, their bright versions 8-15, and if supported, the remainder of the 88-color or 256-color table. If no parameters are given, the entire table will be reset.

P_s = 1 0 5 ; *c* → Reset Special Color Number *c*. It is reset to the color specified by the corresponding X resource. Any number of *c* parameters may be given. These parameters correspond to the special colors which can be set using an OSC 5 control (or by adding the maximum number of colors using an OSC 4 control).

If no parameters are given, all special colors will be reset.

P_s = 1 0 6 ; c ; f → Enable/disable Special Color Number c. The second parameter tells *xterm* to enable the corresponding color mode if nonzero, disable it if zero.

P_c = 0 ← resource **colorBDMode** (BOLD).
P_c = 1 ← resource **colorULMode** (UNDERLINE).
P_c = 2 ← resource **colorBLMode** (BLINK).
P_c = 3 ← resource **colorRVMode** (REVERSE).
P_c = 4 ← resource **colorITMode** (ITALIC).
P_c = 5 ← resource **colorAttrMode** (Override ANSI).

If no parameters are given, this control has no effect.

The *dynamic colors* can also be reset to their default (resource) values:
P_s = 1 1 0 → Reset VT100 text foreground color.
P_s = 1 1 1 → Reset VT100 text background color.
P_s = 1 1 2 → Reset text cursor color.
P_s = 1 1 3 → Reset pointer foreground color.
P_s = 1 1 4 → Reset pointer background color.
P_s = 1 1 5 → Reset Tektronix foreground color.
P_s = 1 1 6 → Reset Tektronix background color.
P_s = 1 1 7 → Reset highlight color.
P_s = 1 1 8 → Reset Tektronix cursor color.
P_s = 1 1 9 → Reset highlight foreground color.

P_s = I ; c → Set icon to file. Sun shelltool, CDE dtterm.
The file is expected to be XPM format, and uses the same search logic as the **iconHint** resource.

P_s = l ; c → Set window title. Sun shelltool, CDE dtterm.

P_s = L ; c → Set icon label. Sun shelltool, CDE dtterm.

Privacy Message

PM P_t ST xterm implements no PM functions; P_t is ignored. P_t need not be printable characters.

Special Keyboard Keys

Terminal keyboards have two types of keys:

- ordinary keys, which you would use as data, e.g., in a text file, and
- special keys, which you would use to tell *xterm* to perform some action.

XTerm detects all of these keys via X key-press and key-release events. It uses the **translations** resource to decide what to do with these events.

- Ordinary keys are handled with the **insert−seven−bit** action, or the **insert−eight−bit** action.
- Special keys may be handled with other resources. However, *xterm* also has built-in logic to map commonly-used special keys into characters which your keypress sends to the application running in *xterm*.

Special keyboard keys send control characters or escape sequences. This is a convention, making it convenient for applications to detect these keys, rather than a standard.

Alt and Meta Keys

Many keyboards have keys labeled “Alt”. Few have keys labeled “Meta”. However, *xterm*’s default translations use the *Meta* modifier. Common keyboard configurations assign the *Meta* modifier to an “Alt” key. By using *xmodmap* one may have the modifier assigned to a different key, and have “real” alt and meta keys. Here is an example:

```
! put meta on mod3 to distinguish it from alt
keycode 64 = Alt_L
clear mod1
add mod1 = Alt_L
keycode 115 = Meta_L
clear mod3
add mod3 = Meta_L
```

The **metaSendsEscape** resource (and **altSendsEscape** if **altIsNotMeta** is set) can be used to control the way the *Meta* modifier applies to ordinary keys unless the **modifyOtherKeys** resource is set:

- prefix a key with the ESC character.
- shift the key from codes 0-127 to 128-255 by adding 128.

The table shows the result for a given character “x” with modifiers according to the default translations with the resources set on or off. This assumes **altIsNotMeta** is set:

key	**altSendsEscape**	**metaSendsEscape**	**result**
x	off	off	x
Meta-x	off	off	shift
Alt-x	off	off	shift
Alt+Meta-x	off	off	shift
x	ON	off	x
Meta-x	ON	off	shift
Alt-x	ON	off	ESC x
Alt+Meta-x	ON	off	ESC shift
x	off	ON	x
Meta-x	off	ON	ESC x
Alt-x	off	ON	shift
Alt+Meta-x	off	ON	ESC shift
x	ON	ON	x
Meta-x	ON	ON	ESC x
Alt-x	ON	ON	ESC x
Alt+Meta-x	ON	ON	ESC x

When **modifyOtherKeys** is set, ordinary keys may be sent as escape sequences:

- When **modifyOtherKeys** is set to 1, the usual shift- and control-modifiers work as expected, but other modifiers (such as alt- and meta-modifiers) cause ordinary keys to be encoded as if they were function-keys. For example, *alt-Tab* sends CSI 2 7 ; 3 ; 9 ~ (the second parameter is "3" for *alt*, and the third parameter is the ASCII value of tab, "9").

  Ordinary characters can have non-ASCII values. *XTerm* uses the X11 libraries to obtain the character encoding used for a given *keysym* (key symbol). The X11 *keysymdef.h* header lists 2104 predefined key symbols, as well as documenting how arbitrary Unicode values are represented. The other **keysym.h* headers add 480 symbols.
- When **modifyOtherKeys** is set to 2, all of the modifiers apply. For example, *shift-Tab* sends CSI 2 7 ; 2 ; 9 ~ rather than CSI Z (the second parameter is "2" for *shift*).
- If **modifyOtherKeys** is set to 3, unmodified keys also are sent as escape sequences. For example, *space* sends CSI 2 7 ; 1 ; 3 2 ~

There are a few variations available; they can be set statically with resource values or dynamically using control sequences:

- The **formatOtherKeys** resource tells *xterm* to change the format of the escape sequences sent when **modifyOtherKeys** applies. When **modifyOtherKeys** is set to 1, for example *alt-Tab* sends CSI 9 ; 3 u (changing the order of parameters). One drawback to this format is that applications may confuse it with CSI u (restore-cursor).

  The resource value can be updated with a corresponding control sequence.
- The **modifyOtherKeys** resource can be updated via a corresponding control sequence. The control sequence accepts a single subparameter, which is interpreted as a mask of modifier bits to factor out of the parameter encoding of the escape sequence. For example CSI > 4 : 1 m factors out the shift modifier.

Some of the predefined key symbols can be automatically derived, or are duplicates. That leaves about half of the key symbols which *xterm* may encounter. Most of these are non-character key symbols which may be assigned to positions on a keyboard.

- The most common non-character key symbols are in the range 0xfd00 to 0xffff. *XTerm* maps them to the Unicode BMP private use area beginning at U+E001. These account for about a quarter (329) of the non-character symbols.
- The other non-character key symbols are vendor/platform specific. Those use codes above 0x10000000. *XTerm* maps them to the Unicode BMP private use area beginning at U+F0000.
- The remaining non-Unicode values are characters used in the DEC Technical character set. *XTerm* maps them to the Unicode BMP private use area beginning at U+EEEE.

XTerm has additional resource settings (and control sequences) to send these non-character symbols as escape sequences. The *Xutil.h* header defines some of the macros which *xterm* uses for categorizing these non-character symbols:

IsKeypadKey(*keysym*)
: is used for the numeric keypad (see **modifyKeypadKeys** and **formatKeypadKeys**).

