Text styling control sequences – modernised ECMA-48 (ISO/IEC 6429) text styling

Kent Karlsson
Stockholm, Sweden
2022-12-03

1 Introduction

There are several 'higher-level mark-up' techniques for styling text. Web pages use HTML combined with CSS (either embedded or by reference). RTF (Rich Text Format) and TEX are other examples of markup. The origin for HTML, SGML, is also used. Some "simpler" markup languages are called "mark-down", but they still use (only) printable characters (thus easy to enter by keyboard), so cannot be invisible in plain text, only when interpreted at a higher level. These schemes can have surprising effects, like turning `~nnn~` into text with strike-through: ~~nnn~~.

Sometimes a somewhat more light-weight mark-up (but not necessarily one lacking power) is desired, such as mark-down, but *with the added requirement* that it be "invisible" at the "plain" text level (and better at avoiding surprising unintended formatting). One could try to design one from scratch. But… there is already one such scheme, indeed a well-established one, and in common use, namely ECMA-48 *control sequences* that control text style. While the use of ECMA-48 control sequences may seem old-fashioned, it has several advantages, and there is no need to consider it outdated:

1. ECMA-48 is a well-established standard, since several decades by now. The first edition dates from September 1976, nearly 46 years ago. The latest edition is from 1991, approximately 31 years ago. It does need some updating, for Unicode, for modern display capabilities, modern document display and edit conventions, additional capabilities common in "higher level" schemes for styling text, etc.

2. The ECMA-48 control sequences have the advantage that they can be handled (interpreted or ignored) at the text level, no need for a parser at *higher* level. That is, due to the use of a (lead) control code, the mark-up does not consist solely of printable characters, like in so-called mark-downs (like that used in Wikipedia and elsewhere) or HTML, RTF and the like. The syntax is such that ECMA-48 control sequences, following the standard, can be parsed out easily and "blindly" (not needing to know if it is interpreted), and there is no need for any secondary encoding of some printable characters (like "&lt;" for '<' in HTML).

3. ECMA-48 control sequences are still in very common use. They were particularly popular for terminals, that were later replaced by terminal emulators, which is what is used today. The latter have since been developed beyond the limits of any once-existing "physical" terminal model. The continued popularity of ECMA-48 control sequences is partly for historical reasons, and partly due to point 2. Those who use terminal emulators will encounter ECMA-48 styled text in just about every use instance. For example, it is used for the "`man`"

command (print manual pages on the terminal); underline and bold occurs through-out those pages. It is also used by many other commands for colouring (e.g., `grep` in modern Linux can mark matching substrings in bold red), as well as being popular for use in prompts, often embedding certain status data as styled text in the prompt. Many other programs that output to a terminal also use ECMA-48 styling of the output. Even though ECMA-48 is "old" it is very much still in active use and there is no end in sight.

4. The SGR (SELECT GRAPHIC RENDITION) control sequences (including the extensions given below) and HTSA, SPD, PTX (described below) are well interpretable in modern text editors. Just like formats such as RTF, HTML, etc., are supported by many modern text editors. Likewise, if desired, charset declaration, as well as images and graphics, see hints below (but is out of scope for the here proposed additions), based on ECMA-48 are well supportable in text editors. Thus, one could have a special file format (with a suitable extension such as "txtf": *file*.**txtf**) for text styled with these ECMA-48 control sequences. Such a format would exclude interpretation of cursor movement or other control sequences that are not aimed at text formatting.

This is the closest one can get to "plain text formatting" (since control characters mark the "markup") *while using an existing standard*. (In "higher level markup", printable characters (only) act as control sequences per the syntax of the markup. That goes for HTML, RTF, markdown, etc.)

Some more items of interest w.r.t. ECMA-48:

1. ECMA-48 styling is popular to use for bold, underlining, italics as well as foreground and background colours. There is already support in the standard for much more, even though some need further extensions to be useful. In addition, some features should be moved to SGR (from other control sequences), to improve the design. Many implementations already support extensions w.r.t. colours and underlining (e.g., wavy underlines and colour of underlines separate from colour of the text). ECMA-48 also has support for CJK style of emphasis (similar to underlining).

2. A variety of PTX (PARALLEL TEXTS) is used for so-called "ruby" text, which is a particular kind of clarification text written above the actual (usually Japanese or Chinese) text. This is a feature only relatively lately added to HTML. But supported, standards-wise, by ECMA-48 since more than 30 years, long before it was added to HTML.

3. ECMA-48 may appear not to have any mechanisms for specifying tables (though tab stops can be set, preferably via (resurrected) HTSA (CHARACTER TABULATION SET ABSOLUTE)). But it has another variety of PTX which, with some minor extensions (see below), can be used to make table rows, and hence true tables. Together with some other extensions, quite feature-rich tables can be supported. PTX and SDS/SRS (the latter two for bidi control, similar to the nesting Unicode bidi controls) are the only control sequence from ECMA-48 5th edition that allows for nesting (so one can have tables in tables, or "ruby" text in tables (though "ruby" in "ruby" or tables inside ruby may be less meaningful). However, PTX control sequences must not nest inside bidi controls (whether ECMA-48 or Unicode bidi controls).

4. Headings and item lists are supported as *direct* formatting (and direct numbering) only (in particular size and bold or bold italic, as well as tab stops and HTJ). There are no "meta-

formatting" controls by giving some kind of "element type", and then elsewhere define the formatting for instances of that "element type" (cmp. CSS, which is a prime example of that approach, but exists also in, e.g., TeX) for this or anything else, like paragraphs or table cells. That may in a modern context be seen as a major flaw. But it is a basic design feature of ECMA-48 formatting, and therefore cannot be changed, or at least quite hard to change. We will therefore not introduce any "meta-formatting" controls here, though not prohibited for a future update.

5. Images and vector graphics have no apparent support in ECMA-48. But that is not true. There are extension mechanisms that are "private use" or "for future standardisation" that can be used for this. In particular APC (APPLICATION PROGRAM COMMAND), which takes a string parameter that can encode an image (in base-64 or use a link) or a kind of vector graphics (plotter commands or even SVG). (Some more hints below, but details are out of scope for this proposal.)

6. Hyperlinks were barely invented when ECMA-48 was last updated, so there is no standard support for them in ECMA-48. But some recent implementations based on ECMA-48 do have them via an extension. (And we will hint more on that below, even though details are out of scope for this proposal).

7. There are no named styles. Just a "style state", in general no stack of them (except for PTX and an extension based on an xterm extension). So, by modern standards, it is quite rudimentary. It is really a low-grade "higher-level" protocol for text styling, except that there is no higher level. The codes are "ignorable" (as well as interpretable) for display of text, at the text level, by design. Even so, it is already quite advanced in some respects, for example it has (for decades) included support, standards-wise, a) for CJK style emphasis and b) CJK "ruby" texts (both supported in modern HTML/CSS) as well as framing of text on a line basis (similar in nature to Egyptian hieroglyph cartouches) also supported in modern HTML/CSS.

8. A type of text formatting which does not have any support in ECMA-48 5th edition is math expressions. While one could use **APC** and embed, e.g., TeX math expressions. However, for one thing, that does not cover full Unicode support for math expressions at least not for TeX math expressions.

In this proposal we also include some additions specifically for enabling easier conversion from ISCII styling and from Teletext styling (including that for Teletext subtitling, which is still a common use for Teletext today) to ECMA-48 styling (these are SGR extensions). The details of such conversions themselves are, however, out of scope for this proposal of additions.

ECMA-48 5th edition actually does not say how to use the control sequences, in particular it does not say how to use styling control sequences. For some other control sequences, there are some indications on how to use them. E.g. cursor movement control sequences are to be sent from the "display component" (to be understood as a display with associated keyboard; in practice what today is called a PC, it need not be a terminal or terminal emulator) to the "data component" (to be understood as the software that actually manages a document or a pseudo-document; in practice, text editor programs or text display programs, or programs that has some subcomponents that has text in them, like web browsers with an address line or fields in a form).

For text styling, which is the focus of this proposed update, the styling controls can be used as means of (approximately) display content (with mostly text) which is styled by other means. E.g. programs that display web pages mapping (approximately!) the HTML/CSS styling to ECMA-48 styling. An example of that is the Lynx web browser (http://lynx.browser.org/) displaying (text) web pages, in a very approximate manner, in a terminal emulator.

But the text styling control sequences can also be used as the storage format (in a text document), storing the styling as ECMA-48 control sequences directly. This is then a more light-weight alternative to using HTML/CSS or some other "highly capable" document formats (such as MS Word XML format, ECMA-376-1:2016: Office Open XML File Formats — Fundamentals and Markup Language Reference). Often, one does not need all the advanced capabilities of those storage formats for text documents, and a more light-weight format is sufficient. For that, ECMA-48 text styling (as storage format) comes in handy, since completely unstyled "plain text" can be a bit too poor. So ECMA-48 text styling bridges a gap between the "highly capable" text formats, and the "very poor" pure plain text. And that with an existing mechanism. Though old it is still in widespread use, not needing to invent a completely new middle-ground format. Even RTF may be considered too heavy-handed. For terminal emulators, only ECMA-48 text styling is viable. All other currently existing text formatting mechanisms are out of the question, for technical reasons. In addition, there are also compatibility reasons, ruling out other technically possible (and currently imaginary) styling mechanisms. But here we will focus on storable documents and using ECMA-48 text styling for them in storage, no matter how they are displayed or handled internally in, say, text editing programs. Still, many of the suggested updates here are applicable also to terminal emulators, and certain proposals here are kept in line with what has been done for some terminal emulators.

2 Goals and non-goals for this update proposal

Goals for the additions and clarifications in this proposal:

a) Encouraging existing implementations to support bold/regular/lean font weights in the modern way (no colour change), full colour specifications, use of proper syntax (in particular for full colour control sequences) and other commonly implemented features of ECMA-48 SGR so they are consistent from a user point of view. That is, trying to converge the interpretations of exactly how these should be interpreted (such as fixing commonly implemented colours to be the same RGB values across implementations, just like HTML/CSS named colours are reliable across implementations).

b) Extending the ECMA-48 styling mechanism to other styles or style variants now supported in HTML/CSS, though it will not be as flexible as HTML/CSS, but still in "the same vein" as original ECMA-48.

c) Including some commonly supported ECMA-48 extensions, esp. those for colour levels.

d) Moving some styling controls (from other control sequences) to the "**m**" set of controls (i.e., SGR) and generalise them a bit (line and character spacing, font size, font size modification (condensed, extended), and more).

e) Resurrecting HTSA, "the better way" of setting tab stops (and deprecate other ways of setting tab stops).

f) Specifying PTX better and generalise it, so that table rows with different cell widths (heights if vertical lines) can be used. (Note that PTX, oddly, is also used for “Ruby” text for Japanese/Chinese, though with other parameter numbers.)

g) Limiting and extending SPD (SELECT PRESENTATION DIRECTIONS) and refer to the Unicode Bidi algorithm instead of using ECMA-48 direction controls.

Non-goals for the additions and clarifications:

a) To get all implementations of ECMA-48 to necessarily implement all the styling control sequences (indeed, some are not well suited for, e.g., terminal emulators). Likewise for the other control sequences discussed in this paper.

b) Replace other styling mechanisms (RTF, ‘markdown’, …), though support for ECMA-48 styling may be an *additional* mechanism, in particular in text editors.

c) Add “structural” mark-up to ECMA-48 (auto-numbered lists, automatic heading numbering and styling, named styles, etc.) common in more advanced higher-level mark-up mechanisms. However, PTX has nesting structure, allowing to have tables in table cells.

3 Escape sequence, control sequence and control string regular expressions

Originally, the syntax for escape sequences and control sequences were formulated as byte sequences (or even bit sequences) in ECMA-48. But now we use Unicode, with several character representations that do not conserve the representation as byte sequences. So here we formulate the “overall” syntax (a regular expression), not as *byte* sequences, but as *character* sequences. While not a goal here, this also makes the control sequences applicable to EBCDIC based Latin script encodings.

The general (almost catch-all) syntax for ECMA-48 escape sequences, control sequences and control strings are as follows, generalized to specify Unicode (or ISO/IEC 10646) characters rather than bytes, but excluding code page shifting controls and escapes as well as device controls (except **DCS**):

```
c0-control-character ::=     [\u0001-\u0003\u0008-\u000D\u001A-\u001F]  // has exclusions
ESC ::= \u001B
escape-sequence ::=    ESC    [\u0040-\u004C\0051\0052\u0056-\u005F]  // has exclusions

c1-control-character ::=        [\u0080-\u008C\u0091\0092\u0096-\u009F] |
                       ESC    [\u0040-\u004C\u0051\0052\u0056-\u005F]  // has exclusions
cf-control-character ::= <characters with general category Cf, Zl, Zp>

SCI ::= (ESC \u005A|\u009A)    // consider Unicode surrogates, non-characters, and Cf excluded:
sci-sequence ::=        SCI    [\u0001-\u0003\u0008-\u000D\u001C-\u007E\u00A0-\U10FFFD]

CSI ::= (ESC \u005B|\u009B) // using [\003C-\u003F] first or [\u0070-\u007E] is private use:
control-sequence ::=    CSI    [\u0030-\u003F]*     [\u0020-\u002F]* [\u0040-\u007E]
```

Character encoding switching escape sequences, **ESC**[\u0020-\u002F]+[\u0030-\u004C\0051-\0052\u0056-\u007E], are excluded since character encoding switching is outdated and furthermore not allowed to be used with ISO/IEC 10646/Unicode. Code switching and device controls and code switching and device control escape sequences, **ESC**[\u0030-\u003F\0060-\u007E], are excluded since they are not suitable for storing in text documents. Likewise are character references that refer to code switching or device controls, **ESC**[\u004D-\u0050\u0053-\0055], excluded since they are unsuitable for storing in text document, as are all C0 and C1 device controls. Also start of device control string, **DCS**, is excluded since it is not suitable for storing in a text document. In addition, cursor movement control sequences should also be excluded for the same reason, but for brevity that is not expressed in the regular expressions above; but that would entail excluding certain control sequence terminating letters. Cursor movement control sequences are useful in other contexts involving text, but not for storing them in a text document.

A control character, escape sequence, sci sequence, or control sequence shall not be a code page switching operation, and we have excluded existing ones in the regular expressions above. That is for all contexts, not just text storage. Hence the deletion of intermediary characters, [\u0020-\u002F]*, from the syntax for escape sequences, which was reserved for ECMA-35 (*Character Code Structure and Extension Techniques*) and ECMA-43 (*8-Bit Coded Character Set Structure and Rules*) code page switching. Some C0 and C1 control characters were for code page switching (ECMA-43), but those shall not be interpreted, and have been excluded from the syntax above. Likewise, **LS0** (**SI**, \u000E), **LS1** (**SO**, \u000F), **SS2** (\u008E), **SS3** (\u008F), **LS1R** (ESC \u007E), **LS2** (ESC \u006E), **LS2R** (ESC \u007D), **LS3** (ESC \u006F), **LS3R** (ESC \u007C), or any other code-switching control, *shall* be uninterpreted and may cause an error indication. Escape sequences that do not stand for any codepage switching, like control character references, may still be meaningfully interpreted.

There are some C0 and C1 control codes purely for certain 1960-ies type terminals; those are unlikely to be interpreted anywhere at all today (in particular, xterm and its derivatives ignore them), and those controls are not recommended for any use, not even terminal emulators. Some control sequences are for keyboard input; these are the subject of another update proposal (*Keyboard/mouse UI control sequences – modernised ECMA-48 (ISO/IEC 6429) user input handling*). Yet other control sequences are for updating a terminal screen/window. The latter two types of control sequences are of course unsuitable to be used in a text document, and should be ignored or filtered away if they occur in a text document, especially in an environment where they might otherwise be interpreted (such as a text editor showing text to the user via a terminal emulator); such a text editor may use such control sequences to update the terminal screen/window, but then they come from the editor program, not from the text document. The text styling control sequences that are the subject of this proposal can of course be used in a text document that is edited or displayed without any involvement of a terminal emulator; just an ordinary text editor in a modern "GUI"/window environment; just using ECMA-48 styling instead of RTF, a markdown system (there are several), HTML, or some newly invented "private use" markup for storing the text in file, or even as internal representation.

Note that the notation here refers to characters (including that \u0000-\u001F and \u0080-\u009F always refer to ECMA-48 C0 and C1, respectively, control codes), not bit sequences. The character encoding need not be a Unicode (or ISO/IEC 10646) encoding. It may be EBCDIC based; note that EBCDIC has all the C0 control codes, though jumbled (ok, it predated ASCII), but would need the **ESC**-alternatives for the C1 control characters, as the ECMA-48 C1 control characters are not directly present in EBCDIC. Or a legacy Windows codepage. If it is in a Unicode encoding, it may be either one

(UTF-8, UTF-16BE, UTF-16LE, UTF-32BE, UTF-32LE). It cannot be an encoding depending on code page switching, nor any encoding that does not cover the ASCII characters (excluding, among many others, national variants of ISO/IEC 646). The latter may be handled by ad-hoc mappings (not specified here).

A control string allows for freer content in the parameter part: the parameter(s) need not be numbers. However, no details of possible contents are given in ECMA-48, 5th edition. So all and any content for these is "private use" in the 5th edition.

```
DCS ::= (ESC \u0050|\u0090)       // (xterm used this to "program" function keys) (deprecating)
SOS ::= (ESC \u0058|\u0098)       // suitable for embedded image data and hyperlinks
ST  ::= (ESC \u005C|\u009C)       // xterm allowed (deprecated) BEL (\u0007) as alternative
OSC ::= (ESC \u005D|\u009D)       // used in xterm for setting window titles and such
PM : := (ESC \u005E|\u009E)       // (a message targeted for a "status line") (deprecating here)
APC ::= (ESC \u005F|\u009F)       // suitable for vector graphics (like HP-GL or SVG)
control-string ::= (DCS|SOS|OSC|PM|APC) // consider surrogates and non-characters excluded
(\u0001-\u0003\u0008-\u000D\u001C-\007E] | sci-sequence | control-sequence |
ESC[\u0040-\u004C\u0051\u0052\u0059]|[\u0080-\u008C\u0091\u0092\u0099\u00A0-\U10FFFD])*ST
```

Also here, for control strings, consider Unicode surrogates, non-characters, and Cf characters excluded from \u00A0-\U10FFFD, which we for brevity express this way. One should only use assigned character positions, but that changes over time, as more characters are encoded.