IsPrivateKeypadKey(*keysym*)
: is unused.

IsCursorKey(*keysym*)
: is used for the cursor keys, including *Home* and *End* (see **modifyCursorKeys** and **formatCursorKeys**). *XTerm* makes a special check for the PC keyboard's editing keypad, which is not handled by the *Xutil.h* macros.

IsPFKey(*keysym*)

IsFunctionKey(*keysym*)

IsMiscFunctionKey(*keysym*)
: are used for function keys (see **modifyFunctionKeys** and **formatFunctionKeys**).

IsModifierKey(*keysym*)
: is used for modifier keys (see **modifyModifierKeys** and **formatModifierKeys**).

These macros account for 162 of the 329 non-Unicode values in the BMP.

The *xterm* FAQ sections

How can my program distinguish control-I from tab?

XTerm - "Other" Modified Keys

go into greater detail on this topic.

PC-Style Function Keys

If *xterm* does minimal translation of the function keys, it usually does this with a PC-style keyboard, so PC-style function keys result. Sun keyboards are similar to PC keyboards. Both have cursor and scrolling operations printed on the keypad, which duplicate the smaller cursor and scrolling keypads.

X does not predefine NumLock (used for VT220 keyboards) or Alt (used as an extension for the Sun/PC keyboards) as modifiers. These keys are recognized as modifiers when enabled by the **numLock** resource, or by the "DECSET 1 0 3 5" control sequence.

The cursor keys transmit the following escape sequences depending on the mode specified via the DECCKM escape sequence.

Key	Normal	Application
Cursor Up	CSI A	SS3 A
Cursor Down	CSI B	SS3 B
Cursor Right	CSI C	SS3 C
Cursor Left	CSI D	SS3 D

The home- and end-keys (unlike PageUp and other keys also on the 6-key editing keypad) are considered "cursor keys" by *xterm*. Their mode is also controlled by the DECCKM escape sequence:

Key	Normal	Application
Home	CSI H	SS3 H
End	CSI F	SS3 F

The application keypad transmits the following escape sequences depending on the mode specified via the DECKPNM and DECKPAM escape sequences. Use the NumLock key to override the application mode.

Not all keys are present on the Sun/PC keypad (e.g., PF1, Tab), but are supported by the program.

Key	Numeric	Application	Terminfo	Termcap
Space	SP	SS3 SP	-	-
Tab	TAB	SS3 I	-	-
Enter	CR	SS3 M	kent	@8
PF1	SS3 P	SS3 P	kf1	k1
PF2	SS3 Q	SS3 Q	kf2	k2
PF3	SS3 R	SS3 R	kf3	k3
PF4	SS3 S	SS3 S	kf4	k4
* (multiply)	*	SS3 j	-	-
+ (add)	+	SS3 k	-	-
, (comma)	,	SS3 l	-	-
- (minus)	−	SS3 m	-	-
. (Delete)	.	CSI 3 ~	-	-
/ (divide)	/	SS3 o	-	-
0 (Insert)	0	CSI 2 ~	-	-
1 (End)	1	SS3 F	kc1	K4
2 (DownArrow)	2	CSI B	-	-
3 (PageDown)	3	CSI 6 ~	kc3	K5
4 (LeftArrow)	4	CSI D	-	-
5 (Begin)	5	CSI E	kb2	K2
6 (RightArrow)	6	CSI C	-	-
7 (Home)	7	SS3 H	ka1	K1
8 (UpArrow)	8	CSI A	-	-
9 (PageUp)	9	CSI 5 ~	ka3	K3
= (equal)	=	SS3 X	-	-

They also provide 12 function keys, as well as a few other special-purpose keys:

Key	Escape Sequence
F1	SS3 P
F2	SS3 Q
F3	SS3 R
F4	SS3 S
F5	CSI 1 5 ~
F6	CSI 1 7 ~
F7	CSI 1 8 ~
F8	CSI 1 9 ~
F9	CSI 2 0 ~
F10	CSI 2 1 ~
F11	CSI 2 3 ~
F12	CSI 2 4 ~

Note that F1 through F4 are prefixed with SS3 , while the other keys are prefixed with CSI . Older versions of *xterm* implement different escape sequences for F1 through F4, with a CSI prefix. These can be activated by setting the **oldXtermFKeys** resource. However, since they do not correspond to any hardware terminal, they have been deprecated. (The DEC VT220 reserves F1 through F5 for local functions such as *Setup*).

Key	Escape Sequence
F1	CSI 1 1 ~
F2	CSI 1 2 ~
F3	CSI 1 3 ~
F4	CSI 1 4 ~

In normal mode, i.e., a Sun/PC keyboard when the **sunKeyboard** resource is false (and none of the other keyboard resources such as **oldXtermFKeys** resource is set), *xterm* encodes function key modifiers as parameters appended before the *final* character of the control sequence. As a special case, the SS3 sent before F1 through F4 is altered to CSI when sending a function key modifier as a parameter.

Code	Modifiers
2	Shift
3	Alt
4	Shift + Alt
5	Control
6	Shift + Control
7	Alt + Control
8	Shift + Alt + Control
9	Meta
10	Meta + Shift
11	Meta + Alt
12	Meta + Alt + Shift
13	Meta + Ctrl
14	Meta + Ctrl + Shift
15	Meta + Ctrl + Alt
16	Meta + Ctrl + Alt + Shift

For example, shift-F5 would be sent as CSI 1 5 ; 2 ~

If the **alwaysUseMods** resource is set, the Meta modifier also is recognized, making parameters 9 through 16.

The codes used for the *PC-style function keys* were inspired by a feature of the VT510, referred to in its reference manual as DECFNK. In the DECFNK scheme, codes 2-8 identify modifiers for function-keys and cursor-, editing-keypad keys. Unlike *xterm*, the VT510 limits the modifiers which can be used with cursor- and editing-keypad keys. Although the name "DECFNK" implies that it is a mode, the VT510 manual mentions it only as a feature, which (like *xterm*) interacts with the DECUDK feature. Unlike *xterm*, VT510/VT520 provide an extension to DECUDK (DECPFK and DECPAK) which apparently was the reason for the feature in those terminals, i.e., for identifying a programmable key rather than making it simple for applications to obtain modifier information. It is not described in the related VT520 manual. Neither manual was readily available at the time the feature was added to *xterm*.

On the other hand, the VT510 and VT520 reference manuals do document a related feature. That is its emulation of the SCO console, which is similar to the "xterm-sco" terminal description. The SCO console function-keys are less useful to applications developers than the approach used by *xterm* because

- the relationship between modifiers and the characters sent by function-keys is not readily apparent, and
- the scheme is not extensible, i.e., it is an *ad hoc* assignment limited to two modifiers (*shift* and *control*).

VT220-Style Function Keys

However, *xterm* is most useful as a DEC VT102 or VT220 emulator. Set the **sunKeyboard** resource to true to force a Sun/PC keyboard to act like a VT220 keyboard.

The VT102/VT220 application keypad transmits unique escape sequences in application mode, which are distinct from the cursor and scrolling keypad:

Key	Numeric	Application	VT100?
Space	SP	SS3 SP	no
Tab	TAB	SS3 I	no
Enter	CR	SS3 M	yes
PF1	SS3 P	SS3 P	yes
PF2	SS3 Q	SS3 Q	yes
PF3	SS3 R	SS3 R	yes
PF4	SS3 S	SS3 S	yes
* (multiply)	*	SS3 j	no
+ (add)	+	SS3 k	no

Key	**Numeric**	**Application**	**VT100?**
, (comma)	,	SS3 l	yes
- (minus)	−	SS3 m	yes
. (period)	.	SS3 n	yes
/ (divide)	/	SS3 o	no
0	0	SS3 p	yes
1	1	SS3 q	yes
2	2	SS3 r	yes
3	3	SS3 s	yes
4	4	SS3 t	yes
5	5	SS3 u	yes
6	6	SS3 v	yes
7	7	SS3 w	yes
8	8	SS3 x	yes
9	9	SS3 y	yes
= (equal)	=	SS3 X	no

The VT100/VT220 keypad did not have all of those keys. They were implemented in *xterm* in X11R1 (1987), defining a mapping of all X11 keys which might be provided on a keypad. For instance, a Sun4/II type-4 keyboard provided "=" (equal), "/" (divide), and "*" (multiply).

While the VT420 provided the same keypad, the VT520 used a PC-keyboard. Because that keyboard's keypad lacks the "," (comma), it was not possible to use EDT's delete-character function with the keypad. *XTerm* solves that problem for the VT220-keyboard configuration by mapping

Ctrl + to , and
Ctrl − to −

The VT220 provides a 6-key editing keypad, which is analogous to that on the PC keyboard. It is not affected by DECCKM or DECKPNM/DECKPAM:

Key	**Normal**	**Application**
Insert	CSI 2 ~	CSI 2 ~
Delete	CSI 3 ~	CSI 3 ~
Home	CSI 1 ~	CSI 1 ~
End	CSI 4 ~	CSI 4 ~
PageUp	CSI 5 ~	CSI 5 ~
PageDown	CSI 6 ~	CSI 6 ~

The VT220 provides 8 additional function keys. With a Sun/PC keyboard, access these keys by Control/F1 for F13, etc.

Key	**Escape Sequence**
F13	CSI 2 5 ~
F14	CSI 2 6 ~
F15	CSI 2 8 ~
F16	CSI 2 9 ~
F17	CSI 3 1 ~
F18	CSI 3 2 ~
F19	CSI 3 3 ~
F20	CSI 3 4 ~

VT52-Style Function Keys

A VT52 does not have function keys, but it does have a numeric keypad and cursor keys. They differ from the other emulations by the prefix. Also, the cursor keys do not change:

Key	Normal/Application
Cursor Up	ESC A
Cursor Down	ESC B
Cursor Right	ESC C
Cursor Left	ESC D

The keypad is similar:

Key	Numeric	Application	VT52?
Space	SP	ESC ? SP	no
Tab	TAB	ESC ? I	no
Enter	CR	ESC ? M	no
PF1	ESC P	ESC P	yes
PF2	ESC Q	ESC Q	yes
PF3	ESC R	ESC R	yes
PF4	ESC S	ESC S	no
* (multiply)	*	ESC ? j	no
+ (add)	+	ESC ? k	no
, (comma)	,	ESC ? l	no
- (minus)	−	ESC ? m	no
. (period)	.	ESC ? n	yes
/ (divide)	/	ESC ? o	no
0	0	ESC ? p	yes
1	1	ESC ? q	yes
2	2	ESC ? r	yes
3	3	ESC ? s	yes
4	4	ESC ? t	yes
5	5	ESC ? u	yes
6	6	ESC ? v	yes
7	7	ESC ? w	yes
8	8	ESC ? x	yes
9	9	ESC ? y	yes
= (equal)	=	ESC ? X	no

Sun-Style Function Keys

The *xterm* program provides support for Sun keyboards more directly, by a menu toggle that causes it to send Sun-style function key codes rather than VT220. Note, however, that the *sun* and *VT100* emulations are not really compatible. For example, their wrap-margin behavior differs.

Only function keys are altered; keypad and cursor keys are the same. The emulation responds identically. See the xterm-sun terminfo entry for details.

HP-Style Function Keys

Similarly, *xterm* can be compiled to support HP keyboards. See the xterm-hp terminfo entry for details.

Non-Function Keys

On a DEC terminal keyboard, some of the keys which one would expect to see labeled as function keys had special names. The keys actually send character sequences as if they were the expected function keys, but the special names are used in documentation. Because other keyboards may use those names, *xterm* maps the X key symbols which have the corresponding names into the character sequences which the original DEC keyboard would send.

These mappings are used for the DEC (VT220) and other keyboards:

Label	DEC	SUN	HP	SCO
Up	SS3 A	SS3 A	ESC A	CSI A
Down	SS3 B	SS3 B	ESC B	CSI B
Right	SS3 C	SS3 C	ESC C	CSI C
Left	SS3 D	SS3 D	ESC D	CSI D
Clear	-	-	ESC J	-
Find	CSI 1 ~	CSI 1 z	ESC h	-
Insert	CSI 2 ~	CSI 2 z	ESC Q	CSI L
Delete	CSI 3 ~	CSI 3 z	ESC P	-
Keypad Insert	CSI 2 ~	CSI 2 z	ESC Q	CSI L
Keypad Delete	CSI 3 ~	CSI 3 z	ESC P	-
Remove	CSI 3 ~	CSI 3 z	ESC P	-
Select	CSI 4 ~	CSI 4 z	ESC F	-
Prior	CSI 5 ~	CSI 2 1 6 z	ESC T	CSI I
Next	CSI 6 ~	CSI 2 2 2 z	ESC S	CSI G
Help	CSI 2 8 ~	CSI 1 9 6 z	-	-
Menu	CSI 2 9 ~	CSI 1 9 7 z	-	-
Home	-	CSI 2 1 4 z	ESC h	CSI H
End	-	CSI 2 2 0 z	ESC F	CSI F
Begin	-	CSI 2 1 8 z	-	CSI E

The Alternate Screen Buffer

XTerm maintains two screen buffers. The Normal Screen Buffer allows you to scroll back to view saved lines of output up to the maximum set by the **saveLines** resource. The *Alternate Screen Buffer* is exactly as large as the display, contains no additional saved lines. When the *Alternate Screen Buffer* is active, you cannot scroll back to view saved lines. *XTerm* provides control sequences and menu entries for switching between the two.

Most full-screen applications use terminfo or termcap to obtain strings used to start/stop full-screen mode, i.e., *smcup* and *rmcup* for terminfo, or the corresponding *ti* and *te* for termcap. The **titeInhibit** resource removes the *ti* and *te* strings from the TERMCAP string which is set in the environment for some platforms. That is not done when *xterm* is built with terminfo libraries because terminfo does not provide the whole text of the termcap data in one piece. It would not work for terminfo anyway, since terminfo data is not passed in environment variables; setting an environment variable in this manner would have no effect on the application's ability to switch between *Normal* and *Alternate Screen* buffers. Instead, the newer private mode controls (such as 1 0 4 9) for switching between *Normal* and *Alternate Screen* buffers simply disable the switching. They add other features such as clearing the display for the same reason: to make the details of switching independent of the application that requests the switch.