The contents of control strings are here generalised compared to ECMA-48 5th edition. But **BEL** (U+0007) and **NULL** (U+0000) are excluded. Noteworthy is that **ETX** is the default label terminator in HP-GL/2, and that URLs (if using a control string for those) may contain non-ASCII characters. Furthermore, control strings cannot nest, and that is expressed in the syntax above.

A prefix of a *potential* match that in the end does not match, should be treated as a normal character sequence (i.e., only the control characters would, normally, be non-displayed). A true match is normally not shown as is (except in a show-invisibles mode, if at all kept as is), but may affect the display of printable characters that follow. A match must never be case-mapped and should be invisible in display and ignored in collation operations.

The control strings cover DEVICE CONTROL STRING (**DCS**, \u0090, **ESC P**), APPLICATION PROGRAM COMMAND (**APC**, \u009F, **ESC _**), PRIVACY MESSAGE (**PM**, \u009E, **ESC ^**), OPERATING SYSTEM COMMAND (**OSC**, \u009D, **ESC]**), nor START OF STRING (**SOS**, \u009B, **ESC X**) sequences. All of these allow for arbitrary (originally only parts of ASCII; some terminal emulators now allow UTF-8) characters between the start and end control characters, terminated by a STRING TERMINATOR (**ST**, \u009C, **ESC \**) control character/escape sequence (xterm allowed also **BEL** to terminate such sequences, presumably not with the intention to sound the "bell" then). ECMA-48 5th edition has for **SOS**…**ST** a bit different specification for the … part than for the others (**DCS**/**PM**/**OSC**/**APC**). Making that distinction does not seem helpful, so we ignored it here, and all allow the same generalised "content" (in the catch-all-type of syntax given above).

3.1 General additions for control sequences

Here we reserve **=** (EQUAL SIGN), postfix, for denoting a negative value of a parameter. We cannot use HYPHEN-MINUS as that is reserved as an "intermediary character" that can occur only before the final character of the control sequence. And we reserve **?** (QUESTION MARK) for denoting decimal separator, with at least one digit before the **?**. Note that both FULL STOP and COMMA are "intermediary characters" associated with the final character. Note also that when <, =, >, or ? do *not* occur *first* after the **CSI**, they are *not* reserved for private use, but when one of those four characters occur *first* after the **CSI**, the control sequence is reserved for private use (a few such private use control sequences are defined for xterm).

3.2 Control strings: Some possible uses of APC and SOS

Possible uses for APC and SOS control strings are:

- *Declare character encoding*. We here suggest **SOS charset=***charset-name* **ST**, default: **UTF-8**; instead of using ISO 2022 escape sequences to declare character encoding.
- *Declare language*. We here suggest **SOS lang=***IANA language tag* **ST**. No default value.
- *Clickable links* (hypertext references), like **SOS a=***<hypertext reference>* **ST** *<display text>* **STX**; the **STX** starts the "normal" (non-link) text after the link text.
- Embedding *images* (via hypertext references), like **SOS** [**w=***<width>***;h=***<height>***;**]**img:***<link to image>* **ST**; the (optional) width and height are values with unit (incl. *em*, and percent).
- Embedding graphics as *plotter commands.* Here we suggest two possibilities: 1) Use HP-GL/2: **APC** [**w=***<width>***;h=***<height>***;**]**hpgl:***<HP-GL commands or* `.hgl` *filename>* **ST** (as for strings output via HP-GL: HP-GL can in principle, when used in an otherwise ECMA-48 context, be extended to use ECMA-48 styling for label strings). 2) Or even, despite the apparent anachronism, use SVG with inline CSS: **APC** [**w=***<width>***;h=***<height>***;**]**svg:***<SVG 'commands' or* `.svg` *filename>* **ST**. Many programs output "ASCII art", assuming fixed width, to display certain graphics on terminals or resort to produce the graphics in a file to be displayed by *another* program. By allowing the use of this kind of embedded graphics, the graphics can be output (also) directly to a terminal, as well as embed the graphics in a *.txtf* file.
- Define (possibly parametrised) macros via **APC**, that then can be used via control sequences referencing them (and possibly giving parameters). Not (yet) proposed here.

These are here only mentioned as *possibilities*, from the viewpoint of this document. The examples here are applicable for *.txtf* documents (and use ECMA-48 based styling controls, also extended as proposed below).

3.3 Escape sequences and control sequences not suitable for in-text use

ISO/IEC 2022 escape sequence (escape sequence with non-empty "intermediary characters" part), which were used for code switching, are disallowed. Such escape sequences shall be ignored. Also other codepage switching controls shall be ignored, and all are unsuitable to occur in any text.

Certain escape and control sequences in ECMA-48 are edit position movements, screen redraw commands, interrogations/replies to/from a device (terminal) from an application, or other uses that is not text formatting. ***Such*** escape and control sequences shall not be part of any "plan text formatted" document and shall be ignored if occurring in a text document.

3.4 Unicode character references

Certain escape sequences (**ESC** [\u0040-\u005F]) refer to ECMA-48 C1 control characters (**ESC** [\u0040-\u004C\u0050-\u0052\u0056-\u005F], with exclusions of unsuitable C1 controls). They are useful in case the normal C1 controls are not available, or their availability is uncertain. This is a limited form of character references, only for C1 control characters. A more general (and new) character reference scheme (but excluding C1 control characters) is given below.

Unicode has very many characters encoded. We here introduce Unicode character escapes, in ECMA-48 style. Here "escape" does not refer to **ESC** or escape sequences starting with **ESC**. It refers to the more general concept of an "escape", here for "character escape", which in the ECMA-48 instalment actually does start with **ESC** (using the fallback for **CSI**) or **CSI**. This mechanism is similar to HTML's decimal "character escapes" (but in ECMA-48 style). Here we propose as a decimal character escape (as a control sequence):

- **CSI** *n*_, where *n* is a Unicode codepoint (scalar value) in *decimal* form (hexadecimal cannot be used due to ECMA-48 syntactic reasons), without leading zeroes and only referring to valid Unicode scalar values larger than 159 (decimal). **CSI** *n*_ cannot be used directly after **SCI** (nor can an escape sequence nor control string occur directly after **SCI**).

However, direct character use (and using a Unicode character encoding) is preferred. Character references need to be parsed and replaced before certain actions are done, such as bidi, line breaking and more.

Note that the \u009B (CONTROL SEQUENCE INTRODUCER, **CSI**) is in the C1 space, and is therefore avoided by many terminal emulator implementations, due to the risk of misinterpreting something that is *not* a control code for this control code. E.g., terminal emulators may sometimes run in the "wrong" character encoding (compared to what is actually output to the terminal), this is a particularly sensitive issue for terminal emulators. So though standard, the \u009B code (in the currently set character encoding) is often left as is in terminal emulators, not interpreted as a control character, since the encoding setting is not always trustworthy in a terminal emulator where a "client" program (started as a "command") may blindly output in a different encoding than the one set for the terminal emulator. Though far from a complete solution, one is then only using the **ESC [** alternative for **CSI**, in many terminal emulators.

The situation is different for text files with formatting/styling in the ECMA-48 way, where using C1 characters directly should not be a problem if the file is coded in UTF-8 or UTF-16. However, if one is using a Windows codepage or an EBCDIC codepage for the file, one will need to use the **ESC** variants for ECMA-48 C1 characters, as those encodings do not have the ECMA-48 C1 characters.

Ink-less characters

While borderline out of scope for this proposal, the handling of line breaks and spaces is important enough for the rendering of characters, that some guidelines are called for.

Note that these contrast to edit position movement commands, e.g., CURSOR NEXT LINE which is a cursor movement command (for text displaying applications). Control sequences for edit position movement (or extension/shrinking of text selection) are not intended to be part of text. There is unfortunately no quick and easy distinction in ECMA-48 whether a control sequence is fit for "in-text" or not, other than enumeration. The characters below, however, if entered from a keyboard (or by pasting) insert these characters in the text, if working with a text editing application or some kind of field that allows text entry. However, some line break characters may be interpreted as an "execute" control in applications such as command line interpreters.

3.5 Line break characters: LF, CR, CR+LF, PLD, PS, LS, VT, FF

The following characters "just" advance the display position in the line progression direction plus to a beginning of line position in the character progression direction. Just as for most "default ignorable" Unicode characters these should have no glyph lookup via the font (unless in a "show invisibles" mode, where these characters also have glyphs, but may be taken from another font and not related to the character via the font, just via the application).

- 000A; LINE FEED (LF), originally pure line feed, now handled as equivalent to CRLF/NEL/PS.
- 000D; CARRIAGE RETURN (CR) (entered by *return* key) often mapped to LF/CRLF or NEL; but sometimes still interpreted as a pure carriage return, e.g. in 7-bit encoded SMS messages.
- 000D, 000A; CARRIAGE RETURN, LINE FEED (CRLF), should be handled as a single unit.
- 008B; PARTIAL LINE FORWARD (also as character reference: **ESC K**) (PLD) (proposing *ctrl-shift-return*), here proposed to be used only after a character in *this* list of line breaking characters or after a PTX (PARALLEL TEXTS) with argument 1 (iff closed by PTX with argument 2 or 0), 2, 0. Was intended mostly to mimic "typewriter superscript/subscript" (together with PLU). But in modern contexts, one also wants a size change, which is typographically more normal. The use of PLD as "extra space between paragraphs" is still valid though, even if coarser than giving such spacing in points or, better, (fractions of) em.
- 000B; LINE TABULATION (VT) (nowadays often entered as *shift-return*), should act as LS for bidi. This was originally intended for rapid form fill-out (where the paper had the actual form pre-printed, and just filling in the data). But it has gained a new use: to act as LS, but in C0 code space.
- 000C; FORM FEED (FF) (sometimes entered as *ctrl-return*), should act as LS for bidi.
- 0085; NEXT LINE (also **ESC E**) (NEL), was intended as replacement for CRLF but is actively used only in EBCDIC based environments.
- 2029; PARAGRAPH SEPARATOR (PS, intended to disambiguate CR/LF/CRLF)
- 2028; LINE SEPARATOR (LS), intended to disambiguate CR/LF/CRLF, but now, instead, VT is often used in this sense rather than its original sense.
- 001C,001D,001E; IS4, IS3, IS2 (these should not occur in text) should act as BK for line breaking. These were intended for "record structure" in now-defunct record interchange protocols. (Today, that would be called "linearization", and done differently.)
- 001A; SUBSTITUTE, should display similar to U+FFFD, REPLACEMENT CHARACTER, since they serve the same function.

Few applications/"devices" handle these in the way originally intended. (Some systems also fiddle with some of these characters; "cooked" mode vs. "raw" mode; effectively "modified interpretation" vs. "original interpretation". The details are out of scope for this paper.) But that does not mean that they should be left completely uninterpreted. If not interpreted in the original way, interpret (by default) each of them as a line breaking character, though for PLD the line advance should be only about 0.6 to 0.8 em, and should (since it is here proposed not to be for use for superscript or subscript indices, see below for replacement) occur only directly after one of these line breaking characters (including itself).

Many applications today implement automatic line breaking (and automatically re-evaluating the line breaking upon change of "line width" on the "front end"). Certain control sequences (here proposed not to be used, see below) assume that all line breaking is done by hard line-breaks (i.e. explicit line break characters). But much of ECMA-48, esp. SGR, work well also when many line-breaks (or, rather, line wrapping) are made automatically.

Note that when doing automatic line breaking, one should try to avoid local "jaggedness" in line length if possible, like done by TEX via a points system, as well as by automatic hyphenation.

There is some ambiguity as to how to represent a "paragraph break" vs. an explicit "line break". The recommendation here is: LF, CR, CRLF, NEL, PS should be regarded as equivalent, and (CR)VT, (CR)FF, LS should be regarded as equivalent (though (CR)FF also still often retains its original interpretation). VT is commonly used in place of LS in many applications today.

Only the interpretation of the style set by **CSI 75:…m** and **CSI 76:…m** (both new) are affected by the difference between PS (etc.) and LS (etc.): namely whether a line is considered as a:

- paragraph start line: text just after beginning of text, LF, CR, CRLF, NEL or PS,
- not paragraph start line: text just after automatic line break, (CR)VT, (CR)FF, IND or LS.

The distinction also affects the Unicode BiDi algorithm, when that is enabled (see SPD below).

3.6 Space characters: HT, HTJ, SP, NBSP, NNBSP, ENSP, EMSP, IDSP, …

The following characters "move" the display position (and for HTJ also moves some preceding characters) in the character progression direction and just as for "default ignorable" Unicode characters there should be no glyph lookup (except possibly for OGHAM SPACE) via the font (unless in a "show invisibles" mode, where there is also a glyph for these, but that may be taken from another font and not related to the character via the font, just via the application).

- 0009; CHARACTER TABULATION (HT). HT moves *at least* 1 en, to the next tab stop after that in the character progression direction. The stop may be a default one or a set one.
  This was originally intended for "rapid form fill-out", where the paper had the actual form pre-printed, and just filling in the data, just like for mechanical typewriters. But it rapidly gained a slightly different use: rapid spacing, especially at the beginning of lines.
  Note that **CSI** [*n*]**I** (CURSOR FORWARD TABULATION, CHT) and **CSI** [*n*]**Z** (CURSOR BACKWARD TABULATION, CBT) are *cursor movement* commands when editing a document. The latter are editing commands, and not characters that should be stored in a document.
- 0089; CHARACTER TABULATION WITH JUSTIFICATION (HTJ, also as a character reference: **ESC I**). HTJ not only moves to the 'next' tab stop in the character progression direction (moves *at least* 1 en, to end 'tab field') like HT, it also moves the *content* of the preceding tab field (back to the nearest of: beginning of text, nearest preceding (PS…, LS…, automatic) line break or tab character in the same displayed line) to align the end of that text with the next tab stop at least 1 en after the current position. Useful for alignment of numeric literals (with the same number of decimals) on several lines and similar. HT has a special "border" interpretation in the Unicode bidi algorithm; that should HTJ (also when given as the character reference **ESC I**) also have. HTJ should act as HT for line breaking.
- 001F; IS1 (should not occur in text) should act as HT in display (and acts like HT for bidi).

- 0020; SPACE (SP). The width of SP can stretch for automatically justified lines.
- 3000; IDEOGRAPHIC SPACE (IDSP). 1 em wide, but may stretch in justified lines.
- 1680; OGHAM SPACE MARK. May have a glyph; if glyph is missing it acts as an SP.
- 00A0; NO-BREAK SPACE (NBSP). NBSP can stretch for automatically justified lines.
- 202F; NARROW NO-BREAK SPACE (NNBSP). PSP width, suitable as space between digit groups.
- 2007; FIGURE SPACE (no-break). Should be the width of a “0”, esp. if fixed-width digits.
- 2008; PUNCTUATION SPACE (PSP). Should be the width of a FULL STOP or COMMA glyph.
- 0082; BREAK PERMITTED HERE (BPH, also when presented as the character reference **ESC B**). Unicode prefers using U+200B ZERO WIDTH SPACE. Explicitly indicates a position for an allowed automatic line break.
- 200B; ZERO WIDTH SPACE (ZWSP; replacement for BPH).
- 0083; NO BREAK HERE (NBH, also when presented as the character reference **ESC C**). Unicode prefers using U+2060 WORD JOINER.
- FEFF; ZERO WIDTH NO-BREAK SPACE (ZWNBSP (and BOM); was replacement for NBH).
- 2060; WORD JOINER (WJ; zero-width no-break word *separator*).
- 00AD; SOFT HYPHEN. A zero-width space, but explicitly indicates a hyphenation position.
- 2000; EN QUAD (canonically equivalent to U+2002 EN SPACE).
- 2001; EM QUAD (canonically equivalent to U+2003 EM SPACE).
- 2002; EN SPACE (1/2 em)
- 2003; EM SPACE
- 2004; THREE-PER-EM SPACE (1/3 em)
- 2005; FOUR-PER-EM SPACE (1/4 em)
- 2006; SIX-PER-EM SPACE (1/6 em)
- 2009; THIN SPACE (1/8 em)
- 200A; HAIR SPACE (1/24 em)
- 205F; MEDIUM MATHEMATICAL SPACE (4/18 em, 1/4.5 em)
- 180E; MONGOLIAN VOWEL SEPARATOR (zero-width, may change shaping of a m. vowel).

Note that also characters with (Unicode) scalar values greater than 009F will get character references defined below. The ESC character references for C1 characters always refer to C1 as defined in ECMA-48 5th ed. regardless of character encoding, considering what follows the ESC as a character, not a byte (so, e.g., **ESC I** refers to **HTJ** also if the encoding is EBCDIC based). C0 may be rearranged in some character encodings, like those that are EBCDIC based. We refer to the *characters*, **not** the bytes (or larger code representations).

4 SELECT GRAPHIC RENDITION – SGR (extended and clarified)

Note that certain styling should *not* be applied to prestyled characters, e.g., MATHEMATICAL BOLD CAPITAL A is bold upright regardless of weight and italics settings. Emoji are in addition insensitive to text colour setting. Prestyled as superscript characters are displayed as first level superscripts, regardless of superscript/subscript/neither setting (see below). We will not make a full list here. Note also, that when bidi is used, a single logically contiguous style substring may be split into multiple substrings of that style. The details of this are out of scope for this proposal.

ECMA-48 "mode" changes shall *not* be supported. GRCM – GRAPHIC RENDITION COMBINATION MODE must be fixed to CUMULATIVE. TSM – TABULATION STOP MODE must be fixed to MULTIPLE. Further, GCC – GRAPHIC CHARACTER COMBINATION (a kind of character combination; **CSI** … **SP _**) shall not be implemented, nor should BS or CR be used for overtyping, except for typewriter-like devices (which are very rare today); Unicode has other ways of composing characters, and here we give other ways of underlining, overstriking and make bold. RM – RESET MODE (**CSI** …**l**; small ell) and SM – SET MODE (**CSI** …**h**) also shall not be implemented.

The SGR control sequence can specify several rendition changes in a single control sequence. But **CSI** *a***;***b***;***c***;**…**;***x***;***y***;***z***m** can be turned to **CSI** *a***m CSI** *b***m CSI** *c***m**…**CSI** *x***m CSI** *y***m CSI** *z***m** (with no semicolon within the variables, but here they may contain colon, the space between **m** and **CSI** is part of the notation here, not to occur literally).