Bracketed Paste Mode

When bracketed paste mode is set, pasted text is bracketed with control sequences so that the program can differentiate pasted text from typed-in text. When bracketed paste mode is set, the program will receive:
ESC [2 0 0 ~,
followed by the pasted text, followed by
ESC [2 0 1 ~.
For background and discussion, see the FAQ:

XTerm - bracketed-paste

Readline Modes

Several modes provide support for mouse button events in *readline*. Bracketed paste is one of these *readline* modes, but is used more widely.

Some assumptions (particular mouse buttons) and limitations (the mouse is clicked on the current row on the screen) apply:

2 0 0 1

If mouse button 1 is used to end or extend a selection (the **select−end** action), and if the cursor position is on the same row as the mouse-click, send left/right cursor control sequences to the host to adjust the cursor position to match the mouse click.

2 0 0 2

When pasting text (the **insert−selection** action which is normally bound to mouse button 2), if mouse protocol is not enabled, and if the cursor position is on the same row as the mouse-click, send left/right cursor control sequences to the host to adjust the cursor position to match the mouse click.

2 0 0 3

If mouse button 3 is double-clicked when ending or extending a selection, (the **select−end** action), and if the cursor position is on the same line as the mouse-click:

- Send left/right cursor control sequences to the host to adjust the cursor position to match the mouse click.
- In addition to the same *row*, the selection may be part of a wrapped line as in other *xterm* selections (see the **Selection Functions** section in the manual page).
- After adjusting the cursor position, *xterm* sends erase-characters (one for each character in the selection) to tell the host to delete the selected text.

2 0 0 5

When writing a selection to the host (i.e., pasting text), escape each character with the *literal-next* (Ctrl-V) character.

2 0 0 6

Normally when *xterm* writes selections to the host, it translates newlines to carriage returns. This mode disables the translation, passing newlines literally.

Title Modes

The window- and icon-labels can be set or queried using control sequences. As a VT220-emulator, *xterm* "should" limit the character encoding for the corresponding strings to ISO-8859-1. Indeed, it used to be the case (and was documented) that window titles had to be ISO-8859-1. This is no longer the case. However, there are many applications which still assume that titles are set using ISO-8859-1. So that is the default behavior.

If *xterm* is running with UTF-8 encoding, it is possible to use window- and icon-labels encoded using UTF-8. That is because the underlying X libraries (and many, but not all) window managers support this feature.

The **utf8Title** X resource setting tells *xterm* to disable a reconversion of the title string back to ISO-8859-1, allowing the title strings to be interpreted as UTF-8. The same feature can be enabled using the title mode control sequence described in this summary.

Separate from the ability to set the titles, *xterm* provides the ability to query the titles, returning them either in ISO-8859-1 or UTF-8. This choice is available only while *xterm* is using UTF-8 encoding.

Finally, the characters sent to, or returned by a title control are less constrained than the rest of the control sequences. To make them more manageable (and constrained), for use in shell scripts, *xterm* has an optional feature which decodes the string from hexadecimal (for setting titles) or for encoding the title into hexadecimal when querying the value.

Mouse Tracking

The VT widget can be set to send the mouse position and other information on button presses. These modes are typically used by editors and other full-screen applications that want to make use of the mouse.

There are two sets of mutually exclusive modes:

- mouse protocol
- protocol encoding

The mouse protocols include DEC Locator mode, enabled by the DECELR CSI P_s ; P_s ' z control sequence, and is not described here (control sequences are summarized above). The remaining five modes of the mouse protocols are each enabled (or disabled) by a different parameter in the "DECSET CSI ? P_m h" or "DECRST CSI ? P_m l" control sequence.

Manifest constants for the parameter values are defined in *xcharmouse.h* as follows:

```
#define SET_X10_MOUSE               9
#define SET_VT200_MOUSE             1000
#define SET_VT200_HIGHLIGHT_MOUSE   1001
#define SET_BTN_EVENT_MOUSE         1002
#define SET_ANY_EVENT_MOUSE         1003

#define SET_FOCUS_EVENT_MOUSE       1004

#define SET_ALTERNATE_SCROLL        1007

#define SET_EXT_MODE_MOUSE          1005
#define SET_SGR_EXT_MODE_MOUSE      1006
#define SET_URXVT_EXT_MODE_MOUSE    1015
#define SET_PIXEL_POSITION_MOUSE    1016
```

The motion reporting modes are strictly *xterm* extensions, and are not part of any standard, though they are analogous to the DEC VT200 DECELR locator reports.

Normally, parameters (such as pointer position and button number) for all mouse tracking escape sequences generated by *xterm* encode numeric parameters in a single character as *value*+32. For example, ! specifies the value 1. The upper left character position on the terminal is denoted as 1,1. This scheme dates back to X10, though the normal mouse-tracking (from X11) is more elaborate.

X10 compatibility mode

X10 compatibility mode sends an escape sequence only on button press, encoding the location and the mouse button pressed. It is enabled by specifying parameter 9 to DECSET. On button press, *xterm* sends CSI M $C_bC_xC_y$ (6 characters).

- C_b is *button*−1, where *button* is 1, 2 or 3.
- C_x and C_y are the *x* and *y* coordinates of the mouse when the button was pressed.

Normal tracking mode

Normal tracking mode sends an escape sequence on both button press and release. Modifier key (shift, ctrl, meta) information is also sent. It is enabled by specifying parameter 1000 to DECSET. On button press or release, *xterm* sends CSI M $C_bC_xC_y$.

- The low two bits of C_b encode button information:

  0=MB1 pressed,
  1=MB2 pressed,
  2=MB3 pressed, and
  3=release.

- The next three bits encode the modifiers which were down when the button was pressed and are added together:

  4=Shift,
  8=Meta, and
  16=Control.

  The *shift* and *control* modifiers are normally irrelevant because *xterm* uses the *control* modifier with mouse for popup menus, and the *shift* modifier is used in the default translations for button events.

  There is no predefined *meta* modifier. *XTerm* checks first if the keysyms listed in the predefined modifiers include **Meta_L** or **Meta_R**. If found, *xterm* uses that modifier for *meta*. Next, it tries **Alt_L** or **Alt_R**. If none of those are found, *xterm* uses the *mod1* modifier, This is not necessarily the "Meta" key according to **xmodmap**(1).

- C_x and C_y are the x and y coordinates of the mouse event, encoded as in X10 mode.

Wheel mice

Wheel mice may return buttons 4 and 5. Those buttons are represented by the same event codes as buttons 1 and 2 respectively, except that 64 is added to the event code. Release events for the wheel buttons are not reported.

By default, the wheel mouse events (buttons 4 and 5) are translated to **scroll−back** and **scroll−forw** actions, respectively. Those actions normally scroll the whole window, as if the scrollbar was used.

However if *Alternate Scroll* mode is set, then cursor up/down controls are sent when the terminal is displaying the *Alternate Screen Buffer*. The initial state of *Alternate Scroll* mode is set using the **alternateScroll** resource.

Other buttons

Some wheel mice can send additional button events, e.g., by tilting the scroll wheel left and right.

Additional buttons are encoded like the wheel mice,

- by adding 64 (for buttons 6 and 7), or
- by adding 128 (for buttons 8 through 11).

Past button 11, the encoding is ambiguous because the same code may correspond to different button/modifier combinations.