Note that when (and if) bidi processing is applied, a single original "span" with a particular "graphic rendition" (called "style" today) may be split up into *several* display spans with *other* display spans having other styles *in-between*. How to manage and represent that in an implementation is out of scope for this proposal but needs to be handled by an implementation of ECMA-48 styling that also implements bidi processing.

Here we go through the SGR codes, existing in ECMA-48 5th ed. as well as here proposed extensions, additions and clarifications, grouped by function, not just listed numerically.

4.1 RESET RENDITION (extended)

This control sequence resets the SGR settings to the default. What is the default is out of scope for this proposal (as it is for ECMA-48 5th ed.). If pushing SGR attributes is supported (see below), the top-of-stack of stored-away values are counted as default. Reset rendition should be avoided in documents and output. This is a shortcut that can be handy, not needing to reset the style changes one by one. This is useful in cases where the "current SGR state" is not or cannot be known by the application, and a reset is desired. For instance, at the *beginning* of a terminal program's prompt, after a very first **ST**, just in the, slightly unlikely, case the previous output was terminated in the middle of a control string. (If style stacking is implemented, one may also want a **CSI** *n***!]**, for a suitable n, between the **ST** and **CSI 0m**. Imperfect, since……..)

- **CSI 0m** Reset rendition. The 0 can be omitted (**CSI m**). Resets all, except indents/tabs and colour palette modifications, rendition attributes (set via SGR) to the default setting and closes all nesting layout features like PTX (tables), bidi controls (ECMA-48 or Unicode) and other nesting features (like for math expressions), but not explicit stacking of SGR settings.

Hard line breaks do not, according to the ECMA-48 standard, reset the rendition set by SGR control sequences. However, some spacing setting control sequences, here proposed *not* to be used (replacements in SGR are proposed below), are auto-reset on (hard) line-breaks.

A prompt (for a command line interpreter) should *begin* with **ST** (to close any control string that may happen to be open) and then **CSI 0m**, followed by the actual prompt. **CSI 0m** should not be used in any other context (with the rare exception of ISCII formatting being converted, where using **CSI 0m** is handy, but not necessary). **CSI 0m** is the only control sequence in this proposal that is specifically for terminals (or, rather, command line interpreters). All the rest in this proposal is suitable for a text document format, and the UI styling controls suitable for a text document editor. (However, ISCII formatting and Teletext formatting do formatting reset when moving to next line (or page); so when emulating those, e.g., due to character coding conversion, a **CSI 0m** is suitable just after an (explicit) line break (or page break), to make the emulation correct.

A note for terminal emulators (command line interpreters): Some programs (esp. those outputting partial content of large text files or text streams) currently do assume, or even do, effectively reset the SGR set rendition, inserting extra **CSI 0m**, and more, on (some) hard line breaks. That may result in somewhat “style thwarted” display of some portion of text, compared to if the entire preceding part of the document had been read and been ECMA-48 interpreted before display, if the document has styling that spans more than (say) a few lines or “paragraphs”.

4.2 FONT WEIGHT (clarified, extended with variants)

Font weight is the scale where the boldness or leanness of a font is given, there is even degrees of boldness or leanness. Most modern fonts have a normal weight and a bold variant, many have also a lean variant. In multiple-master fonts one can continuously vary this aspect of a font from hyper-lean to hyper-bold within the font (no need for parallel fonts). This setting does not alter the colour of the text and does not affect in-line images (like emoji) in the text, nor prestyled bold, italic or fractur characters, nor ‘line drawing’ characters.

- **CSI 1**[**:***v*]**m** <u>Bold font variant</u>. Cancels lean and normal (and current bold variant). Variants (note that the separator is colon, *not* semicolon): **CSI 1:0m** normal weight (alias for **CSI 22m**), **CSI 1:1m** semibold (if available, otherwise bold), **CSI 1:2m** bold (default), **CSI 1:3m** extra-bold (if available, otherwise bold). Negative variants (with **=** as negation sign, postfix, e.g., **CSI 1:3=m**) select lean variants. Note: <u>*NO CHANGE IN COLOUR*</u>, only in weight. Fonts that are multiple-master do this within the font, otherwise there may be a font change to a related font that is bold. Contexts that are strictly fixed width might not implement the more “extreme” bolds, even if available in the fonts, since they usually need wider advance widths. This styling does not apply to inline images (e.g., emoji) or characters that are prestyled w.r.t. boldness/leanness or italic/Fraktur/double-struck (e.g., MATHEMATICAL characters). Higher values of “boldness” may increase the width of “en” and “em”.

- **CSI 2**[**:***v*]**m** <u>Lean</u> (light) <u>font variant</u>. Cancels bold and normal (and current lean variant). Variants: **CSI 2:0m** normal weight (alias for **CSI 22m)**, **CSI 2:1m** semilean (if available, otherwise lean), **CSI 2:2m** lean (default), **CSI 2:3m** extra-lean (if available, otherwise lean). Negative variants (with **=** as negation sign, postfix, e.g., **CSI 2:3=m**) select bold variants. Note: <u>*NO CHANGE IN COLOUR*</u>, only in weight. Fonts that are multiple-master do this within the font, otherwise there may be a font change to a related font that is lean. Neither bold nor lean affect inline images (like emoji), nor affecting characters that are prestyled in this regard.

- **CSI 22**[**:***v*]**m** <u>Normal weight</u> (neither bold nor lean) (default). Note: <u>*NO CHANGE IN COLOUR*</u>, only in weight. Same variants as for bold/lean, but the default is **:0**.

4.3 ROMAN/ITALIC (or OBLIQUE) STYLE (extended with variants; note that Fraktur is moved to font change)

This styling does not affect inline images (like emoji) in the text, nor prestyled bold, italic or fractur characters, nor 'line drawing' characters.

- **CSI 3**[**:***v*]**m** Italicized or oblique (−8° to −12°). Oblique (normally synthesised) if there is no associated italic version. Variants: **CSI 3:0m** alias for **CSI 23m**, **CSI 3:1m** italicized (if available, otherwise −8° to −12° oblique), **CSI 3:1=m**:+8° to +12° oblique (normally synthesised) which is sometimes used in RTL scripts. This styling does not apply to inline images (e.g., emoji) or characters that are prestyled w.r.t. boldness/leanness or italic/Fraktur/double-struck (e.g., MATHEMATICAL characters). Does not apply to Fraktur (or 'calligraphic') fonts, if such a font is set, this setting also changes to the default font.

- **CSI 23**[**:***v*]**m** Roman/upright, i.e. not italicized/oblique (default). If the current font is a "calligraphic" font, this setting also changes to the default (and non-calligraphic) font. Same variants as **CSI 3**[**:***v*]**m**, but the default variant is **0**. In addition, variant **1** is always interpreted as oblique (−8° to −12°), never as italic. Does not apply to characters that are prestyled w.r.t. italics/bold/Fraktur.

4.4 FONT CHANGE (extended with variants; note that Fraktur is moved here)

One of the following two non-calligraphic fonts should be the default font, i.e., what you get when using "**CSI 0m**", usually a serif font. Changing font does *not* change any currently set tabulation positions.

- **CSI 20m** Change to a predetermined calligraphic font (set). So-called "calligraphic" fonts are extra adorned fonts. This includes Fraktur (https://en.wikipedia.org/wiki/Fraktur). Italic/oblique does not apply to this kind of fonts, so "italics" is implicitly cancelled when changing to a calligraphic font. Which font (or font set with different weights) may be given by the implementation or may be settable in preferences (or even an **APC** control string).
- **CSI 26**[:*v*]**m** Change to a predetermined proportional (non-calligraphic) font (set) (serif or sans-serif). Variants: **CSI 26:0m** serif, **CSI 26:1m** sans-serif. Which font (or font set with different weights, condensed/expanded, and italic or not) may be given by the implementation or may be settable in preferences (or even FNT or an **APC** command).
- **CSI 50m** Change to a predetermined "fixed" width (non-calligraphic) font (set). Kerning is not applied. Which font (or font set with different weights, condensed/expanded, and italic or not) may be given by the implementation or may be settable in preferences. Though called "fixed width", it will have three different widths: 0-width (non-spacing diacritics, most escape and control sequences, as well as most control strings; though the latter will not be handled by the font per se, but long before), narrow (most scripts), wide (double-width) for CJK, Hangul and emoji (and "…"). Note that some characters come as both emoji and non-emoji (may use variant selectors to indicate which); they have different widths.
- For *y* in **10** to **19**, **CSI** *y***m**: Pick from a font palette of up to ten fonts (set permanently, as preferences, via FNT – FONT SELECTION, or via **APC** control string). Each index refers to a set of fonts, varying in italics and weight, if not multiple-master. E.g., **CSI 19;1m** will select the bold

version of the font at index 19 (whether multiple-master or separate font file for the bold). For each such font set, there needs to be information whether it is "calligraphic" or not (for the interpretation of **CSI 3m** and **CSI 23m**). (Note: XTerm has an extension with which one can specify a font by name to change to or change the font palette. It uses **OSC** but should use **APC**.)

Changing font does not in itself change the width of spaces, the advance widths of which are not to be directly read from the font (except for OGHAM SPACE MARK, which may have a glyph, and spaces whose width correlate with the width of certain punctuation characters). However, for fixed width fonts the nominal advance width for SP and NBSP are 1 en. For variable-width ("proportional") fonts, the nominal advance width for SP and NBSP is usually set to a somewhat smaller value (like 0,7 to 0,9 en, but is implementation defined, and may be a user preference).

4.5 FONT SIZE CHANGE (new, replacing GSS, GSM)

In "original" ECMA-48, units for sizes and advances were set by a separate control sequence, SSU (SELECT SIZE UNIT). This design is flawed for several reasons (major reason: the unit is separated from the value and that can easily lead to very wrong interpreted values, especially if one is doing cut-and-paste of raw ECMA-48 styled strings, or similar kinds of substitutions), and replacements for "SSU dependent" control sequences are proposed below. But we do keep the unit codes from SSU, for backwards compatibility on that point. Zoom is applied after size determination. Changing font size or font magnification does *not* change any currently set tabulation positions.

Unit codes (**0** to **8** from SSU, **9** is new; an implementation need not support all units):

- **0** CHARACTER – 'en' (0,5 em) in the character progression direction (usually horizontally), even for double-width scripts (CJK); 'em' in the line progression direction (usually vertically) ('em' is also supported as a unit in CSS)
- **1** MILLIMETRE (this is called 'mm' in CSS, it is not necessarily exactly 1 mm)
- **2** COMPUTER DECIPOINT – 0,03528 mm (254/7200 mm) (this is called 'pt' in CSS)
- **3** DECIDIDOT – 0,03759 mm (10/266 mm)
- **4** MIL – 0,025 4 mm (254/10 000 mm)
- **5** BASIC MEASURING UNIT (BMU) – 0,02117 mm (254/12000 mm)
- **6** MICROMETRE – 0,001 mm
- **7** PIXEL – The smallest increment that can be specified in an (old) device (this is called 'px' in CSS, it is not necessarily one actual pixel in size, especially with modern high-resolution devices 1px may be several actual pixels)
- **8** DECIPOINT – 0,035 14 mm (35/996 mm)
- **9** default size (value parameter ignored) (the top "pushed" size w.r.t. **CSI ![**, see below)

They should be interpreted in the same way as in HTML/CSS (extended for the units that are not in HTML/CSS), see https://www.w3.org/Style/Examples/007/units.en.html.

- **CSI 72:***u***:***s***m** Font size. Unit *u* as per above. *s* can be negative with EQUAL SIGN (**=**) postfix, indicating negation (for an upside down mirrored glyph), *s* can be fractional with **?** as decimal marker, e.g. **CSI 72:2:14?5m**, for 14,5 points. **CSI 72:9m** resets to default size. This replaces GSS (GRAPHIC SIZE SELECTION), which is here proposed not to be used nor implemented. SSU is also proposed not to be used/implemented, and the units are given in the SGR control code sequences instead. This replacement is due to several factors, because GSS was ill designed. a) The unit was set separately, which can cause major unintended rendition flaws. b) Even though there were several units associated with this, GSS did not allow for decimal values, so no (near accurate) conversion was possible, nor did it allow for negative values (which is of course optional to implement). c) Sizing should be reset by **CSI 0m**, but GSS was not. Note that increasing the font size may temporarily increase the line spacing in order to accommodate the larger glyphs, if the line spacing is given in the "em" unit. The size should be glyph size based (e.g. based on Åg-height or similar glyph measure, and the adjustment may vary over a single font), not just verbatim size from the font, so that glyphs appear size-wise similar for a given requested size. Changing the font size also changes the size of the en and em (approx. Åg height, *after* adjusting the declared font em) units.

- **CSI 73:***a***:***b***m** Font magnification. *a* (vertical factor) and *b* (horizontal factor) can be fractional ('**?**' as decimal marker), and *b* can be negative ('**=**' as negation sign, postfix), e.g. **CSI 73:1:0?8m** for narrow (may use condensed/expanded axis in a multiple master font), **CSI 73:1:1m** resets. This setting is *not* accumulative. Arguments are direct factors, not percent. This replaces GSM (GRAPHIC SIZE MODIFICATION) which is here proposed not to be used. This replacement is due to several factors, because GSM was ill-designed. a) It was percentages but did not allow for decimal values. b) Font magnification should be reset by **CSI 0m**, but GSM was not. This setting should take into account condensed/expanded variants in a multiple master font. Increasing the *vertical* height via magnification will "glue" the glyphs at cap height, making them expand below the (normal) baseline on horizontal lines *without* increasing the line spacing, whereas increasing the font size "glue" them to the baseline, growing the line spacing for the lines with increased font size. Changing font magnification does *not* change the size of the en and em units, but changes the glyph and space advance.

4.6 UPPER/LOWER HALF GLYPHS (new)

This control sequence is strongly not recommended. It is proposed *only* to make certain conversions from ISCII more direct (or in some cases: possible).

- **CSI 113:***v***m** Show horizontally halved glyphs. Variants: **:0** show glyphs normally, **:1** show only the upper half of the glyphs (for combining sequences), cutting about half cap height and counting only half an em for these glyphs for line height calculations, **:2** show only the lower half of the glyphs (for combining sequences), moved up to just under the half glyphs of previous line (so that two lines with the same text, constant size, the upper half joins with the lower half). This is for making full conversion from ISCII possible.

4.7 RAISED/LOWERED TO FIRST INDEX POSITION (new, replacing PLU, PLD)

ECMA-48 has PLU (PARTIAL LINE UP/BACKWARD) and PLD (PARTIAL LINE DOWN/FORWARD) for making indexes (if the character progression direction is horizontal), mimicking manual typewriters, moving the paper. They do have a bit of generality, and *nroff* in its days used such partial line movements to output mathematical formulas using typewriter-like terminals (but for mechanical reasons, the printed lines had to be rearranged so that the paper only moved in one direction and several times, with different type wheels). Nowadays, that way of printing formulas is not recommended. In addition, first level superscript and first level subscript (when not in a mathematical formula) are often regarded as character styling nowadays.

The control sequences below will make characters display at the first index position, raised or lowered. This replaces PARTIAL LINE FORWARD/DOWN and PARTIAL LINE BACKWARD/UP codes for making indices. PLD/PLU do not do the size change normally associated with superscripts or subscripts. PLD (\u008B or **ESC K**) is still useful for making a little bit extra spacing (about 0,6 to 0,8 em, default total of about 1,8 em to 2 em instead of default 1,3 em) between, e.g., paragraphs.

Note that for some uses, like powers of units (like m^2), chemical numerical subscripts (like H_2O) as well as phonetic notation (like ^{h}), preformatted Unicode characters are *strongly* preferred over using styling to achieve the raised/lowered display. Preformatted Unicode characters are also sometimes preferred for certain abbreviations, like ordinals with letter superscripts, and also for other common abbreviations in (e.g.) French. Preformatted superscript/subscripts are not affected by the control sequences for superscript/subscript styling.

On the first instance after a “normal” character of a switch directly from superscript to subscript or vice versa, whether by these control sequences or preformatted superscript/subscript characters, the display position is moved back to after the last “normal” character. When ending, the display position is moved to after the longest of the two sequences. This way, one can apply superscript and subscript nicely together. However, attempting to apply multiple raised or multiple lowered strings to the same “normal” character will have implementation defined effect.

Superscripting and subscripting is common notation in mathematics. But those superscripts subscripts (that are part of math expressions) will need to use a different mechanism in order to fit with math expression layout.

- **CSI 56:1**[**:***v*]**m** Raised to first superscript level and slightly smaller, about 80% current set size. Overline will temporarily skip; strike-through and underline are not affected (not moved). Does not accumulate, i.e., one cannot achieve second level superscript or superscript of a subscript. *v*=0 (default): no underline at superscript level, *v*=1: singly underlined at superscript level, *v*=2: doubly underlined at superscript level.
- **CSI 56:1=m** Lowered to first subscript level and slightly smaller, about 80% current set size. Underline will temporarily skip; strike-through and overline are not affected (not moved). Does not accumulate, i.e., one cannot achieve second level subscript or subscript of a superscript.
- **CSI 56**[**:0**]**m** Not raised/lowered and back to the set size. (Default.)

4.8 LIGATURES/MARKED ZERO (new)

Modern variable width fonts often have automatic ligatures (those that would, or should, occur in modern typeset texts). Fixed width fonts usually do not have ligatures, not even modern ones, but may have. Either type of font would in “ordinary” contexts not be marking zero (0) specially (to make it more clearly different from O, capital o). In some computer contexts, zero is sometimes marked with an internal slash (internal to make it different from the letter Ø), or an internal dot. This marking is often used in fixed-width fonts used in terminals (terminal emulators).

- **CSI 57:1**[**:***v*]**m** Allow modern ligatures (according to font, but commonly fi, fj, fl, fb, fh, fk, ff, ffi, ffj, ffl, ffb, ffh, ffk, but also others), do not mark zero (if the font supports that feature). This is ordinary running text style. Variants, *v*: **:0** only affect "0", **:1** only affect modern ligaturing, **:2** affect both (default), **:3** affect both and allow also optional (discretionary) ligatures (such as st and ct ligatures, but also others).
- **CSI 57:2**[**:***v*]**m** No ligatures but mark zero (if the font supports that feature). This is terminal style. Variants, *v*: **:0** only affect "0", **:1** only affect ligaturing (modern and optional), **:2** affect both (default), **:3** only turn off the optional ligatures.
- **CSI 57**[**:0**]**m** Reset to default w.r.t. ligatures and marked zero.

4.9 SMALL CAPS (new)

Small caps style is often the preferred style for abbreviations in uppercase. Note that for phonetic notation, preformatted Unicode characters are strongly preferred. This styling does not apply to letters that are already small caps. Small caps can be simulated by using a smaller font size.

- **CSI 66**[**:***v*]**m** Display uppercase letters as small caps. Lowercase letters are not affected. If the current font does not support the small caps feature, there is no effect. Lowercase letters, CJK as well as preformatted characters of any kind are not affected, nor are inline images, like emoji. Uppercase digits (0-9) should also be covered and shown as lowercase digits (old-style digits). Variants: **:1** as described (default), **:999** leave uppercase as normal, but display lowercase letters as small caps, if there is a case mapping to uppercase for the letters and the font supports small caps (for capital letters). **:0** Display letters as normal.
- **CSI 67**[**:***v*]**m** Display letters as normal (default). Same variants as for **CSI 66**[**:***v*]**m** but the default is **:0**.

4.10 PROPORTIONAL WIDTH/LOWERCASE DIGITS (new)

Lowercase digits are often preferred in running text (which is mostly lowercase), but not in tabular-like contexts or arithmetic expressions.

- **CSI 69m** Lowercase digits (usually proportional width, if font is proportional width), if available in the font. Applies to (narrow) digits 0 to 9. Other digits are not affected. Often preferred in running text, but not in (numeric) tables nor calculations.
- **CSI 70m** Proportional width uppercase digits, if available in the font. Applies to (narrow) digits 0 to 9. Other digits are not affected. Does not apply to fixed-width fonts.
- **CSI 71m** Fixed width uppercase digits. Digits are often fixed width (and uppercase) also in otherwise "proportional" fonts. Applies to (narrow) digits 0 to 9. Other digits are not affected. Useful in tables or summations where numbers are lined up as well as arithmetic expressions and is also terminals style.

4.11 LINE SPACING (new, replacing SLS, SVS, SPI)

The line progression direction "movement" when there is a line break. Default, about 1,3 em.

- **CSI 74:***u***:***s***m** <u>Line spacing</u>. *u* unit as per SSU (0 is em, 1 to 8 per SSU, 9 resets to default 1,3 em line spacing regardless of *s*, which then can be omitted). *s* can be fractional (e.g., **CSI 74:0:1?5m** for 1,5 em line spacing). This replaces SLS (SET LINE SPACING), SVS (SELECT LINE SPACING), SPI (SPACING INCREMENT) which are proposed not to be used. If the unit is 'em', it is the em of the largest glyph in the displayed line. Extra space between paragraphs can be achieved by using PLD (also **ESC K**).

4.12 LINE INDENTS AND JUSTIFICATION (new, replacing JFY, QUAD)

Line indents can be used for first line intent for "ordinary" paragraphs, and for non-first line indents for bulleted/numbered lists. For bulleted/numbered lists, the bullet or the number can be in a "tab field" and have an HTJ (CHARACTER TABULATION WITH JUSTIFICATION, U+0089 or **ESC I**) just after the bullet/number part, and then a HT to the tab position set at the same position as the non-start line indent. That functionality of course requires that HTJ is interpreted. 'en' here is the 'en' at point of setting these.

The line indent control sequences should occur only at beginning of a paragraph. At other positions, the result is implementation defined. Note: line indents are *not* reset by **CSI 0m**. SLH, SLL, SPH and SPL should not be used since they are ill-defined: their "positions" are undefined.

- **CSI 75:***u***:***v*[**:***s*]**m** <u>First line</u> (of paragraph if *s* is 1 (default), after LS/VT/… if *s* is 2) <u>BOL</u> (beginning of line) <u>indent</u> (beginning of text, and after: LF, CR, PS, …; after LS, VT, … if *s* = 2). Distance from *default* BOL position (as set by preference setting or by context (such as window size, or cell size)). On left side for LTR, on right side for RTL, on top for vertical lines. Does not accumulate. Unit *u* as per SSU units (0 to 8), 9 for reset to default (value ignored and can be omitted). *v* is positive (or zero), can be fractional. If the *s* = 2 variant is not set after an *s* = 1 variant setting, the *s* = 2 setting inherits the *s* = 1 setting from **CSI 76:***u***:***v*[**:***s*]**m.**

- **CSI 76:***u***:***v*[**:***s*]**m** <u>Non-first line</u> (of paragraph if *s* is 1 (default), after LS/VT/… if *s* is 2) <u>BOL</u> <u>line indent</u> (after auto-line-break). Distance from *default* BOL position (as set by preference setting or by context (such as window size, or cell size)). On left side for LTR, on right side for RTL, on top for vertical lines. Does not accumulate. Unit *u* as per SSU units (0 to 8), 9 for reset to default (value ignored and can be omitted). If *s* = 1 (default), this also sets two tab stops that "belong" to the **CSI 76:***u***:***v***:1m** (*s* = 1)setting: one at the position given, and one 1 en (current font and size; additional spacing can be achieved by using various space characters) prior (in the character progression direction) to that position. These two tab stops are not altered via HTSA (see below). Default tab stops before (in the character progression direction) these two tab stops are ignored when "tabbing". *v* is positive (or zero), can be fractional. (Nit: hanging indents can be useful for terminal emulators as well.) If *s* = 2 variant is not set after *s* = 1 variant setting, the *s* = 2 setting inherits the *s* = 1 setting from **CSI 76:***u***:***v***:1m**.

- **CSI 77:***u***:***v*[**:***j*]**m** <u>EOL</u> (end of line) <u>line indent.</u> Distance from *default* EOL position (as set by preference setting or by context (such as window size, or table cell size)). On right side for LTR, on left side for RTL, on bottom for vertical lines. Does not accumulate. Unit *u* as per SSU units (0 to 8), 9 for reset to default (remaining two parameters are then ignored and can be omitted). *v* is positive (or zero), can be fractional.
  j = 0: (default) Flush lines to set BOL, as set by **CSI 75:***u***:***v*[**:***s*]**m** and **CSI 76:***u***:***v*[**:***s*]**m**.

j = 1: Flush lines (after BOL, or if tab stops are used then after last used tab stop) to set EOL position (space/tab characters, at end of line not counted).
j = 2: Centre line content on the available line between BOL, or latest used tab stop used, and EOL positions as set by the (**CSI 75:***u***:***v*[**:***s*]**m** or **CSI 76:***u***:***v*[**:***s*]**m** or HTSA) and **CSI 77:***u***:***v*[**:***j*]**m** control sequences (spaces/tabs at end of line are not counted).
j = 3: (replacing JFY) Stretch lines between BOL (or latest used tab stop) and EOL positions (if possible) *that are not at end of paragraph nor ends with VT/LS/FF/…* (otherwise, flush to BOL) to fill up the line space by widening spaces that are not of fixed size, though that should be avoided for spaces before/after words in certain 'cursive' scripts (e.g. Arabic) where the word is instead lengthened by inserting "tatweels" inside the word.
j=4 or *j*=5: No automatic line breaking, first align VT/LS/… separated lines in each paragraph on first occurrence of COMMA, FULLWIDTH COMMA or ARABIC DECIMAL SEPARATOR (*j*=4) or to FULL STOP or FULLWIDTH FULL STOP (*j*=5) (align with end of line if no occurrence of those characters in the line), then align the entire paragraph to end of line as set by **CSI 77:***u***:***v*[**:***j*]**m** (assuming that the line contents all fit).
The behaviour for line content that cannot fit in the line is implementation defined, but should normally use automatic line breaking.
Except for *j*=0, if there are HT/HTJ characters in the line, only the part of after the last HT/HTJ character in the displayed line is affected by the adjustment from that HT/HTJ character's "target" position, which is used as *temporary* BOL position, to EOL position.

4.13 ADVANCEMENT MODIFICATION (new, replacing SACS, SDCS, SSW)

Not recommended, but extra letter separation has been used for emphasis, esp. in German Fraktur, but also in other instances.

- **CSI 110**[**:***z*]**m** <u>Advancement modification</u> (times **CSI 111:***x***m** (spaces only) and **CSI 73:***a***:***b***m** modification) in the character progression direction after kerning. *z* can be fractional (**?** as decimal marker), affects spaces as well, **CSI 110:1m** resets (factor = **1**; short **CSI 110m**), no accumulation. E.g., **CSI 110:1?3m**abc**CSI 110:1m**d sets "abcd" with 1,3 times the nominal advance width of each letter, "stretching" the word. Factors other than 1 may break up ligatures and cursive joins, but for Arabic 'kashida' strokes may be inserted. Factors less than 1 should *not* be used, as that may result in (partial) overtyping with implementation defined display results. (As an extreme example, **CSI 110:0m**abc**CSI 110:1m**d sets "abcd" all at the 'same' position.) Changing the advance does *not* change the "en" and "em" unit sizes, nor tab stops. This replaces SACS and SDCS which are here proposed not to be used. The latter depended on separately set unit and were not reset by **CSI 0m**, were additive, not multiplicative, and were reset by line breaks.

- **CSI 111**[**:***x*]**m** <u>Space</u> (Unicode general category Zs characters) <u>advancement modification</u> (times **CSI 110:***z***m** and **CSI 73:***a***:***b***m** modification), in the character progression direction. *x* can be fractional (**?** as decimal marker); **CSI 111:1m** resets (factor = **1**; short: **CSI 111m**), no accumulation. Note that if the line is justified (**CSI 77:***u***:***v***:3m**), some spaces may be "stretched" a bit more than given here (but never less), due to line justification. Values less than 1 should not be used. Changing the space advance factor does *not* change the "en" and "em" unit sizes, nor tab stops. This replaces SSW which is here proposed not to be used. The latter depended on separately set unit and was not reset by **CSI 0m**, was additive, not multiplicative, and was reset by line breaks.

4.14 UNDERLINE (extended with variants)

ECMA-48 does not allow for several different types of underlines on the same piece of text, and even if one were to allow it, it would not be possible to have differently coloured underlines on the same piece of text (but see the section on UI underlines below, which are *not* part of the text styling, but part of the UI). Similarly for overline and strike-through. The underline is drawn just below the baseline. The underline should skip descenders, like those in "jgpqy" as well as letters with diacritics below. But do not skip LOW LINE, which nowadays is usually displayed on the baseline, not below the baseline, even though its origin in typewriters was for underlining.

- **CSI 4**[**:***v*]**m** <u>Singly underlined</u>. Variants: **CSI 4:0m** not underlined (same as **CSI 24m**), **CSI 4:1m** solid (medium, default), **CSI 4:2m** double thin solid lines (same as **CSI 21m**), **CSI 4:3m** wavy/curly, **CSI 4:4m** dotted, **CSI 4:5m** dashed (medium), **CSI 4:6m** double wavy, **CSI 4:7m** thin solid, **CSI 4:8m** bold solid, **CSI 4:9m** bold wavy, **CSI 4:10m** thin dashed, **CSI 4:11m** bold dashed, **CSI 4:12m** double thin dashed. Each variant of **CSI 4m** cancels current **CSI 4m**, if any, and cancels **CSI 21m**. Thin and thick is relative to current font size.
- **CSI 21**[**:***v*]**m** <u>Doubly underlined</u>. **CSI 21m** cancels **CSI 4m**, if any. Same as **CSI 4**[**:***v*]**m**, but with different default (**:2**).
- **CSI 24**[**:***v*]**m** <u>Not underlined</u>. Same as **CSI 4**[**:***v*]**m**, but with different default (**:0**).

4.15 OVERLINE (extended with variants)

The overline is drawn slightly above cap height. Used for some math notations, like decimal sequence repetition for rational numbers, but should still skip diacritics above capital letters.

- **CSI 53**[**:***v*]**m** <u>Overlined</u>. Same variants as **CSI 4m**, default is **:1**, medium solid line. Each **CSI 53**[**:***v*]**m** cancels current **CSI 53**[**:***v*]**m**, if any.

- **CSI 55**[**:***v*]**m** <u>Not overlined</u>. Same as **CSI 53**[**:***v*]**m**, but with different default (**:0**).

4.16 STRIKE-THROUGH (extended with variants)

Strike-through, or crossed-out, is commonly used to mark text that is planned to be deleted or otherwise outdated, wrong, but still shown for comparison with new text (that may be marked in some way to indicate that it is new), or just marking as closed (e.g., issue is closed, proposal rejected, or similar).

- **CSI 9**[**:***v*]**m** <u>Crossed out</u> (strike-through). Same variants as **CSI 4m**, default is **:1**, medium solid line. Each **CSI 9**[**:***v*]**m** cancels current **CSI 9**[**:***v*]**m**, if any.

- **CSI 29**[**:***v*]**m** <u>Not crossed out</u>. Same as **CSI 9**[**:***v*]**m** but with different default (**:0**).

4.17 EMPHASIS MARKING FOR CJK (extended with variants)

Traditionally, CJK does not use bold or italic/oblique but other ways of emphasising; and the “underline” is different (CJK glyphs have a lower baseline than non-CJK characters). Compare CSS (https://developer.mozilla.org/en-US/docs/Web/CSS/text-emphasis-style). Proposed here are some extensions to ECMA-48, to handle some variants that CSS covers. All of these should cancel “western” underline/overline, but not cancel strike-through/crossed-out.

- **CSI 60**[:*v*]**m** CJK underline (on the right side if vertical character progression direction). Same variants as **CSI 4m**, default is **:1**, medium solid line. (Negative variant value can be used for overline/left side line.)
- **CSI 61**[:*v*]**m** CJK double underline (on the right side if vertical character progression direction). Same as **CSI 60**[:*v*]**m**, but different default (**:2**). (Negative variant value can be used for overline/left side line.)
- **CSI 62**[:*v*]**m** CJK overline (on the left side if vertical character progression direction). Same variants as **CSI 4m**, default is **:1**, medium solid line. (Negative variant value can be used for underline/right side line.)
- **CSI 63**[:*v*]**m** CJK double overline (on the left side if vertical character progression direction). Same as **CSI 62**[:*v*]**m**, but different default (**:2**). (Negative variant value can be used for underline/right side line.)
- **CSI 64**[:*v*]**m** CJK stress marks (dot placed under/over (right/left side if vertical writing)). Variants: **CSI 64:0m** no CJK stress mark or line (same as **CSI 65m**), **CSI 64:1m** filled dot (●) under/right each CJK combining sequence (default), **CSI 64:1=m** filled dot over/left, **CSI 64:2m** filled sesame (﹅) under/right, **CSI 64:2=m** filled sesame over/left, **CSI 64:3m** hollow dot (○) under/right, **CSI 64:3=m** hollow dot over/left, **CSI 64:4m** hollow sesame (﹆) under/right, **CSI 64:4=m** hollow sesame over/left. Ruby text (see below, PTX) should be placed “outside” of these marks.
- **CSI 65m** Cancel the effect of the renditions established by parameter values 60 to 64. Each of 60 to 64 primary parameter values should cancel any of the five; including “itself” since the variant may change (solid, double, wavy, dotted, dashed; position; dot/sesame, filled/hollow).

Each of the six above cancel the “western” underline and overline, but not strike-through. “Western” underline/overline cancel the CJK emphasis marks.

4.18 FRAMING TEXT EMPHASIS (clarified and extended with variants)

Put an inline frame around a span of text. This styling is a bit like old Egyptian cartouches.

- **CSI 51**[:*v*[:*r*]]**m** Framed string. At line break, the framing is terminated (no line at EOL side) and restarted (no line at BOL side) on the new line. Cancelled by **CSI 51**[:*v*[:*r*]]**m**, **CSI 52**[:*v*[:*r*]]**m**, **CSI 54**[:*v*[:*r*]]**m** (which all may also start another framing). Line above is just above singly accented capital letters, like Å, É. Line below is just below descenders, for letters like “jgpqy”. Same variants for *v* as **CSI 4m**, default variant is **:1**, medium solid line.
r, default **0**, gives corner rounding; **:0** is non-rounded corners, **:1** is for 0,25 en radius rounded

corners, **:2** is for 0,5 en radius rounded corners, **:3** is for 0,75 en radius rounded corners **:4** is for 1 en radius rounded corners. If background colour is set at the same positions as encirclement, the colour change ideally follows the encirclement. If there is a font size change within a framed string, the effect is implementation defined.

- **CSI 52**[**:***v*[**:***r*]]**m** <u>Encircled string</u>. Same as **CSI 51**[**:***v*[**:***r*]]**m**, but the default for *r* is **4**.
- **CSI 54**[**:***v*[**:***r*]]**m** <u>Not framed, not encircled</u>. Same as **CSI 51**[**:***v*[**:***r*]]**m** and **CSI 52**[**:***v*[**:***r*]]**m**, but the default for *v* is **0**.

These are useful not only for Egyptian cartouche-like effect (which is an unusual use for these, but handy for explaining this), but also for such things like prompts in terminal emulators or grouping “special” letter sequences, or name labels. Compare framed “span”s in HTML.

4.19 COLOURING (extended and clarified, ‘named’ colours fixated in colour)

The original ECMA-48 assumed only two “colour planes”, background and foreground. Modern text applications (used with common higher-level protocols for document formatting) have several more planes. Also, some uses of ECMA-48 have already been extended with more “colour planes”. In particular, fill background colour and underline (etc.) colour planes.

User interface windows with ECMA-48 text is here proposed to have the following logical graphic “planes”, from “bottom” to “top”. However, an implementation need not work explicitly with colour planes, and there may be more logical colour planes than described here.

Colour values should be in the interval 0—255,75, negative values interpreted as 0 and larger values interpreted as 255,75. A value is then rounded down to an available value on the device.

0) Anything that is behind the window (like other windows and desktop background). Usually this is not seen at all (within the window in question) due to opaque colours on the higher planes, but some applications do allow transparency so that what is behind the window can be seen.

0.5) TV/video image; for Teletext-like applications.

1) Background fill colour or image. This may be (partially) transparent, even though it usually isn’t. May be settable in preferences.

1.5) Block background colours. Settable via FRAMED BLOCK, **CSI 114:…m** (see below, section 4.21). It is actually several planes if PTX table rows are used (see below, section 6).

2) Text background colour or better called text highlight colour if the default text background colour (plane 2) is fully transparent, which is common nowadays. Filling up the line height for the *full* line, even if the line height is varying within the line (usually horizontal but may be vertical for CJK and Mongolian script); from below descenders and up; or, for vertical text: the full width of the vertical line. Usually, but not necessarily, defaults to a solid colour (often white, or same as plane 1 colour), or fully transparent. The default may be settable in preferences. Colour can be set by **CSI 48**:….**m**. A *negative* transparency for the colours on planes 4 to 6 may “punch trough” this colour plane; e.g. **CSI 43;38:2::0:0:0:255?75=m** will make a “gold” background “punched through” by the text.