It is not possible to use these buttons (6-11) in *xterm*'s **translations** resource because their names are not in the X Toolkit's symbol table. However, applications can check for the reports, e.g., button 7 (left) and button 6 (right) with a Logitech mouse.

Highlight tracking

Mouse highlight tracking notifies a program of a button press, receives a range of lines from the program, highlights the region covered by the mouse within that range until button release, and then sends the program the release coordinates. It is enabled by specifying parameter 1001 to DECSET. Highlighting is performed only for button 1, though other button events can be received.

Warning: this mode requires a cooperating program, else *xterm* will hang.

On button press, the same information as for normal tracking is generated; *xterm* then waits for the program to send mouse tracking information. *All X events are ignored until the proper escape sequence is received from the pty:*
$\boxed{\text{CSI}}\,P_s\boxed{;}\,P_s\boxed{;}\,P_s\boxed{;}\,P_s\boxed{;}\,P_s\boxed{\text{T}}$

The parameters are *func, startx, starty, firstrow,* and *lastrow*:

- *func* is non-zero to initiate highlight tracking and zero to abort.
- *startx* and *starty* give the starting x and y location for the highlighted region.
- The ending location tracks the mouse, but will never be above row *firstrow* and will always be above row *lastrow.* (The top of the screen is row 1.)

When the button is released, *xterm* reports the ending position one of two ways:

- if the start and end coordinates are the same locations:

  $\boxed{\text{CSI}}\boxed{\text{t}}\,C_x C_y$
- otherwise:

  $\boxed{\text{CSI}}\boxed{\text{T}}\,C_x C_y C_x C_y C_x C_y$

The parameters are *startx, starty, endx, endy, mousex,* and *mousey*:

- *startx, starty, endx,* and *endy* give the starting and ending character positions of the region.
- *mousex* and *mousey* give the location of the mouse at button up, which may not be over a character.

Button-event tracking

Button-event tracking is essentially the same as normal tracking, but *xterm* also reports button-motion events. Motion events are reported only if the mouse pointer has moved to a different character cell. It is enabled by specifying parameter 1002 to DECSET. On button press or release, *xterm* sends the same codes used by normal tracking mode.

- On button-motion events, *xterm* adds 32 to the event code (the third character, C_b).
- The other bits of the event code specify button and modifier keys as in normal mode. For example, motion into cell x,y with button 1 down is reported as

  CSI M @ $C_x C_y$

  (@ = 32 + 0 (button 1) + 32 (motion indicator)). Similarly, motion with button 3 down is reported as

  CSI M B $C_x C_y$

  (B = 32 + 2 (button 3) + 32 (motion indicator)).

Any-event tracking

Any-event mode is the same as button-event mode, except that all motion events are reported, even if no mouse button is down. It is enabled by specifying 1003 to DECSET.

FocusIn/FocusOut

FocusIn/FocusOut can be combined with any of the mouse events since it uses a different protocol. When set, it causes *xterm* to send CSI I when the terminal gains focus, and CSI O when it loses focus.

Extended coordinates

The original X10 mouse protocol limits the C_x and C_y ordinates to 223 (=255 - 32). *XTerm* supports more than one scheme for extending this range, by changing the protocol encoding:

UTF-8 (1005) This enables UTF-8 encoding for C_x and C_y under all tracking modes, expanding the maximum encodable position from 223 to 2015. For positions less than 95, the resulting output is identical under both modes. Under extended mouse mode, positions greater than 95 generate “extra” bytes which will confuse applications which do not treat their input as a UTF-8 stream. Likewise, C_b will be UTF-8 encoded, to reduce confusion with wheel mouse events.

Under normal mouse mode, positions outside (160,94) result in byte pairs which can be interpreted as a single UTF-8 character; applications which do treat their input as UTF-8 will almost certainly be confused unless extended mouse mode is active.

This scheme has the drawback that the encoded coordinates will not pass through **luit**(1) unchanged, e.g., for locales using non-UTF-8 encoding.

SGR (1006) The normal mouse response is altered to use

- CSI < followed by semicolon-separated
- encoded button value,
- P_x and P_y ordinates and
- a final character which is M for button press and m for button release.

The encoded button value in this case does not add 32 since that was useful only in the X10 scheme for ensuring that the byte containing the button value is a printable code.

- The modifiers are encoded in the same way.
- A different final character is used for button release to resolve the X10 ambiguity regarding which button was released.

The highlight tracking responses are also modified to an SGR-like format, using the same SGR-style scheme and button-encodings.

URXVT (1015) The normal mouse response is altered to use

- CSI followed by semicolon-separated
- encoded button value,
- the P_x and P_y ordinates and final character M .

This uses the same button encoding as X10, but printing it as a decimal integer rather than as a single byte.

However, CSI M can be mistaken for DL (delete lines), while the highlight tracking CSI T can be mistaken for SD (scroll down), and the Window manipulation controls. For these reasons, the 1015 control is not recommended; it is not an improvement over 1006.

SGR-Pixels (1016)

Use the same mouse response format as the 1006 control, but report position in *pixels* rather than character *cells*.

Graphics

Sixel Graphics

If *xterm* is configured as VT240, VT241, VT330, VT340 or VT382 using the **decTerminalID** or **decGraphicsID** resource, it supports Sixel Graphics controls, a paletted bitmap graphics system using sets of six vertical pixels as the basic element.

CSI P_s c Send Device Attributes (Primary DA), DEC graphics terminals, *xterm*. *xterm* responds to Send Device Attributes (Primary DA) with these additional codes:

P_s = 4 → Sixel graphics.

CSI ? P_m h Set Mode (with corresponding Reset Mode CSI ? P_m l):

P_s = 8 0 → Sixel Display Mode (DECSDM), VT330, VT340, VT382.

P_s = 1 0 7 0 → use private color registers for each graphic, *xterm*.

P_s = 8 4 5 2 → Sixel scrolling leaves cursor to right of graphic, RLogin, *xterm*.

DCS P_a ; P_b ; P_h q P_s..P_s ST

Send SIXEL image, DEC graphics terminals, VT330, VT340, VT382. See:

VT330/VT340 Programmer Reference Manual Volume 2:

Graphics Programming

Chapter 14 Graphics Programming

The sixel data device control string has three positional parameters, following the q with sixel data.

P_a → pixel aspect ratio

P_b → background color option

P_h → horizontal grid size (ignored).

P_s → sixel data

ReGIS Graphics

If *xterm* is configured as VT125, VT240, VT241, VT330 or VT340 using the **decTerminalID** or **decGraphicsID** resource, it supports Remote Graphic Instruction Set, a graphics description language.

CSI P_s c — Send Device Attributes (Primary DA), DEC graphics terminals, *xterm*. *xterm* responds to Send Device Attributes (Primary DA) with these additional codes:

P_s = 3 → ReGIS graphics.

CSI ? P_m h — Set Mode, *xterm*. *xterm* has these additional private Set Mode values:

P_s = 1 0 7 0 → use private color registers for each graphic.

DCS P_m p P_r..P_r ST

Enter or exit ReGIS, VT300, *xterm*. See:

VT330/VT340 Programmer Reference Manual Volume 2:
Graphics Programming
Chapter 1 Introduction to ReGIS

The ReGIS data device control string has one positional parameter with four possible values:

P_m = 0 → resume command, use fullscreen mode.
P_m = 1 → start new command, use fullscreen mode.
P_m = 2 → resume command, use command display mode.
P_m = 3 → start new command, use command display mode.