2.5) Text selection colours. Used for marking text as “selected”. Fully transparent for parts that are not selected. Some systems may have a separate “search match highlight” level. The colours for these are set by the system but may be settable in preferences. For a more ECMA-48 like solution, they can be set by control sequences proposed in the “UI text marking” section below; however, such UI text marking control sequences should never be part of a document.

3 and 3.5) Shadow planes. For character “shadows”. Colours set by **CSI 8***z*:….**m** (there can be multiple shadows, with different colours, all-around shadow is on plane 3.5). This kind of shadow is supported in HTML/CSS. Text decoration, like underlines, associated with shadowed text, should then also get a shadow.

4) Emphasis marks (CJK emphasis marks, the *lines* for underline/overline). Defaults to follow text foreground colour. Colour can be set by **CSI 58**:….**m**.

5) Text foreground colour (colour for the character glyphs). Usually defaults to black colour. Default may be settable in preferences. Apart from transparency (including blinking and concealed (transparent) characters) and negative image, the colour setting does not affect emoji (or other inline images/graphics, if such are permitted). Colour can be set by **CSI 38**:….**m**.

6) Text “decoration” (the *lines* for overstrike/strikeout, and for encircled/framed). Defaults to follow text foreground colour. Colour can be set by **CSI 68**:….**m**.

6.5) Maybe multiple colouring planes for spell and style checking underlines. Can be set by control sequences proposed in the “UI text marking” section below; such sequences should never be part of a document.

7…) Some systems allow for (temporary) popups (with styled text) overlaying the “real” text. From the point of view of this document, that is part of the UI, not the document display itself.

Implementations are not required to support all planes. E.g. an implementation may choose to omit the shadow plane and the text “decoration” plane. But the text background plane and the text foreground plane are required.

4.19.1 Text background colour (full colour as basis, clarified and extended)

Better referred to as text highlight colour, when using more than two colour planes, and the default colour for this is fully transparent. Each of *r*, *g*, *b*, *c*, *m*, *y*, *k* in 0—255,75; *t* in 0—255,75.
(0 is fully opaque, 255,75 is fully transparent). 0 is the default value for transparency, i.e. fully opaque. Unless it is negative, the parameter *i* is ignored (and can be omitted, but not the separators), so is the parameter *a*. With just two planes this is the colour behind the text. With more colour planes this is the text highlight colour (preferably with a default that is fully transparent). Usage hint: colours should be selected so that there is sufficient contrast between (total) background colour, and foreground colours (text, underlines/ overlines, string framing).

10-bit colour, if supported, can use fractional values (**?** as decimal marker): *x***?00** (for ‘00’ as last two bits), *x***?25** (for ‘01’ as last two bits), *x***?50** (for ‘10’ as last two bits), *x***?75** (for ‘11’ as last two bits), with *x* in 0—255. Rounding to these values may be applied. The decimal part is ignored for 8-bit colour. The colour “painted” should cover at least from nominal descender level to nominal first accent above capital level of the displayed line and may be vertically contiguous; similarly, for vertical lines and marks left/right of the base character and those may be horizontally contiguous.

- **CSI 48:0m** Reset background colour to default text background/highlight colour. The 0 can be omitted (**CSI 48m**). Same as **CSI 49m**. (Default is top of stack, if stacking is implemented.)
- **CSI 48:1m** Fully transparent text background/highlight (i.e. no highlight, the fill background is seen behind the text). Short for **CSI 48:2::0:0:0:255?75m**. Recommended default.
- **CSI 48:2:**[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Text background colour as RGB(T) (the separator here is colon).
- **CSI 48:3:**[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Text background colour as CMY(T) (the separator here is colon).
- **CSI 48:4:**[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Text background colour as CMYK (the separator here is colon).
- **CSI 48:5:***p*[**:***t*]**m** Text background/highlight colour from colour palette. *p* from 0 and up. Size of palette is implementation defined. Palette colours may be fixed by the implementation or be (partially) settable in preferences. If settable, various colour themes may be settable for the palette. There is also a possibility to set colour palette values via control sequences. If a transparency value is given, that overrides the transparency value from the palette.
- **CSI 48:6m** Copy current foreground colour to background colour. Not recommended. Useful (only) for making conversion from Teletext more convenient. Deprecated.
- **CSI 49m** Reset to default text background/highlight colour (may be settable in preferences).

4.19.2 Text foreground colour (full colour as basis, clarified and extended)

Each of *r*, *g*, *b*, *c*, *m*, *y*, *k* in 0—255,75; *t* in 0—255,75 (0 is opaque, 255,75 is fully transparent; negative value may be used to “punch through” text background/highlight colour, with given degree of transparency for foreground colour). 0 is the default value for transparency, i.e. fully opaque. Unless it is negative, the parameter *i* is ignored (and can be omitted, but not the separators), so is the parameter *a*. The transparency part of the foreground colour applies also to inline images/graphics (like emoji), which otherwise keep their colours, in a multiplicative manner. 10-bit colour can use fractional values (**?** as decimal marker): *x***?00**, *x***?25**, *x***?50**, *x***?75**, with *x* in 0—255.

Of course, for readability, the text foreground colour should be in sufficient contrast to the effective background colour (whether that is via text background/highlight colour or the plane 1 background). There is no automatic contrast guarantee.

- **CSI 38**[**:0**]**m** Reset foreground colour to default foreground colour. The 0 can be omitted (**CSI 38m**). Same as **CSI 39m**. (If stacking is implemented, “default” refers to top of stack.)
- **CSI 38:1m** Fully transparent foreground. Short for **CSI 38:2::0:0:0:255?75m**. Not recommended, but useful for conversion from Teletext. Deprecated.
- **CSI 38:2:**[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Foreground colour as RGB(T) (the separator here is colon).
- **CSI 38:3:**[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Foreground colour as CMY(T) (the separator here is colon).
- **CSI 38:4:**[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Foreground colour as CMYK (the separator here is colon).
- **CSI 38:5:***p*[**:***t*]**m** Foreground colour from colour palette. Uses the same palette as for background colour. *p* is in an implementation defined range, but not negative. If a transparency value is given, that overrides the transparency value from the palette.
- **CSI 38:6m** Copy current background colour to foreground colour. Not recommended. Useful (only) for making conversion from ISCII more convenient. Deprecated.
- **CSI 38:7m** Copy current all around shadow colour to foreground colour. Not recommended. Useful (only) for making conversion from ISCII more convenient. Deprecated.

- **CSI 38:8m** Optionally transparent foreground. Like **CSI 38:1m**, but can be “revealed”.
- **CSI 38:9:*b*:*p1*:**[*t1*]**:*p2***[**:***t2*]**m** Gradient glyph colouring. *p1* (start colour) and *p2* (end colour) are indices in the colour palette. *b* is an angle between **180=** (−180) and **180** degrees, 0 degrees is up. The angle gives the direction of the gradient, which starts and stops just within the “em box”. For gradient purposes, glyphs should be considered to be horizontally “centred” in the em box. *t1* and *t2*, if present, are transparency overrides.
- **CSI 39m** Reset to default text foreground colour (may be settable in preferences).

4.19.3 Short forms for colours (extended, and colours are given fixed values)

The colours here are a compromise and should be used for new or updated implementations. For a palette, possibly user settable (see section 4.21.7 below), use **CSI 38:5:*n***[**:***t*]**m** or **CSI 48:5:*n***[**:***t*]**m**. The colours for the short forms are *not* fetched from the palette, they are fixed colours.

Most terminal emulators nowadays use some variety of dull colours rather than clear colours for shortcut numbers in the 30 and 40 range, but are keeping the clear colours at higher numbers after the **CSI** (and, for compatibility, we follow that here). These colours are here fixed (just like the named colours for CSS), to get an (as close as possible) reliable colour output.

x in **3**, **4**: **CSI 3*y*:…** foreground; these colours are fully opaque colours
CSI 4*y*:… background; these may be partially transparent (transparency is implementation defined, may be settable via preferences or **APC** control)

CSI *x*0[:*t*]**m** is short for **CSI *x*8:2::0:0:0:*t*m**	Pure black ⬛ (CSS Black)
CSI *x*1[:*t*]**m** is short for **CSI *x*8:2::205:0:0:*t*m**	Dull red 🟥 (Xterm’s Red)
CSI *x*2[:*t*]**m** is short for **CSI *x*8:2::0:205:0:*t*m**	Dull green 🟩 (Xterm’s Green)
CSI *x*3[:*t*]**m** is short for **CSI *x*8:2::255:215:0:*t*m**	Dull yellow 🟨 (CSS Gold)
CSI *x*4[:*t*]**m** is short for **CSI *x*8:2::0:0:205:*t*m**	Dull blue 🟦 (CSS Mediumblue)
CSI *x*5[:*t*]**m** is short for **CSI *x*8:2::205:0:205:*t*m**	Dull magenta 🟪 (Xterm’s Magenta)
CSI *x*6[:*t*]**m** is short for **CSI *x*8:2::0:205:205:*t*m**	Dull cyan ■ (Xterm’s Cyan, ≈ CSS DarkTurquoise)
CSI *x*7[:*t*]**m** is short for **CSI *x*8:2::255:255:255:*t*m**	Pure white (CSS White)

Largely following existing implementations, we define some additional short forms for fixed background and foreground colours. This is a proposed addition, but most terminal emulators already implement some variety of clear/pure colours here.

x in **9**, **10**: **CSI 9*y*:…** foreground, reset by **CSI 39m**, *z* is **3**; fully opaque colours
CSI 10*y*:… background, reset by **CSI 49m**, *z* is **4**; these may be partially transparent (implementation defined)

CSI *x*0[:*t*]**m** is short for **CSI *z*8:2::105:105:105:*t*m**	Dark grey ■ (CSS DimGrey)
CSI *x*1[:*t*]**m** is short for **CSI *z*8:2::255:0:0:*t*m**	Clear red 🟥 (CSS Red)
CSI *x*2[:*t*]**m** is short for **CSI *z*8:2::0:255:0:*t*m**	Clear green 🟩 (CSS Lime)
CSI *x*3[:*t*]**m** is short for **CSI *z*8:2::255:255:0:*t*m**	Clear yellow 🟨 (CSS Yellow)
CSI *x*4[:*t*]**m** is short for **CSI *z*8:2::0:0:255:*t*m**	Clear blue 🟦 (CSS Blue)
CSI *x*5[:*t*]**m** is short for **CSI *z*8:2::255:0:255:*t*m**	Clear magenta 🟪 (CSS Magenta)
CSI *x*6[:*t*]**m** is short for **CSI *z*8:2::0:255:255m:*t***	Clear cyan ■ (CSS Cyan)
CSI *x*7[:*t*]**m** is short for **CSI *z*8:2::220:220:220:*t*m**	Light grey ⬜ (CSS Gainsboro)

CSI *x***8**[**:***t*]**m** is short for **CSI** *z***8:2::169:169:169:***t***m**	Medium grey ◼ (CSS DarkGrey)
CSI *x***9**[**:***t*]**m** is short for **CSI** *z***8:2::255:140:0:***t***m**	Orange 🟧 (CSS DarkOrange)

4.19.4 Text emphasis mark colour (new, but has implementations)

Underline, overline, CJK emphasis lines/marks. *p*, *r*, *g*, *b*, *c*, *m*, *y*, *k*, *t*, *i*, *a* as above.

- **CSI 58**[**:0**]**m** Reset text emphasis mark colour to follow foreground colour. Default. The 0 can be omitted (**CSI 58m**).
- **CSI 58:1m** Fully transparent text emphasis. (Not recommended.) Shadow, if any, will still be visible.
- **CSI 58:2:**[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Text emphasis colour as RGB(T) (the separator here is colon).
- **CSI 58:3:**[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Text emphasis colour as CMY(T) (the separator here is colon).
- **CSI 58:4:**[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Text emphasis colour as CMYK (the separator here is colon).
- **CSI 58:5:***p*[**:***t*]**m** Text emphasis colour from colour palette. Uses the same palette as for background colour.
- **CSI 59m** Reset emphasis colour to follow text foreground colour (i.e. reset any colour set by **CSI 58:…m** or **CSI 68:…m**).

4.19.5 Text overstrike and string framing colour (new)

Overstrike (strike-through), encircling lines, framing lines (not for block frames), margin lines. *p*, *r*, *g*, *b*, *c*, *m*, *y*, *k*, *t*, *i*, *a* as above.

- **CSI 68**[**:0**]**m** Reset text crossed-out/strike-through and string framing colour to follow foreground colour. The 0 can be omitted, **CSI 68m**.

- **CSI 68:1m** Fully transparent overstrike and framing lines. (Not recommended.)

- **CSI 68:2:**[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Text overstrike and string framing colour as RGB(T) (the separator here is colon).

- **CSI 68:3:**[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Text overstrike and string framing colour as CMY(T) (the separator here is colon).

- **CSI 68:4:**[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Text overstrike and string framing colour as CMYK (the separator here is colon). (Fully opaque.)

- **CSI 68:5:***p*[**:***t*]**m** Text overstrike and string framing colour from colour palette. Uses the same palette as for background colour.

4.19.6 Text shadow colours (new)

Outline and shadow are included as styles in ISCII, and shadow is supported (with lots of variation) via CSS. The shadow styling can be used to produce (attached-to-glyph) shadow, outline and bevel effects. Text shadows are also useful (and is used when not using text background) for such things as subtitles, that should be readable for a large variety of background colours (in a video image).

Note that the full glyph casts a shadow, that can be seen through transparent glyphs. Glyphs can have several shadows, usually of different colours. How overlapping shadows display is implementation defined. Also transparent, including fully transparent, glyphs cast shadows. The shadows here *need not* behave exactly like “real” shadows, despite the name.

e is an angle between **180=** and **180** degrees, 0 degrees is "light" from top of "paper".
d in **1** (0.01em shadow), **2** (0,02em), ..., **99** (0,99em shadow)
f in **0** (no blur, default), **1** (0,01em blur), ..., **99** (0,99em blur); *r*, *g*, *b*, *c*, *m*, *y*, *k*, *t* as above.

Note that the second parameter (*d*) and the seventh parameter (*f*) differ in interpretation here compared to how these parameters are interpreted in other colour control sequences.

- **CSI 80**[**:0**]**m** Cancel all-around shadow, directional shadow and its "counter-shadow". All-around shadow is on colour plane 3.5. The all-around shadow does *not* itself cast a shadow.
- **CSI 80:2:***d***:***r***:***g***:***b*[**:***t*[**:***f*]]**m** All-around shadow colour as RGB(T).
- **CSI 80:3:***d***:***c***:***m***:***y*[**:***t*[**:***f*]]**m** All-around shadow colour as CMY(T).
- **CSI 80:4:***d***:***c***:***m***:***y***:***k*[**:***f*]**m** All-around shadow colour as CMYK. Fully opaque.
- **CSI 80:5:***d***:***p*[**:**[*t*][**:***f*]]**m** All-around shadow colour from palette. *p* is palette index.
- **CSI 80:6:***d*[**:***f*]**m** Copy current foreground colour to all-around shadow colour. Not recommended. Useful for making conversion from ISCII more convenient.
- **CSI 80:9:***d***:***b***:***p1***:**[*t1*]**:***p2*[**:**[*t2*][**:***f*]]**m** Gradient all-around glyph shadow colouring. *p1* (start colour) and *p2* (end colour) are indices in the colour palette. *b* is an angle between **180=** and **180** degrees, 0 degrees is towards top of "paper". The angle gives the direction of the gradient, which starts and stops just within the "em box" of each (base) character. For gradient purposes, glyphs should be considered to be horizontally "centred" in the em box.
- **CSI 81**[**:0**]**m** Cancel directed shadow, and its "counter-shadow" (if any).
- **CSI 81:***e***:2:***d***:***r***:***g***:***b*[**:***t*[**:***f*]]**m** Directional shadow colour as RGB(T).
- **CSI 81:***e***:3:***d***:***c***:***m***:***y*[**:***t*[**:***f*]]**m** Directional shadow colour as CMY(T).
- **CSI 81:***e***:4:***d***:***c***:***m***:***y***:***k*[**:***f*]**m** Directional shadow colour as CMYK.
- **CSI 81:***e***:5:***d***:***p*[**:**[*t*][**:***f*]]**m** Directional shadow colour from palette.
- **CSI 81:***e***:6:***d*[**:***f*]**m** Copy current foreground colour to directed shadow colour. Not recommended. Useful for making conversion from ISCII more convenient.
- **CSI 81:***e***:9:***d***:***b***:***p1***:**[*t1*]**:***p2*[**:**[*t2*][**:***f*]]**m** Gradient directional glyph shadow colouring. *p1* (start colour) and *p2* (end colour) are indices in the colour palette. *b* is an angle between **180=** and **180** degrees, 0 degrees is up. The angle gives the direction of the gradient, which is just within the em box of each (base) character. For gradient purposes, glyphs should be considered to be horizontally "centred" in the em box.
- **CSI 82:…m** "Counter-shadow". Same parameters as for **CSI 81:***e***:…m**, but the inherited direction is 180 degrees opposite to the (just set) direction for **CSI 81:***e***:…m**. Using a "counter-shadow" can be used to get a bevel-like or embossed effect.

Note: If using **CSI 80:...m** and the foreground colour is same as the (fill or text) background opaque colour, that can give an outline effect. Multiple shadows, and with the foreground colour same as the (text or fill) background opaque colour, that can give an embossed effect, depending on colour selection for the shadows. Compare the use of shadows in HTML/CSS, as illustrated in:

https://codepen.io/daryl/pen/yAuGj, (note `background-clip:text`); https://www.midwinter-dg.com/permalink-7-great-css-based-text-effects-using-the-text-shadow-property_2011-03-03.html.

4.19.7 Colour palette (extended with capability to be set via control sequences)

The colour palette, accessible via **CSI** ***s*****8:5:***n*[**:***t*]**m** (*s* in 3, 4, 5, 6) as well as gradient colouring as well as UI text colours, holds a set of colours, and a colour is accessible by an index number (*n*). These colours may be settable via preferences or installation of a colour palette. Another possibility is to have **CSI** ***s*****8:***u***:***n***=**:*d*:*e*:*f*:*g*[**:***a***:0**]**m** (*s* in 3, 4, 5, 6 but is ignored, *u* in 2 (RGBT), 3 (CMYT), 4 (CMYK)) set the colour palette at index *n* to the colour given by the rest of the arguments, *without changing the current colour setting*, only the palette. An implementation may limit the portion of the palette that is settable. Note: changes to the palette are *not* reset by **CSI 0m**. If the colour palette at index *n* is changed during the use of that palette index, the effect is implementation defined. It may take effect immediately or be take effect when that palette index is next explicitly used (or anywhere there-between). The palette colour settings do not affect the fixed colours available via the short forms for predefined colours. (The Linux console uses an **OSC** command for this kind of setting of the palette colours.)

E.g. https://www.fossmint.com/nord-modern-design-color-theme-palette-for-your-terminal/ has 16 pastel colours that could be set as the colours of index 0 to 15 of the palette (if the implementation allows such setting); this does *not* change the colours of the "named" colours in ECMA-48 which we have fixed here. https://linuxconfig.org/the-best-linux-terminal-color-schemes-for-2019 has a list of various colour palettes, suitable for (a part of) the palette.

At indices 16 to 255 the conventional colours should be stored, and be protected from change.

4.19.8 Negative image (clarified)

The negative image applies to colour planes 1 to 6, just where the affected text is displayed, and should apply to inline images/graphics (like emoji) which should also be turned into negatives (as in photo negative). Often **CSI 7m** has erroneously been interpreted as switching background and foreground colours. That would not work for transparency, nor for planes other than 2 and 5; and, importantly, that is unlikely to produce a negative image.

Negative image should *not* be used for marking "selected" (for editing, e.g. copy) text. Marking "selected" text is better done, as common today, by a more aesthetic change of background colour (though usually not via control sequences, but see below on UI colouring; on colour plane 2.5, seeing to that the contrast to the foreground colour(s) for the "selected" text is sufficient; in extreme cases maybe "negating" some or all of the foreground colours (planes 4 to 6) when there is a 'selected text' colour background).

- **CSI 7**[:*f*]**m** Negative image (if *f* is **0**). Interpret given CMY values as if they were RGB values (not converting), and given RGB values as CMY values, that way creating a negative (no change or reinterpretation of the transparency). CMYK values are first converted to CMY by adding the *K* value to each of the values *C*·(255,75−*K*)/255,75, *M*·(255,75−*K*)/255,75, *Y*·(255,75−*K*)/255,75; all colours specified as CMYK are fully opaque. Transparency is not affected. Repeat with same *f* has no effect (i.e., no toggling).
  f, default value **0**, can be fractional (with **?** as decimal marker), value between 0 and 1 inclusive. *f* is a factor to use on each RGB colour value *c* (after mapping ECMA-48 value to the

range [0, 1], (255,75−*c*)/255,75, 0.0 is fully negative colour, 1,0 is fully positive colour), final colour c_1, for each of R, G, B (*c*) is computed as $c_1 = f \cdot c + (1-f) \cdot (1-c) = 2 \cdot f \cdot c + 1 - c - f$.
- **CSI 27**[:*f*]**m** Positive image (if *f* is **1**). Reset to normal interpretation of colours. Same as **CSI 7**[:*f*]**m**, but the default for *f* is 1.

4.19.9 Concealed (censored) characters (extended with variants)

"Hide" characters. Not recommended. The concealed text may still be visible if the text is "selected" (for editing, e.g., copying by a copy-paste operation; depending on how "selected" is visibly marked), and a subsequent paste will also reveal the characters. Thus, one should not use "concealed" (nor any other form of transparent) for passwords or other text that is meant not to be extractable. This was not a concern for original ECMA-48, but it is in a modern setting. Indeed, it is common not to show characters in a password at all (Unix approach), or hide the password by displaying substitute characters, like bullets (web page approach).

- **CSI 8**[:*v*]**m** Concealed/censored characters. Text spans are at planes 3 to 6 temporarily fully transparent. This overrides any set or changed colours for the text. Variants: **:0** cancel concealed characters, **:1** (default) fully transparent as described, **:2** instead blur or **:3** pixelate the text to a degree that the text is unreadable, **:4** replace with an Åg-height line (in the text foreground colour at starting point; still cut and paste can reveal the text).
- **CSI 28**[:*v*]**m** Cancel "concealed characters" effect. Same as **CSI 8:0m**. Variants as above.

4.19.10 Blinking (extended with variants)

Cyclically varying transparency (between set transparency and 255,75 (fully transparent)) for text spans and at colour planes 3 to 6. If this is used in a context where inline images are allowed (e.g. emoji), the images in the span blink as well. It should be noted, in comparison, that in HTML/CSS blinking text is deprecated, and most modern browsers do not implement blinking text (but that can be circumvented by JavaScript code or by using GIFs). Also note, in comparison, that some terminal emulators have blinking text turned off by default. Blinking can still be useful to draw attention to things like "still running" ("heart-beat"), urgent errors/alarms, and the like; but without needing to redraw *all the time* with different transparency for the blinking part.

- **CSI 5**[:*v*[:*w*]]**m** Slowly blinking (approx. 1,5 Hz to 2 Hz). Cancels rabid blinking. Variants: **CSI 5:0m** steady (same as **CSI 25m**), **CSI 5:1m** slow blinking, **CSI 5:2m** fast blinking. Second variant (:*w*): **:1** sinusoidal, **:2** triangular, **:3** trapezoidal, **:4** rectangular (default).
- **CSI 6**[:*v*[:*w*]]**m** Rapidly blinking (approx. 2,5 Hz to 3 Hz). Cancels slow blinking. Same variants as for **CSI 5**[:*v*[:*w*]]**m**, but the default for *v* is **2**.
- **CSI 25**[:*v*[:*w*]]**m** Steady (not blinking). Same variants as for **CSI 5**[:*v*[:*w*]]**m**, but the default for *v* is **0**.

4.19.11 Fade control (new)

Teletext allows a page to be made transparent, except parts specially marked. This is intended to allow important parts of a page be shown even if the overall page is not shown, instead showing the background image (colour plane 0 or 0.5), in the case of Teletext, the TV image. The intended use is for "flash news" and subtitles to be shown when otherwise watching the TV program. In the Teletext version this allows the rest of the page to fade (completely), except the subtitles (or news flash) which do not fade. (Note that a Teletext page otherwise covers the entire TV image.) In Teletext, whether to fade or not is controlled by control bits in the Teletext protocol.

- **CSI 112:*f*m** Fade control factor. *f* can be fractional (with **?** as decimal marker), value between 0 and 1 inclusive. Factor to use on transparency value (after mapping ECMA-48 value to the range [0, 1], (255,75−*t*)/255,75; 0,0 is fully transparent, 1,0 is fully opaque) when set by preference to use this factor. Default is not to use this factor. Final transparency factor is computed as 1−*p*·(1−*f*), where *p* = 1 (from the preference setting) means to use the fade control factor, *p* = 0 means don't use the fade control factor (default); values between 0 and 1 can be used also for the preference setting (this allows for intermediate fades). In a TV-like setting, and *p* = 1, using factor *f* = 0 will show the TV background (colour plane 0), factor *f* = 1 will show the text. Affects both text background (all backgrounds, planes 1 to 3.5), text decorations, text itself including inline images/graphics, i.e. planes 1 to 6. The nominal computed total transparency, in [0, 1] with 1 as opaque, is multiplied with (1−*p*·(1−*f*)) giving a final transparency for planes 1 to 6.

4.20 MARGIN LINE (new)

A line in the margin for a displayed line of text may be used as a kind of emphasis marking. It is also popular as change marks ("change bars"). Another use for this kind of line is for block quotes (marking one or more 'paragraphs' (as defined here) in entirety, usually also indenting the lines).

- **CSI 78**[**:**[*v*][**:***p*]]**m** Line in the margin before the beginning-of-line position. Positioned just outside ("before") the minimum of **CSI 75:…m** and **CSI 76:…m** setting. **CSI 78:0m** terminates. Displayed for lines where (non-zero) **CSI 78:*v*m** is in effect, at about 1 en to 1 em before the beginning of lines. Same variants (*v*) as **CSI 4m**, default is **:8**, bold solid line. If there are several different non-zero **CSI 78:…m** in effect for one display text line, the displayed effect is implementation defined. *p* is an index in the colour palette (see below) for the colour of the margin line. Default is current text foreground colour. If started in a table cell, the margin line is inside the cell (at the current nesting level, see PTX below), rather than in the "page" margin.
- **CSI 79**[**:**[*v*][**:***p*]]**m** Line in the margin after end-of-line position. Positioned just outside ("after") the **CSI 77:…m** setting. **CSI 79:0m** terminates. Displayed for lines where (non-zero) **CSI 79:*v*m** is in effect, at about 1 en to 1 em after the end of lines. Same variants (*v*) as **CSI 4m**, default is **:8**, bold solid line. If there are several different non-zero **CSI 79:…m** in effect for one physical text line, the displayed effect is implementation defined. *p* is an index in the colour palette (see below) for the colour of the margin line. Default is current text foreground colour. If started in a table cell, the margin line is inside the cell (at the current nesting level, see PTX below), rather than in the "page" margin.

Note that the margin lines are displayed only in the margin of lines (after line wrapping) where the margin line styling is in effect; it is *not* just considering explicit line breaks, but also automatic ones.

4.21 FRAMED BLOCK (new)

Framing a block allows for paragraphs (and row cells) to be framed and get a different background colour (colour plane(s) 1.5, see below). These control sequences should occur "between" paragraphs (or row cells), if they occur at other locations the effect is implementation defined. The margin positions of the lines are just outside the default start position and default end position of lines (line indent control sequences do not affect the framed block line positions); if it is a row cell (see PTX below) that is framed, the lines are drawn at the cell boundaries. Compare framed "div"s in HTML. If multiple framed block lines are co-located (from adjacent framed blocks), the displayed result is implementation defined.

- **CSI 114**[**:**[*v*][**:**[*p1*][**:**[*b*][:[*p2*][:*c*]]]]]**m** Start frame of paragraphs. Same variants (*v*) as for **CSI 4m**, default is **3**, medium solid line, *p1* is the palette (see below) index for the frame colour (default: current text foreground colour), *p2* is the palette index for the frame fill colour, default is transparent (colour plane(s) 1.5). *b* a value in the range 0—15, default 15, read as a bit pattern indicating which sides get the line; bit 1: line progression direction lead side, bit 2: character progression lead side, bit 3: character progression terminating side, bit 4: line progression terminating side. Sides not listed inherit line setting from immediately preceding **CSI 114**…**m** if any (there may be up to four in a row, with different values for *v* and *b*; only the last value for *p2* and *c* take effect), otherwise no line. Terminated by **CSI 114**…**m** which also starts another block frame unless it is *directly* after another **CSI 114**…**m**, by **CSI 115m**, and by end of cell. If pagination is supported, and a framed block crosses a page break (automatic or explicit), there is no block line drawn at the page break. *c*=0 allow automatic page breaks within the block (should still avoid typographic widows/orphans), *c*=1 (default) avoid automatic page breaks within the block.
- **CSI 115m** End frame of paragraphs. If the block framing was started in a row cell (see PTX below), the block frame closes automatically at end of that cell. There may thus be multiple levels of block frames, one "outer", and one per row level (there may be rows within cells; see PTX below). Through this mechanism, cells (from PTX rows) may be framed and coloured (the entire cell, not just the text portion, if started at the beginning of the cell (just after a **CSI 1\** or a **CSI 2\**) and auto-terminated at cell end, at a **CSI 2\** or a **CSI 0\**, ideally joining lines of layoutwise adjacent cells, also if in different rows).

5 Resurrecting CHARACTER TABULATION SET ABSOLUTE – HTSA (old but improved, replacing HTS, CTC5, TBC3)

CHARACTER TABULATION SET ABSOLUTE (HTSA) has the control sequence **CSI** *t1*;*t2*;…;*tn* **SP N** (note the space before the capital **N**, it is part of the control sequence), where each *ti* is a position in the line (from the beginning position of the line, not counting current line indents). HTSA was deprecated in the fourth edition of ECMA-48 (for unknown reasons), and it does have a flaw (which we will fix here). But the other way of setting tab stops in ECMA-48 (HTS) has a *much* more serious flaw: one must move (by printing something, likely spaces) to the positions where one wants to set the tab stops. Besides outputting something (which is often an absolute no-no for setting tab stops), the positioning is also unreliable, and depend on the advance widths of glyphs (or spaces), which vary in a "proportional" font and between fonts. HTSA sets the tab stops without anything getting printed, and the positions are not dependent on individual glyph's advance widths. Thus, here we instead suggest that HTS and CTC are not to be used in favour of resurrecting HTSA. VTS, for setting line tab stops, is in practice already unused. Before the first HTSA, the default is stops every 4 en to 8 en of the *default* font size (implementation defined, may be part of preferences, changing the ("global" to the document) default font size preference may also change the position of the default tab stops).

Each HTSA replaces all previously set HTSA tab stops. **CSI SP N** (or **CSI !N** (see below); empty list of positions) in effect resets to using the default tab stops (but keeping the **CSI 76:*u*:*v*m** tab stops). Until the last, in the character progression direction, HTSA or **CSI 76:*u*:*v*m** tab stop, they overshadow the default tab stops.

But HTSA does have a flaw (or two). The unit was set by SSU, and we have already proposed not to use other control sequences that depended on SSU as well as not using SSU itself. So, in the resurrected version, we ignore the SSU, and have units associated with each of the tab position values. The default unit is 'en' (code 0). The unit codes (0 to 8) are the same as for SSU (0 is 'en' of the current font, current size, not counting **CSI 73:….m** setting). The unit code is written as a prefix to each value, with a colon between the unit code and the value; if there is no colon separator, it is just the value, with the unit 'en'. Fractional values are allowed with **?** as decimal marker. Maybe the "**SP N**" should be replaced by "**!N**" to indicate that it is the resurrected version.

- **CSI** *stoplist***!N** <u>Set tab stops according to the stoplist</u>. Tab stops set by a previous HTSA are removed. The stop list is a semicolon-separated unordered list of positions from the default start of line in the character progression direction. Positions are given as *unitcode*:*value*, where the unit code is from the SSH unit codes, and the value is a (possibly fractional) value indicating the distance (when combined with the unit) from default start of line in the character progression direction. The unit can be omitted, and the unit code is then **0** (en).

E.g., **CSI 0:5;0:11?5;0:15!N** sets tab stops at 5 en, 11,5 en and 15 en from the default beginning position of lines (in the character progression direction; not considering **CSI 76:….m** setting). Since en is used as unit, one needs to associate the values with the current "absolute" value of en or convert the position values for the tab stops to use a non-en unit. The tab positions may be sorted internally, to make their use (internally) easier.

Note that using en as unit in an HTSA control sequence can be convenient, and while already set tab stops must not be changed even if the size of en changes, follow-on HTSA control sequences are affected by a change of en, if they are using the en unit.

The tab positions need not be ordered in the HTSA control sequence, but the positions (when computed) should not be closer to each other than 1 em (2 en). The values (with unit) are each an offset from the default *beginning* of line (in the character progression direction), not from the previous tab stop (hence "absolute"). Default tab stops 1 em or more after, in the character progression direction, the furthest explicitly set by HTSA are still in effect. The HTSA control sequence should occur only at beginning of a line. At other positions, the result is implementation defined.

6 SELECT PRESENTATION DIRECTIONS – SPD

ECMA-48 has a mechanism for setting text presentation direction, SELECT PRESENTATION DIRECTIONS (SPD). An application may also have a (global) preference setting text direction, unless SPD is implemented and overrides that preference. SPD selects the line and character progression directions. SPD has the format **CSI** ***c*****;0 SP V** (*c* is a code for which directions are set; the **0** here can be omitted). There is a space before the terminating **V**, but using space in the control sequences is not ideal. Here we recommend using "**!V**" instead of "**SP V**" (space before the V).

For the Unicode bidi algorithm, PDF and PDI should be interpreted as equivalent, LRE should be interpreted as LRI, RLE should be interpreted as RLI, FSI should be interpreted as LRI if the character progression is left-to-right and as RLI if the character progression is right-to-left (i.e., *no* dependence on "first strong bidi" character for FSI, since direction needs to be predictable). In addition, **CSI 0]** and **CSI 0[** are equivalent to PDF (U+202C), **CSI 1]** is equivalent to LRO (U+202D),
CSI 2] is equivalent to RLO (U+202E), **CSI 1[** is equivalent to LRO (U+202D) if the current direction is RL and equivalent to RLO (U+202E) if the current direction is LR. Whether the legacy (ECMA-48) bidi control sequences are interpreted is implementation defined. This limits the "range" of the original ECMA-48 5th edition bidi controls to a paragraph; a limitation originally not present.

With the (new) SPD the following text display directions, and bidi on/off can be set:

- **CSI 0!V** Character progression left-to-right, for most scripts including CJK/Hangul horizontal; **no** bidi processing and *no* automatic cursive joining or ligatures for right-to-left scripts, but see **CSI 8!V** below. Glyphs for Mongolian characters are rotated 90° clockwise (in some fonts they may already be so rotated) and are *not* cursively joined. This covers also so-called "visual order" (and pre-joined glyphs for Arabic) for right-to-left scripts.
- **CSI 1!V** CJK/Hangul vertical, line progression right to left, **no** bidi processing (no cursive joining). Characters that are non-CJK, non-Hangul, non-Mongolian, non-wide, but are strong left-to-right or are symbols get their glyphs rotated 90° clockwise, strong right-to-left characters get their glyphs rotated 90° counter-clockwise.
- **CSI 2!V** Mongolian vertical, line progression left to right, no bidi processing. Characters that are non-CJK, non-Hangul, non-Mongolian, non-wide, but are strong left-to-right or are symbols get their glyphs rotated 90° clockwise, strong right-to-left characters get their glyphs rotated 90° counter-clockwise. Cursive joining is done for Mongolian characters.
- **CSI 3!V** Character progression right-to-left, (new) *with* Unicode bidi processing. Mongolian characters are considered bidi strong RTL and their glyphs are rotated 90° clockwise. Note that LTR scripts are still LTR; this SPD setting only sets the "paragraph direction" and leaves the right-to-leftness to the bidi algorithm. Cursive joining, and ligatures, are done for scripts that has that feature (and the script is supported by the implementation).
- **CSI 8!V** (new) Character progression left-to-right, *with* Unicode bidi processing. Mongolian characters are considered bidi strong RTL and their glyphs are rotated 90° clockwise.
- **CSI 10!V** (new) Within a paragraph, every odd line left-to-right, every even line right-to-left. Boustrophedon. For some ancient texts, where glyphs are mirrored when going right-to-left. Extremely rare use; included mostly to illustrate that it is possible to handle here.
- **CSI 99!V** (new) Reset to default directions, per preference setting.