A few of the VT330/VT340 private modes conflict with *xterm*. Codes 4 0 to 4 7 were first used by xterm in X10R4 (December 1986). While X11R1 *xterm* dropped codes 4 1 and 4 2, the remaining ones are still used. The VT330/VT340 introduced in April 1987 uses 4 4 to 4 7 for color graphics printing controls. When configured for ReGIS, *xterm* uses the VT330/VT340 interpretation of these private modes.

Non-VT100 Modes

Tektronix 4014 Mode

Most of these sequences are standard Tektronix 4014 control sequences. Graph mode supports the 12-bit addressing of the Tektronix 4014. The major features missing are the write-through and defocused modes. This document does not describe the commands used in the various Tektronix plotting modes but does describe the commands to switch modes.

Some of the sequences are specific to *xterm*. The Tektronix emulation was added in X10R4 (1986). The VT240, introduced two years earlier, also supported Tektronix 4010/4014. Unlike *xterm*, the VT240 documentation implies (there is an obvious error in section 6.9 “Entering and Exiting 4010/4014 Mode”) that exiting back to ANSI mode is done by resetting private mode 3 8 (DECTEK) rather than ESC ETX. A real Tektronix 4014 would not respond to either.

BEL — Bell (Ctrl-G).

BS — Backspace (Ctrl-H).

TAB — Horizontal Tab (Ctrl-I).

LF — Line Feed or New Line (Ctrl-J).

VT — Cursor up (Ctrl-K).

FF — Form Feed or New Page (Ctrl-L).

CR — Carriage Return (Ctrl-M).

ESC ETX — Switch to VT100 Mode (ESC Ctrl-C).

ESC ENQ — Return Terminal Status (ESC Ctrl-E).

ESC FF — PAGE (Clear Screen) (ESC Ctrl-L).

ESC SO Begin 4015 APL mode (ESC Ctrl-N). This is ignored by *xterm*.

ESC SI End 4015 APL mode (ESC Ctrl-O). This is ignored by *xterm*.

ESC ETB COPY (Save Tektronix Codes to file COPY*yyyy-mm-dd.hh:mm:ss*).
ETB (end transmission block) is the same as Ctrl-W.

ESC CAN Bypass Condition (ESC Ctrl-X).

ESC SUB GIN mode (ESC Ctrl-Z).

ESC FS Special Point Plot Mode (ESC Ctrl-\).

ESC 8 Select Large Character Set.

ESC 9 Select #2 Character Set.

ESC : Select #3 Character Set.

ESC ; Select Small Character Set.

OSC P_s ; P_t BEL
Set Text Parameters of VT window.
P_s = 0 → Change Icon Name and Window Title to P_t.
P_s = 1 → Change Icon Name to P_t.
P_s = 2 → Change Window Title to P_t.
P_s = 4 6 → Change Log File to P_t. This is normally disabled by a compile-time option.

ESC ` Normal Z Axis and Normal (solid) Vectors.

ESC a Normal Z Axis and Dotted Line Vectors.

ESC b Normal Z Axis and Dot-Dashed Vectors.

ESC c Normal Z Axis and Short-Dashed Vectors.

ESC d Normal Z Axis and Long-Dashed Vectors.

ESC h Defocused Z Axis and Normal (solid) Vectors.

ESC i Defocused Z Axis and Dotted Line Vectors.

ESC j Defocused Z Axis and Dot-Dashed Vectors.

ESC k Defocused Z Axis and Short-Dashed Vectors.

ESC l Defocused Z Axis and Long-Dashed Vectors.

ESC p Write-Thru Mode and Normal (solid) Vectors.

ESC q Write-Thru Mode and Dotted Line Vectors.

ESC r Write-Thru Mode and Dot-Dashed Vectors.

ESC s Write-Thru Mode and Short-Dashed Vectors.

ESC t Write-Thru Mode and Long-Dashed Vectors.

FS Point Plot Mode (Ctrl-\).

GS Graph Mode (Ctrl-]).

RS Incremental Plot Mode (Ctrl-^).

US Alpha Mode (Ctrl-_).

VT52 Mode

Parameters for cursor movement are at the end of the ESC Y escape sequence. Each ordinate is encoded in a single character as *value*+32. For example, ! is 1. The screen coordinate system is 0-based.

ESC < Exit VT52 mode (Enter VT100 mode).

ESC = Enter alternate keypad mode.

ESC > Exit alternate keypad mode.

ESC A Cursor up.

ESC B Cursor down.

ESC C Cursor right.

ESC D Cursor left.

ESC F Enter graphics mode.

ESC G Exit graphics mode.

ESC H Move the cursor to the home position.

ESC I Reverse line feed.

ESC J Erase from the cursor to the end of the screen.

ESC K Erase from the cursor to the end of the line.

ESC Y P_s P_s Move the cursor to given row and column.

ESC Z Identify.

→ ESC / Z ("I am a VT52 emulated by VT100.").

or

→ ESC / K ("I am a VT52.").

depending on whether *xterm* is started as a VT52 by setting the **decTerminalID** resource to "52" or not.

Further reading

Technical manuals

Manuals for *hardware* terminals are more readily available than similarly-detailed documentation for terminal *emulators* such as *aixterm*, *shelltool*, *dtterm*.

However long, the technical manuals have problems:

- DEC's manuals did not provide a comprehensive comparison of the features in different model.

  Host Interface Functions Checklist by Peter Sichel (January 12, 1994) is helpful. This spreadsheet is useful for noting which model introduced a given feature (although there are a few apparent errors such as the DECRQSS feature cited for VT320 whereas the technical manual omits it).
- Sometimes the manuals disagree. For example, DEC's standard document (DEC STD 070) for terminals says that DECSCL performs a *soft* reset (DECSTR), while the VT420 manual says it does a *hard* reset (RIS).
- Sometimes the manuals are simply incorrect. For example, testing a DEC VT420 in 1996 showed that the documented code for a valid or invalid response to DECRQSS was reversed.

  The VT420 test results were incorporated into the *vttest* program. At the time, DEC STD 070 was not available, but it also agrees with *vttest*. Later, documentation for the DEC VT525 was shown to have the same flaw.
- The VT330/VT340 reference manual for graphics programming documents sixel graphics in some detail in chapter 14. Overlooked in the first edition, the second edition mentions *Sixel Scrolling*. The VT382 Kanji and Thai manuals provide less information, about sixel graphics, but do mention DECSDM. They differ in their comment about the private mode DECSDM (CSI ? 8 0 h), which each manual agrees should *set* the Sixel Scrolling feature. The VT330/VT340 graphics programming manual (second edition, March 1988) says

  > When sixel display mode is set, the *Sixel Scrolling* feature is enabled.
  > When sixel display mode is reset, the *Sixel Scrolling* feature is disabled.

while the VT382 Kanji manual (page 6-6, undated) says

Disable sixel scroll

and the VT382 Thai manual (page C-30, August 1989) says

No Sixel scrolling

The standard (DEC STD 070) in chapter 9 (August 3, 1990) states on page 17 that video devices will scroll when advancing the Sixel active position past the bottom margin, but on page 19, in the section on deviations, states that VT125 and VT240 did not scroll in this situation. The standard does not mention VT330/VT340 or VT382. Nor does it document DECSDM.

- Not all details are clear even in DEC STD 070 (which is more than twice the length of the VT520 programmer's reference manual, and almost three times longer than the VT420 reference manual). However, as an internal standards document, DEC STD 070 is more likely to describe the actual behavior of DEC's terminals than the more polished user's guides.

That said, here are technical manuals which have been used in developing *xterm*. Not all were available initially. In August 1996 for instance, the technical references were limited to EK-VT220-HR-002 and EK-VT420-UG.002. Shortly after, Richard Shuford sent a copy of EK-VT3XX-TP-001. Still later (beginning in 2003), Paul Williams' vt100.net site provided EK-VT102-UG-003, EK-VT220-RM-002, EK-VT420-RM-002, EK-VT520-RM A01, EK-VT100-TM-003, and EK-VT102-UG-003. In addition, several documents were found on the bitsavers site.

- *DECscope User's Manual*.
  Digital Equipment Corporation (EK-VT5X-OP-001 1975).
- *VT100 Series Video Terminal Technical Manual*.
  Digital Equipment Corporation (EK-VT100-TM-003, July 1982).
- *VT100 User Guide*.
  Digital Equipment Corporation (EK-VT100-UG-003, June 1981).
- *VT102 User Guide*.
  Digital Equipment Corporation (EK-VT102-UG-003, February 1982).
- *VT220 Programmer Pocket Guide*.
  Digital Equipment Corporation (EK-VT220-HR-002, July 1984).
- *VT220 Programmer Reference Manual*.
  Digital Equipment Corporation (EK-VT220-RM-002, August 1984).
- *VT240 Programmer Reference Manual*.
  Digital Equipment Corporation (EK-VT240-RM-002, October 1984).
- *VT330/VT340 Programmer Reference Manual*
  Volume 1: Text Programming.
  Digital Equipment Corporation (EK-VT3XX-TP-001, March 1987).
- *VT330/VT340 Programmer Reference Manual*
  Volume 2: Graphics Programming.
  Digital Equipment Corporation (EK-VT3XX-GP-001, March 1987).
- *VT330/VT340 Programmer Reference Manual*
  Volume 2: Graphics Programming.
  Digital Equipment Corporation (EK-VT3XX-GP-002, May 1988).
- *VT382 Kanji Display Terminal*
  Programmer Reference Manual.
  Digital Equipment Corporation (EK-VT382-RM-001, undated).
- *VT382 Thai Display Terminal*
  Installing and Using Manual.
  Digital Equipment Corporation (EK-VT38T-UG-001, August 1989).

- *Installing and Using*
  The VT420 Video Terminal
  (North American Model).
  Digital Equipment Corporation (EK-VT420-UG.002, February 1990).
- *VT420 Programmer Reference Manual*.
  Digital Equipment Corporation (EK-VT420-RM-002, February 1992).
- *VT510 Video Terminal*
  Programmer Information.
  Digital Equipment Corporation (EK-VT510-RM B01, November 1993).
- *VT520/VT525 Video Terminal*
  Programmer Information.
  Digital Equipment Corporation (EK-VT520-RM A01, July 1994).
- *Digital ANSI-Compliant Printing Protocol*
  Level 2 Programming Reference Manual
  Digital Equipment Corporation (EK-PPLV2-PM B01, August 1994).
- *Disk Operating System*
  DOS 2.00
  Microsoft, Inc.
  First edition, January 1983.
- *4014 and 4014-1 Computer Display Terminal*
  User's Manual.
  Tektronix, Inc. (070-1647-00, November 1979).

Standards

The DEC terminal family (VT100 through VT525) is upward-compatible, using standards plus *extensions*, e.g., "private modes". Not all commonly-used features are standard. For example, scrolling regions are not found in ECMA-48. On the other hand, ECMA-48 was not intended to be all-encompassing. Quoting from the second edition:

```
Full conformance to a standard means that all its requirements are met.
For such conformance to be unique the standard must contain no options.
This is typically the case for hardware standards, for instance Standard
ECMA-10 for data interchange on punched tapes.

This Standard ECMA-48 is of a different nature and as a result, it is only
practicable to envisage limited conformance to it, as defined hereunder.

This Standard addresses a whole class of devices which can vary greatly
from each other depending on the application for which a device has been
specifically designed.  Obviously, a product which implements all facili-
ties described in this standard - thus being in "full conformance" with it
- whilst theoretically possible, would be technically and economically un-
thinkable.
```

Again, it is possible to find discrepancies in the standards:

- The printed ECMA-48 5th edition (1991) and the first PDF produced for that edition (April 1998) state that SD (scroll down) ends with 05/14, i.e., ^, which disagrees with DEC's VT420 hardware implementation and DEC's manuals which use 05/04 T. (A few other terminals such as AT&T 5620 and IBM 5151 also used 05/04, but the documentation and dates are lacking).

  ECMA created a new PDF in April 2003 which changed that detail to use T, and later in 2008 provided PDFs of the earlier editions which used T.

- The first edition of ECMA-48 has not been available, to compare. As of September 2021, ECMA's website provides a copy of ECMA-**46** in its place.

  Earlier versions of ISO 6429 have never been available. The first three editions of ISO 6429 were issued in 1983, 1988, and 1992.

- *ANSI X3.64-1979* does not list color as a feature of the SGR sequence (page 49).

  In Appendix A, it mentions ECMA-48:

  > (8) This document represents a coordinated effort to develop a single technical standard in the United States and Europe (see ECMA-48 standard entitled *Additional Controls for Character Imaging Input/Output Devices*).

  Appendix H clarifies the relationship between these documents somewhat though it confuses the first two editions of ECMA-48. The typo for "work" versus "owkr" appears in the original document:

  > ANSI X3.64-1979, and ECMA-48, *Additional Controls for Character-Imaging I/O Devices*, were developed in parallel, with close liaison. ISO DP 6429, *Additional Control Functions for Character-Imaging Devices*, was developed as a synthesis of X3.64 and ECMA-48. During this process, some control functions as well as additional selective parameters were added. Except for point 1 below, X3.64 is a subset of ISO 6429. Although the two standards use different language, the intent is that the subset is technically identical. X3.64 was balloted and forwarded prior to the final resolution of ISO 6429 and does not incorporate the *owkr* of IS0/TC97/SC2 in completing ISO 6429. Revision of X3.64 will attempt to incorporate those elements and assumptions of X3.64.

  ANSI X3.64 goes on to say that the SGR codes 8, 30-47 are in ISO 6429. It includes 38 and 39, but omits 48 and 49. At the time, ISO 6429's first edition was still four years in the future. The writer probably was referring to the ongoing process of making ECMA-48 second edition into the ISO standard.

- The VT320, VT420, VT520 manuals claim that DECSCL does a hard reset (RIS).

  Both the VT220 manual and DEC STD 070 (which documents levels 1-4 in detail) state that it is a soft reset, e.g., DECSTR.

Here are the relevant standards:

- *Additional Controls for Use with American National Standard Code for Information Interchange, ANSI X3.64-1979*
  FIPS Publication 86. July 18, 1979.
  American National Standards Institute, Inc.
- *ECMA-35: Character Code Structure and Extension Techniques*
  (6th Edition, December 1994).
- *ECMA-43: 8-bit Coded Character Set Structure and Rules*
  (3rd Edition, December 1991).
- *ECMA-48: Control Functions for Coded Character Sets*
  (5th Edition, June 1991).
- *DEC STD 070 Video Systems Reference Manual*.
  Digital Equipment Corporation (A-MN-ELSM070-00-0000 Rev H, December 3, 1991).