Through bidi processing, a portion of a paragraph's text can be displayed in the character progression direction "opposite" to that set by SPD. Note that a paragraph's "bidi direction" is *never* derived from first "strong bidi" character in the paragraph, it is set from SPD (or its default). The START DIRECTED STRING/START REVERSED STRING control sequences are not to be used in favour to the Unicode bidi control characters. For automatic line breaking, LS (or VT or FF) shall automatically temporarily be inserted *before* bidi processing. Thus, glyph (including ligaturing and cursive shaping) widths must be computed before bidi processing. Note that escape sequences, control sequences and control strings must be interpreted before bidi processing. Exactly how style ranges are represented is implementation defined; but which characters have which style shall not be altered by bidi processing. An embedded graphics shall work as if it is a single control character (or an OBJECT REPLACEMENT CHARACTER) for bidi processing. Likewise for each math expression. Math expressions are out of reach for the bidi algorithm and for any case mapping.

If line progression direction is changed, it is per page (beginning of text or explicit FF (FORM FEED), if FF retains its original interpretation). If line progression is not changed, but character progression within lines are changed, the change is per "paragraph" (between hard line breaks, except VT/LS/FF). The change code should be at the beginning of the part of text affected. If not, the change will take effect only after next explicit break (FF or automatic page break for line progression changes, hard line break except VT/LS/FF for character only progression changes).

7 PARALLEL TEXTS – PTX (clarified and extended)

PTX, as specified in ECMA-48 5th ed., have two completely different uses. While both are "parallel texts" in some sense, one is most easily conceived of as a table row, and the other is for so-called Ruby annotations as used in Japanese and Chinese (though for Chinese there is a fallback to use a parenthetical instead). Unfortunately, they start with the same control sequence in ECMA-48 5th ed. Note: As a singular exception, PTX/table row does have a nesting structure.

7.1 Multiple texts in a single (table) row (extended with width indications)

This defines a single table row with (formatted) text content in each cell or the row. There is no automatic framing. If a cell has no data, a negative width indication and lines up with a cell in a directly preceding row, if any, those cells are merged.

- **CSI 1**[**;***u***:***s*]\ <u>Beginning of first text (of the list of parallel texts) (i.e., start of first cell in a row).</u> *u***:***s* gives the initial (minimum) row advance (as unit and measure); default is current line advance as set by **CSI 74:***u***:***s***m**. All bidi controls are closed.
- **CSI 2**[**;***u***:***s*[**;***v*]]\ <u>End of previous text and beginning of next text (i.e., cell boundary).</u> *u***:***s* gives the width of the cell (in the character progression direction) just terminated, default is width of content without automatic line breaking. *v*=0: flush the set of lines vertically (horizontally for vertical lines) to the "lines beginning" of the cell (default); *v*=1: flush the set of lines to the "end of lines" (in the cell); *v*=2: centre the set of lines vertically (horizontally for vertical lines) on the available space; *v*=3: row span cell, content is concatenated with the cell above in the column. All SGR styling is reset to that at the start of the row. All bidi controls are closed.
- **CSI 0**[**;***u***:***s*[**;***v*]]\ <u>End of list of parallel texts (i.e. end of cell and end of row).</u> *u***:***s* as above. *v* as above. The syntax for this use of PTX is **CSI 1\***principal text*(**CSI 2**[**;***ux***:***sx*[**;***vx*]]**\***supplementary text*)+**CSI 0**[**;***uy***:***sy*[**;***vy*]]\. This corresponds in principle to the HTML: **<tr><td>***principal text***</td>**(**<td>***supplementary text***</td>**)+**</tr>**, but with cell widths (heights for vertical lines). All SGR styling is reset to that at the start of the row. All bidi controls are closed.

Note that all Unicode bidi controls are closed at end of paragraph, and here also at the PTX controls.

The layout is unspecified in ECMA-48, but multi-column display is what is mentioned ("*the strings may* [here interpreted as "shall", to get a stringent, and generally useful, interpretation of PTX] *be presented in successive lines in **parallel columns**, with their beginnings aligned with one another and the shorter of the paragraphs followed by an appropriate amount of "white space"."*). This is (was) intended for translations or similar but is actually more general and can be used for any kind of table row. Note that cell texts may have several instances of the Ruby variant of PTX (see next section), and also nested use of this kind of PTX parallel texts (i.e., table rows within a table cell). Tab stops and line indents are inherited and are interpreted from "beginning of cell" for each cell.

Automatic line breaking is done on text within each cell (if given a width), and the cell height (width for vertical lines) is the maximum cell height in the row. The beginning and end of lines should have about 1 en margin to the cell border. Multiple contiguous rows (with explicit line break between) form a table, but there is no automatic alignment of cell borders, and columns form only if there is alignment. There are no automatic lines drawn between table cells, but framed blocks (see section 4.21 above) can be used to both frame a cell and give it a background colour.

Note that **CSI 1\**…**CSI 2\**…**CSI 2\**……**CSI 0\** (PTX) is one of the few mechanisms in ECMA-48 that specifies nesting; the others are the bidi controls. Note also that bidi bidi algorithm needs to be applied in isolation for PTX cells, and that each a sequence of PTX rows is a single "object" for the bidi algorithm outside of the sequence of PTX rows. I.e., each table row, as a whole, must be atomic (i.e., handled as a single, bidi neutral, character) for bidi processing. A table row's cells are laid out according to the current character presentation direction. And the line and character progression directions are inherited to each cell, but can be set per cell via SPD per cell. Further, bidi, if enabled, applies to each cell of the row, in isolation.

There must be a line break between each **CSI 0\** and **CSI 1\** in order to create a table of rows, where the rows are laid out in the line progression direction. A (single) PTX row with no line break before nor after, will be an inline table row; note that a cell in such a row can have a table. Cells in each row should have consistent widths, with the exception of cells that span columns.

7.2 Japanese/Chinese pronunciation annotations of words/phrases (clarified)

This is the other (completely different) use for PTX. It has nothing to do with tables.

- **CSI 1\** Beginning of principal text to be phonetically annotated. The principal text should be a "word" or short phrase. This must not contain any PTX control sequences.
- **CSI 3\** or **CSI 4\** End of principal text and beginning of Japanese (3) or Chinese (4) phonetic annotation text (Ruby text). The phonetic annotation text is usually Hiragana or Katakana text for Japanese, Pinyin (i.e. Latin) or Bopomofo text for Chinese. There can be only one such phonetic annotation per principal text. The principal and supplementary texts must *not* contain any PTX control sequences. **CSI 3\** does not allow for parenthesis fallback, but **CSI 4\** does (compare **<rp>** in HTML). All SGR styling is reset to that at the start of the Ruby.
- **CSI 5\** End of Chinese/Japanese phonetic annotation. The syntax for this use of PTX is **CSI 1\***principal text***CSI** (**3** or **4**)**\phonetic annotationCSI 5\**. This corresponds in principle to the HTML **<rb>***principal text***</rb><rt>***phonetic annotation***</rt>**. An implementation may allow **CSI 0\** to terminate also Ruby annotations. All SGR styling is reset to that at the start of the Ruby.

This corresponds to the Ruby markup in HTML (https://www.w3.org/International/articles/ruby/markup.en) for producing so-called Ruby text, i.e. phonetic annotations written above (horizontal lines) or to the right (vertical lines) of the principal text in Japanese or Chinese (though for Chinese, ECMA-48 suggests the alternative of using a parenthetical inline annotation). For Ruby layout details, see text on HTMLs Ruby markup. There should be no bidi or RTL in Ruby notation.

Since Ruby is (in principle) targeted at Japanese and Chinese, bidi is not expected to even be enabled. But in case it is: The Ruby component as a whole must be atomic (i.e., handled as a single, bidi neutral, character) for bidi processing. And the character progression direction is inherited to each part, in isolation. Further, bidi, if enabled, applies to each part of the Ruby notation, in isolation. The parentheses fallback is laid out in the character progression direction.

8 STYLE BRACKETING – SBR – Pushing and popping SGR, HTSA and palette states (new)

In contrast to, e.g., HTML, there is no general nesting, or bracketing, structure in ECMA-48. But PTX, PARALLEL TEXTS, does have a structure and nesting ability (**CSI 1\**.....**CSI 0\**; row start…row end). So does SDS/SRS (START DIRECTED STRING (**CSI 1]**.....**CSI 0]** and **CSI 2]**.....**CSI 0]**; cmp. LRO…PDF and RLO…PDF), START REVERSED STRING (**CSI 1[**.....**CSI 0[**; cmp. LRO/RLO…PDF, LRO or RLO is selected to be opposite the direction at the point of **CSI 1[**). Math expression also needs to have a nesting structure.

But XTerm has a nonstandard extension for pushing and popping SGR state. That extension uses private-use “final bytes” (final characters), namely “**{**” and “**}**”. If one wants to standardise this functionality, non-private-use final characters need to be used (with intermediary character(s)). Here we suggest the control sequences:

- **CSI** *n*!**[** push a copy of all currently set rendition attributes and more: SGR, HTSA, colour palette plus language, charset (can conceivably change for terminal emulators), as well as font palette. Mark the stack entry with the “GUID”, or non-GUID, *n* (decimal digits only).
- **CSI** *n*!**]** pop the rendition attributes stack to and including the topmost level marked with the GUID *n*, using the last popped (GUID *n*) stacked attributes as the currently set ones (both SGR, HTSA colour palette, language, charset, and font palette). **No-op** w.r.t. those settings if *n* is not encountered the stack. For output to a terminal emulator, the GUID should be hard to “guess” if not known. For use inside a document, one may use a very simple non-GUID, like “1” or the current save stack depth.
  All PTX, math expressions, and bidi control nesting are closed, even if the GUID is not encountered in the stack.
  Note that **ESC c** (RESET TO INITIAL STATE, for terminal emulators only) resets even more, if implemented.

9 UI TEXT MARKING – UTM (new, not to be stored in docs)

Some applications mark, for instance, search matches, or substrings selected for edit operations (the latter usually only one substring at the time) by using a differently coloured background (exactly which colour varies by application, or even preference setting). Some applications also mark suspected spell errors by a special underline (wavy red seems popular), or suspected style issues (of various kinds) by another coloured underline. Such markings are not part of the styling of the text, but are part of the user interface. An application which does these markings (only) on the "display side" need not invoke any control sequences (or HTML/CSS, or other markup) to display these; and indeed should use different colour layers for those than for the regular background/underline. But for an application where a "remote side" (using ECMA-48 control sequences) decides what to mark that way, it is possible to sometimes repurpose the backgrounds and underlines available via SGR. For such applications (like for applications intended to be displaying the text on a terminal) an ability to "access" the colour planes normally used for those UI markings would be better.

So here we propose UI marking control sequences, which are separate from SGR. UI markings should not be stored in documents. These control sequences are intended for applications where the "remote" side "orders" temporary UI markings, such as "selected"/"matching", spelling or style "remark", substring "now being processed", and similar cases. As opposed to SGR background or underline which cannot have multiple underlines or multiple backgrounds (a limitation compared to some other markup systems), a substring can have multiple UI backgrounds (e.g. both "matching" and "selected"), and multiple UI underlines (e.g. both spell error and style warning), and they do not override the backgrounds or underlines that are part of the document that is displayed (though they may interfere visually). When bidi is used, one span may be split into several separately displayed sections (just as for SGR styles).

- **CSI 0!m** <u>End all UI text marks</u>. Stop all UI text markings.
- **CSI 1:***n***!m** <u>Start UI background colour mark to the colour of index *n* in the palette</u>. *n* is here used both as an index to the palette (same palette as used for SGR) and as an identifier for which UI background is referenced. This way multiple markings (of different palette indices) can overlap, though there is no nesting. Colour plane 2.5. Terminated by **CSI 1:***n***!m** with same value for *n* (restarting with the same colour) or **CSI 2:***n***!m** with same value for *n* or by **CSI 0!m**.
- **CSI 2:**n**!m** <u>End UI background colour mark that started by using index *n* in the palette</u>. Other UI background colour marks are not affected.
- **CSI 3:***n*[:*v*]**!m** <u>Start UI underline having the colour of index *n* in the palette</u>. *n* here is used both as an index to the palette and as an identifier for which UI underline is referenced. This way multiple markings can overlap (e.g. both spell check marking and style check marking; of different palette indices), though there is no nesting. Same variants as **CSI 4m**, but these lines are a bit lower under the characters (close to the descenders). Default is double wavy underline. Multiple UI underlines on the same substring may stack. If the line progression direction is left to right, the "underline" is to the right of the text. If the line progression direction is right to left, the "underline" is to the left of the text. Colour planes 6.5. Terminated by **CSI 3:***n*:[:*v*]**!m** with same value for *n* (possibly changing type of underline, but not colour) or **CSI 4:***n***!m** with same value for *n*, and by **CSI 0!m**.
- **CSI 4:***n***!m** <u>End UI underline that started by using by index *n* in the palette</u>. Other UI underlines are not affected.

10 Converting Teletext styling to ECMA-48 styling

Teletext is still in common use around the world, especially for optional subtitling, though the use of Teletext for news pages has declined or even been abandoned. Teletext allows for certain style settings, mostly to do with colour. There are also control bits in the Teletext protocol for handling bold, italic and underline. The colour codes also select which set of characters to use (either alphanumeric or “mosaic” characters, the latter are Teletext specific symbols used to build up larger (and crude…) graphics). Note that the coding for “alphanumeric” characters have multiple variants for G0/G2, various Latin subsets, Greek, Cyrillic, Arabic and Hebrew. These are selected by control bits in the Teletext protocol, in addition **ESC** can be used to switch between a primary and a secondary G0+G2 set. Further, the “mosaic” characters (G1) have two variants, a “contiguous” form (default, and selected by **0x19**) and a “separated” form (selected by **0x1A**). From a modern encoding point of view the latter are separate characters (not yet in Unicode). Teletext also allows for dynamically defined (“bitmapped”) fonts of unspecified charset.

The table is a rough sketch only. Teletext overrides (italics, bold, underline, more colours, prop. font, G3 chars.) are not covered in the sketch mapping here, though the functionality is covered. Character sets are specified in control bits in the Teletext protocol, that is not covered here.

Teletext	**ECMA-48** (as extended here)
0x00 Alpha black (and sets “alpha mode”, G0) (default)	**SP***?* **CSI 30m**
0x01 Alpha red (and sets “alpha mode”, G0)	**SP***?* **CSI 91m**
0x02 Alpha green (and sets “alpha mode”, G0)	**SP***?* **CSI 92m**
0x03 Alpha yellow (and sets “alpha mode”, G0)	**SP***?* **CSI 93m**
0x04 Alpha blue (and sets “alpha mode”, G0)	**SP***?* **CSI 94m**
0x05 Alpha magenta (and sets “alpha mode”, G0)	**SP***?* **CSI 95m**
0x06 Alpha cyan (and sets “alpha mode”, G0)	**SP***?* **CSI 96m**
0x07 Alpha white (and sets “alpha mode”, G0)	**SP***?* **CSI 37m**
0x08 Flash	**SP***?* **CSI 5m** or **SP***?* **CSI 6m**
0x09 Steady (default)	**CSI 25m SP***?*
0x0A End box (default)	**SP***?* **CSI 112:0m** (allow 100% fading)
0x0B Start box (news flashes; also used for subtitles)	**SP***?* **CSI 112:1m** (block fading)
0x0C Normal size (default)	**SP***?* **CSI 73:1:1m**

0x0D Double height (note: background & foreground overshadows text on the next line)	**SP***?* **CSI 73:2:1m** (overshadowing is implementation defined)
0x0E Double width	**SP***?* **CSI 73:1:2m**
0x0F Double size (note: extends down one "row" (overshadows text on the next line))	**SP***?* **CSI 73:2:2m** (might not overshadow text on the next line)
0x10 Mosaics black (and sets "mosaics mode", G1)	**SP***?* **CSI 30m**
0x11 Mosaics red (and sets "mosaics mode", G1)	**SP***?* **CSI 91m**
0x12 Mosaics green (and sets "mosaics mode", G1)	**SP***?* **CSI 92m**
0x13 Mosaics yellow (and sets "mosaics mode", G1)	**SP***?* **CSI 93m**
0x14 Mosaics blue (and sets "mosaics mode", G1)	**SP***?* **CSI 94m**
0x15 Mosaics magenta (and sets "mosaics mode", G1)	**SP***?* **CSI 95m**
0x16 Mosaics cyan (and sets "mosaics mode", G1)	**SP***?* **CSI 96m**
0x17 Mosaics white (and sets "mosaics mode", G1)	**SP***?* **CSI 37m**
0x18 Conceal (ignored on the currently displayed page when user presses a 'reveal' button)	**CSI 38:8m SP***?* (*not* **CSI 8m**)
0x19 ("mosaics mode") Contiguous mosaic graphics (default); note: does *not* unset underlined in "alpha mode"	(converter change) **SP***?*
0x1A ("mosaics mode") Separated mosaic graphics; note: does *not* set underlined in "alpha mode"	(converter change) **SP***?*
0x1B Escape (toggles between two G0 code pages)	**SP***?* (converter change)
0x1C Black background (default)	**CSI 40m SP***?*
0x1D New background (foreground to background)	**CSI 48:6m SP***?*
0x1E Hold mosaics (details of hold/release mosaics is beyond the scope of this paper)	(**SP***?* **:=** *<mos. char>*; **SP***?*
0x1F Release mosaics (details of hold/release mosaics is beyond the scope of this paper)	**SP***?*; **SP***?* **:= SP**
line break (implicit; Teletext has numbered "rows") (style settings are reset at the beginning of each row)	*line break* (CRLF/CR/LF), **CSI 0m**; **SP***?* **:= SP**

11 Converting ISCII styling to ECMA-48 styling

ISCII formally allows for 31 style settings (many of them outline/shadow) by combining several ATR (attribute; the code for ATR in ISCII is **0xEF**) codes (similar to that ECMA-48 stylings can be combined; but not quite same, vide: ATR HLT ATR BLD does not set independent style attributes, but specifies extra-bold, **CSI 1:3m**). Here we give some of the ISCII styles, and their corresponding ECMA-48 styling codes.