Miscellaneous

A few hardware terminals survived into the 1990s only as terminal emulators. Documentation for these and other terminal emulators which have influenced *xterm* are generally available only in less-accessible and less-detailed manual pages.

- *XTerm* supports control sequences for manipulating its *window* which were implemented by Sun's *shelltool* program. This was part of SunView (SunOS 3.0, 1986). The change-notes for *xterm*'s *resize* program in X10.4

(1986) mention its use of these “Sun tty emulation escape sequences” for resizing the window. The X10.4 *xterm* program recognized these sequences for resizing the terminal, except for the iconify/deiconify pair. SunView also introduced the SIGWINCH signal, used by the X10.4 *xterm* and mentioned in its *CHANGES* file:

> The window size is passed to the operating system via TIOCSWINSZ (4.3) or TIOCSSIZE (sun). A SIGWINCH signal is sent if the vtXXX window is resized.

While support for the Sun control-sequences remained in *resize*, the next release of *xterm* (X11R1 in 1987) omitted the code for interpreting them.

Later, the SunView program was adapted for the *OPEN LOOK* environment introduced 1988-1990.

Still later, in 1995, *OPEN LOOK* was abandoned in favor of *CDE*. The *CDE* terminal emulator *dtterm* implemented those controls, with a couple of additions.

Starting in July 1996, *xterm* re-implemented those control sequences (based on the *dtterm* manual pages) and further extended the group of window controls.

There were two sets of controls (CSI P_s [; P_m ; P_m] t, and OSC P_s *text* ST) implemented by *shelltool*, documented in appendix E of both *PHIGS Programming Manual* (1992), and the unpublished *X Window System User’s Guide (OPEN LOOK Edition)* (1995). The *CDE* program kept those, and added a few new ones.

Code	Sun	CDE	XTerm	Description
CSI 1 t	yes	yes	yes	de-iconify
CSI 2 t	yes	yes	yes	iconify
CSI 3 t	yes	yes	yes	move window to pixel-position
CSI 4 t	yes	yes	yes	resize window in pixels
CSI 5 t	yes	yes	yes	raise window to front of stack
CSI 6 t	yes	yes	yes	raise window to back of stack
CSI 7 t	yes	yes	yes	refresh window
CSI 8 t	yes	yes	yes	resize window in chars
CSI 9 t	-	-	yes	maximize/unmaximize window
CSI 1 0 t	-	-	yes	to/from full-screen
CSI 1 1 t	yes	yes	yes	report if window is iconified
CSI 1 2 t	-	-	-	-
CSI 1 3 t	yes	yes	yes	report window position
CSI 1 4 t	yes	yes	yes	report window size in pixels
CSI 1 5 t	-	-	yes	report screen size in pixels
CSI 1 6 t	-	-	yes	report character cell in pixels
CSI 1 7 t	-	-	-	-
CSI 1 8 t	yes	yes	yes	report window size in chars
CSI 1 9 t	-	-	yes	report screen size in chars
CSI 2 0 t	-	yes	yes	report icon label
CSI 2 1 t	-	yes	yes	report window title
CSI 2 2 t	-	-	yes	save window/icon title
CSI 2 3 t	-	-	yes	restore window/icon title
CSI 2 4 t	-	-	yes	resize window (DECSLPP)
OSC 0 ST	-	yes	yes	set window and icon title
OSC 1 ST	-	yes	yes	set icon label
OSC 2 ST	-	yes	yes	set window title
OSC 3 ST	-	n/a	yes	set X server property
OSC I ST	yes	yes	yes	set icon to file
OSC l ST	yes	yes	yes	set window title
OSC L ST	yes	yes	yes	set icon label

Besides the Sun-derived OSC controls for setting window title and icon label, *dtterm* also supported the *xterm* controls for the same feature.

The *CDE* source was unavailable for inspection until 2012, so that clarification of the details of the window operations relied upon *vttest*. However, the manual page for the control sequences (i.e., **dtterm**(5) in the file formats section) was readily available. DEC adapted the control sequences for setting the window and icon labels in the VT525. In doing so, DEC's VT520/VT525 manual changed the letter **l** to a number **1**, and added a parameter **2** before the *l/L* (or *1/L*) code used to distinguish the window and icon labels.

- The SCOSC/SCORC control sequences for saving/restoring the cursor and for saving/restoring "DEC Private Mode Values" (XTSAVE and XTRESTORE) may appear to be related (since the "save" controls both end with s), but that is coincidental. The latter was introduced in X10.4 (December 1986):

  ```
  Most Dec Private mode settings can be saved away internally using \E[?ns,
  where n is the same number to set or reset the Dec Private mode.  The mode
  can be restored using \E[?nr.  This can be used in termcap for vi(1), for
  example, to turn off saving of lines, but restore whatever the original
  state was on exit.
  ```

  while the SCOSC/SCORC pair was added in 1995 by XFree86 (and documented long afterwards).

  The SCO *ANSI* console terminal descriptions did not use these controls (they used the VT100-compatible SC/RC pair). SCOSC/SCORC were an artifact of DOS 2.00 (January 1983), by Microsoft and later supported by SCO and other vendors.

  The SCOSC/SCORC pair is considered a *private mode* because the final characters (s and u) fall in the range from "`" to "~" (octal 0140 to octal 0176). Other *private* control sequences can be constructed by using octets 074 to 077 (characters "<", "=", ">", or "?") at the beginning of the parameter string. The XTSAVE and XTRESTORE controls use "?") in this manner.

  Because the XTSAVE and XTRESTORE controls are private, other terminals may behave differently. For example, DEC (a contributor to the early *xterm* as well as a manufacturer of terminals) used an incompatible private control in one of its terminals more than five years later (for the VT420 PCTerm, announced in February 1992).

  In that model of the VT420, CSI ? P_m ; P_c r selects the *PC TERM* emulation mode. When this mode is enabled, the keyboard sends *scan codes* rather than characters (analogous to X keyboard events). The first parameter of this private control enables or disables *PC TERM* mode, while the second selects a character set. An ambiguity arises if an application omits the second parameter. In that special case, it cannot be distinguished from XTRESTORE. DEC did not take this into account when designing the feature.

  If there were potential users, *xterm* could accommodate this by a resource setting. In retrospect (thirty years later), there have been no uses of *PC TERM*, while the XTRESTORE feature is still in use.

- The *aixterm* manual page gives the format of the control sequence for foreground and background colors 8-15, but does not specify what those colors are. That is implied by the description's mention of *HFT*:

  ```
  The aixterm command provides a standard terminal type for programs that do
  not interact directly with Enhanced X-Windows.  This command provides an
  emulation for a VT102 terminal or a high function terminal (HFT).  The
  VT102 mode is activated by the -v flag.
  ```

  Unlike *xterm*, there are no resource names for the 16 colors, leaving the reader to assume that the mapping is hard-coded. The control sequences for colors 8-15 are not specified by ECMA-48, but rather (as done in other instances by *xterm*) chosen to not conflict with current or future standards.