The table is a rough sketch only. We will not give the combinations here, just the basic ones.

ISCII (all these ATR toggle)	**ECMA-48** (as extended here)
ATR BLD (= ATR **0**) (bold)	**CSI 1m** or **CSI 1:3m**, **CSI 22m**
ATR ITA (= ATR **1**) (italic)	**CSI 3m**, **CSI 23m**
ATR UL (= ATR **2**) (underline)	**CSI 4m**, **CSI 24m**
ATR EXP (= ATR **3**) (expand)	**CSI 73:1:2m**, **CSI 73:1:1m**
ATR HLT (= ATR **4**) (highlight/bold)	**CSI 1m** or **CSI 1:3m**, **CSI 22m**

ATR OTL (= ATR **5**) (outline)	**CSI 80:6:5;38:6m** (setting the all-around 0,05 em shadow colour to be the current text colour, and then setting the text colour to be the same as the (opaque) background colour), **CSI 38:7;89m** (if not combined with SHD)
ATR SHD (= ATR **6**) (shadow)	**CSI 81:45:6:10;80:6:5;38:6m** (extra shadow to the 'south-east'), **CSI 38:7;89m** (if not combined with OTL)
ATR TOP (= ATR **7**) (top half of double height glyphs)	**CSI 73:2:1;113:1m**, **CSI 73:1:1;113:0m**
ATR LOW (= ATR **8**) (low half of double height glyphs)	**CSI 73:2:1;113:2m**, **CSI 73:1:1;113:0m**
ATR DBL (= ATR **9**) (double size)	**CSI 73:2:2m** (combined with ATR TOP and ATR LOW one can get **CSI 73:4:2m** and combined with ATR EXP also **CSI 73:4:4m**) or maybe **CSI 72:0:2m** (not clear from ISCII), **CSI 73:1:1m** or maybe **CSI 72:0:0?5m**
line break char (in ISCII all style settings are auto-reset at the beginning of each line)	*line break char*, **CSI 0m**

12 Cut and paste

Modern systems offer a "cut and paste" functionality, enabling to save a "clip" in a "clipboard" and then insert that "clip" somewhere else, often via another application. The data saved in the clipboard is in a system dependent format, or even multiple formats. But it usually allows for saving text styling attributes when saving a text clip.

However, it is (and will probably continue to be) unlikely that the styling is saved by using ECMA-48 control sequences in the "clipboard" (though the ECMA-48 control sequences can in principle be stored in the "plain text" version of the clip; recipient programs might not be able to interpret ECMA-48 control sequences). If one does not want to lose the styling attributes (when cutting-and-pasting from an application that does interpret ECMA-48 styling attributes to one that does not, but supports other styling methods), then conversion(s) are needed at some point(s), so that what is saved in the clipboard closely follows the style of the original, to the extent reasonably possible. In addition, saving a clip as "plain text" may or may not keep ECMA-48 style control sequences, whether interpreted or not. That depends on the application and other circumstances.

The clipboard, as mentioned, as well as the receiving application need not support the ECMA-48 way of specifying styling, but one can still get a good, or in ideal cases even perfect, styling copied via the clipboard. E.g., HTML with inline CSS, a possible clipboard format, can represent nearly all the styles mentioned above (some styles, like "margin lines", use of "colour palette", excepted). Further, that format might not be able to fully handle all the line break characters listed above, nor HTJ nor tab stop settings, though LF and functionally equivalent characters can be represented by "<p>" or "</p><p>" (unless within a "<pre>") and VT and functionally equivalent characters by "<br/>" in HTML. In any case one needs to pick up formatting in effect from before the beginning of the text to be copied, and reset at the end of the text copied.

When pasting to a terminal emulator, only proper plain text (no ECMA-48 control sequences, nor any other ESC sequences) is often preferable (removing them from “plain text” if need be). An exception could be an ECMA-48 enabled terminal window text editor, but then the pasting needs to convert the styles stored in the clipboard to ECMA-48 control sequences.

When copying from a terminal emulator to an external text editor, it may be a good idea to convert LF to VT (in addition to keeping styling). That way auto-numbered lists (common in rich text editors) will not get messed up if pasting into a list item from a terminal emulator.

13 Control/escape/sci sequences and combining characters

A C0/C1/Cf control character, escape sequence or a control sequence must not occur just before a combining character. Since the final character in an escape sequence or a control sequence, as well as an SCI sequence, is a printable (ASCII) character, often a letter, applying normalisation to NFC may combine that final character and the combining character to a precomposed character, thus “ruining” the escape/control/sci sequence, and misapply the combining character to the wrong base character.

14 Security aspects

Filtering out ECMA-48 control sequences or escape sequences at the “wrong point” of the process, may expose security sensitive data (such as blocked commands) that may have been “hidden” by ECMA-48 control/escape sequences earlier on. These issues have been ever-present in ECMA-48.

The same warning is valid for any other substring(s) removal or substitution, such as replacing line breaks with spaces or spaces with line breaks, removing HTML tags or converting HTML character references, removing uninterpreted control characters, or removing/replacing portions that have encoding errors; all of these are common automatic edits.

SCI controls have an inherent length limit, exactly one character after the SCI is part of the SCI sequence. For CSI control sequences, it is reasonable to limit the length to, say, 80 characters. (We do not give a firm limit here.)

For APC/DCS/OSC/PM/SOS control sequences/strings, a limit of 100 characters is reasonable, except that for say graphic drawing commands (like HPGL or SVG, which is a possible private-use use) one could allow larger content, as long as there are no syntax errors and there is some reasonably visible output being “plotted” (which can be hard to determine…). Even if the data is seen as text that is not a script or commands, the substrings between occurrences of APC/DCS/OSC/PM/SOS and the respective occurrences of ST may contain scripts or commands, some of which may have implementation dependent security issues.

15 Conclusion

Whether to regard ECMA-48 text styling as a higher-level protocol (for text styling) or not is mostly a matter of taste. Technically it is a (text styling) protocol at the text level. The current principal application, in terminal emulators, relies on it being text level, rather than higher level.

ECMA-48 may seem a bit old-fashioned, since the first edition was issued in September 1976. It has seen little use outside of terminals and terminal emulators, where it is still “alive and well”; though there were some efforts to incorporate parts of it in a document format standard. Much of ECMA-48 is concerned with “terminal only” functionality (some technically possible to apply to modern text editors, also those that do not run “under” a terminal emulator, like cursor movement control sequences, but modern editors use other mechanisms). Despite this, the styling mechanisms can still be used in a modern context, also when storing (styled) text in files. Especially if one modernises the capabilities of ECMA-48 styling, which we are proposing in this paper. Those modernisations can be applied both to terminal emulators as well as text files storing text in a “plain text formatting” format. While one could invent from scratch a different “plain text formatting” format, there seems little need for that considering that we already have ECMA-48. For backward compatibility reasons, terminal emulators will need to stay with ECMA-48. The additions proposed in this paper are in “the spirit of” ECMA-48, and much can apply to terminal emulators as well. With the modernisations, ECMA-48 styling is viable as a simpler yet powerful text formatting protocol for text files, as well as terminal emulators.

In addition to extending ECMA-48 styling in order to modernise it (in a compatible way), we have also added some functionality to support styling that was present in ISCII, and support for styling that is used in current TeleText use around the world (mostly kept for user optional subtitling, though sometimes still in use for news and other pages). With these extensions, ISCII text that use ISCII formatting can be converted to Unicode while preserving the formatting via these ECMA-48 extensions. Likewise, Teletext texts can be converted to Unicode preserving the formatting via these ECMA-48 extensions, assuming that Teletext “graphic” characters (such as SEPARATED BLOCK SEXTANTs) will be defined also in Unicode. For Teletext, one also need to interpret the Teletext protocol itself for bold/italics/underline, line breaks, page/subpage numbers, as well as charset information.

16 REFERENCES

[CSS]	Cascading Style Sheets – World Wide Web Consortium (W3C), https://www.w3.org/Style/CSS.
[ECMA-48]	*Control Functions for Coded Character Sets*, 5th ed., June 1991, https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-048.pdf. (Earlier editions as well.)
[HPGL]	*Hewlett-Packard Graphics Language/2*, https://www.hpmuseum.net/document.php?catfile=213, 1996.
[HTML]	W3C HTML – World Wide Web Consortium (W3C), https://www.w3.org/html.
[ISCII]	Indian Standard – *Indian script code for information interchange* – ISCII, 1991. http://varamozhi.sourceforge.net/iscii91.pdf, https://ia800908.us.archive.org/19/items/gov.in.is.13194.1991/is.13194.1991.pdf.
[ISO/IEC 6429:1992]	Information technology – *Control functions for coded character sets*, 3rd ed. Same as [ECMA-48] 5th ed.
[ISO/IEC 8613-6:1994]	Information technology – *Open Document Architecture (ODA) and Interchange Format: Character content architectures*. Same as [ITU-REC-T.416].
[ISO/IEC 10646]	Information technology – *Universal Coded Character Set* (UCS), https://standards.iso.org/ittf/PubliclyAvailableStandards/c069119_ISO_IEC_10646_2017.zip. Same coded characters as in [Unicode].
[ITU-REC-T.416]	Information technology – *Open Document Architecture (ODA) and interchange format: Character content architectures*, 1993. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.416-199303-I!!PDF-E&type=items.
[Markdown]	Markdown, John Gruber, described by Matt Cone in https://www.markdownguide.org/.
[Math controls]	Kent Karlsson. *A true plain text format for math expressions (and its XML compatible equivalent format)*. 2022.
[RTF]	Word 2003: *Rich Text Format (RTF) Specification*, version 1.8, Microsoft, http://www.microsoft.com/downloads/details.aspx?familyid=AC57DE32-17F0-4B46-9E4E-467EF9BC5540&displaylang=en, 2004.
[SVG]	Scalable Vector Graphics – World Wide Web Consortium (W3C), https://www.w3.org/Graphics/SVG.
[Teletext]	*Enhanced Teletext specification*, ETSI EN 300 706 V1.2.1, https://www.etsi.org/deliver/etsi_en/300700_300799/300706/01.02.01_60/en_300706v010201p.pdf, 2003.
[Teletext in DVB]	Digital Video Broadcasting (DVB); *Specification for conveying ITU-R System B Teletext in DVB bitstreams*, ETSI EN 300 472 V1.4.1, https://www.etsi.org/deliver/etsi_en/300400_300499/300472/01.04.01_60/en_300472v010401p.pdf, 2017.
[TeX]	Knuth, Donald Ervin, *The TeXbook*, Computers and Typesetting, Addison-Wesley, ISBN 0-201-13448-9, 1984. (Plus later developments.)
[Unicode]	*The Unicode Standard*, http://www.unicode.org/versions/latest/, web site https://home.unicode.org/, http://unicode.org/main.html, …, 2017, …
[Wikitext]	Mediawiki syntax, markdown as used in Wikipedia, https://en.wikipedia.org/wiki/Help:Wikitext.
[XTerm control seq.]	*XTerm Control Sequences*, https://invisible-island.net/xterm/ctlseqs/ctlseqs.html, 2019.

Appendix – control sequences summary

SGR

CSI [**0**]**m** Reset rendition. Handy for things like beginning of prompts, where the current state is unknown.
CSI 1[**:***v*]**m** Bold font variant.
CSI 2[**:***v*]**m** Lean font variant.
CSI 3[**:***v*]**m** Italicized or oblique.
CSI 4[**:***v*]**m** Singly underlined.
CSI 5[**:***v*[**:***w*]]**m** Slowly blinking.
CSI 6[**:***v*[**:***w*]]**m** Rapidly blinking.
CSI 7[**:***f*]**m** Negative image.
CSI 8[**:***v*]**m** Concealed/censored characters.
CSI 9[**:***v*]**m** Crossed out (strike-through) characters.
CSI 10m, ..., **CSI 19m** Palette of font sets (… —lean—normal—bold--extra bold— …; italic/upright; “optical” adaptations for compressed/expanded and different sizes, or a multiple-masters font). (Each should have information if it is calligraphic or not. If not calligraphic, there should be information whether they are sans-serif, serif, or fixed-width.)
CSI 20m Change to a predetermined calligraphic font set.
CSI 21[**:***v*]**m** Doubly underlined.
CSI 22[**:***v*]**m** Normal weight (neither bold nor lean) (default).
CSI 23[**:***v*]**m** Roman/upright, i.e. not italicized/oblique (default).
CSI 24[**:***v*]**m** Not underlined (default).
CSI 25[**:***v*[**:***w*]]**m** Steady (not blinking) (default).
CSI 26[**:***v*]**m** Change to a predetermined serif or non-serif proportional (but non-calligraphic) font set.
CSI 27[**:***f*]**m** Positive image (default).
CSI 28[**:***v*]**m** Cancel “concealed/censored characters” effect (default).
CSI 29[**:***v*]**m** Not crossed out (default).

Foreground colour shorts (default transparency is 0, i.e. fully opaque):
CSI 30[**:***t*]**m** is short for **CSI 38:2::0:0:0:*t*m**, Pure black foreground.
CSI 31[**:***t*]**m** is short for **CSI 38:2::205:0:0:*t*m**, Dull red foreground.
CSI 32[**:***t*]**m** is short for **CSI 38:2::0:205:0:*t*m**, Dull green foreground.
CSI 33[**:***t*]**m** is short for **CSI 38:2::255:215:0:*t*m**, Dull yellow foreground.
CSI 34[**:***t*]**m** is short for **CSI 38:2::0:0:205:*t*m**, Dull blue foreground.
CSI 35[**:***t*]**m** is short for **CSI 38:2::205:0:205:*t*m**, Dull magenta foreground.
CSI 36[**:***t*]**m** is short for **CSI 38:2::0:205:205:*t*m**, Dull cyan foreground.
CSI 37[**:***t*]**m** is short for **CSI 38:2::255:255:255:*t*m**, Pure white foreground.
CSI 38[**:0**]**m** Reset text foreground colour to default.
CSI 38:1m Fully transparent foreground. Short for **CSI 38:2::0:0:0:255?75m**.
CSI 38:2:[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Foreground colour as RGB(T).
CSI 38:3:[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Foreground colour as CMY(T).
CSI 38:4:[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Foreground colour as CMYK.
However, if *i* is negative (including negative zero, **=0**), there is no change in foreground colour, but there is an assignment to the colour palette (unless the position is write protected).
CSI 38:5:*p*[**:***t*]**m** Foreground colour from colour palette.
CSI 38:6m Copy current background colour to foreground colour. Not recommended.
CSI 38:7m Copy current all-around shadow colour to foreground colour. Not recommended.
CSI 38:8m Optionally transparent foreground. (For Teletext.) Deprecated.
CSI 38:9:*b***:***p1***:**[*t1*]**:***p2*[**:***t2*]**m** Gradient glyph colouring.
CSI 39m Reset to default text foreground colour.

Background (or highlight) colour shorts (default transparency is implementation defined, and may be settable in preferences):

CSI 40[**:***t*]**m** is short for **CSI 48:2::0:0:0:***t***m**, Pure black background.
CSI 41[**:***t*]**m** is short for **CSI 48:2::205:0:0:***t***m**, Dull red background.
CSI 42[**:***t*]**m** is short for **CSI 48:2::0:205:0:***t***m**, Dull green background.
CSI 43[**:***t*]**m** is short for **CSI 48:2::255:215:0:***t***m**, Dull yellow background.
CSI 44[**:***t*]**m** is short for **CSI 48:2::0:0:205:***t***m**, Dull blue background.
CSI 45[**:***t*]**m** is short for **CSI 48:2::205:0:205:***t***m**, Dull magenta background.
CSI 46[**:***t*]**m** is short for **CSI 48:2::0:205:205:***t***m**, Dull cyan background.
CSI 47[**:***t*]**m** is short for **CSI 48:2::255:255:255:***t***m**, Pure white background.
CSI 48[**:0**]**m** Reset background colour to default text background/highlight colour.
CSI 48:1m Fully transparent text background/highlight.
CSI 48:2:[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Text background colour as RGB(T).
CSI 48:3:[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Text background colour as CMY(T).
CSI 48:4:[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Text background colour as CMYK.
However, if *i* is negative (including negative zero, **=0**), there is no change in background/highlight colour, but there is an assignment to the colour palette (unless the position is write protected).
CSI 48:5:*p*[**:***t*]**m** Text background/highlight colour from colour palette.
CSI 48:6m Copy current foreground colour to background colour. Not recommended.
CSI 49m Reset to default text background/highlight colour (may be settable in preferences).

CSI 50m Change to a predetermined "fixed" width font set. (Three widths, width 0 en for non-spacing characters, width 1 en for "narrow" characters, 2 en (i.e., 1 em) for "wide" characters.)

CSI 51[**:***v*[**:***r*]]**m** Framed string.
CSI 52[**:***v*[**:***r*]]**m** Encircled string.
CSI 53[**:***v*]**m** Overlined.
CSI 54[**:***v*[**:***r*]]**m** Not framed, not encircled.
CSI 55[**:***v*]**m** Not overlined.
CSI 56:=1m Lowered to first subscript level and slightly smaller.
CSI 56[**:0**]**m** Not raised/lowered and back to the set size (default.)
CSI 56:1[**:***v*]**m** Raised to first superscript level and slightly smaller.
CSI 57[**:0**]**m** Reset to default w.r.t. ligatures and marked zero.
CSI 57:1[**:***v*]**m** Allow modern ligatures (may be default for text display/editing).
CSI 57:2[**:***v*]**m** No ligatures but mark zero (may be default for terminal emulators).

CSI 58[**:0**]**m** Reset text emphasis mark colour to follow foreground colour. Default.
CSI 58:1m Fully transparent text emphasis. (Not recommended.)
CSI 58:2:[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]m Text emphasis colour as RGB(T).
CSI 58:3:[*i*]**:***c***:***m***:**y[**:**t[**:***a***:0**]]m Text emphasis colour as CMY(T).
CSI 58:4:[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]m Text emphasis colour as CMYK.
However, if *i* is negative (including negative zero, **=0**), there is no change in emphasis (like underline) colour, but there is an assignment to the colour palette (unless the position is write protected).
CSI 58:5:*p*[**:***t*]**m** Text emphasis colour from colour palette.
CSI 59m Reset emphasis colour to follow text foreground colour.

CSI 60[**:***v*]**m** CJK underline (on the right side if vertical character progression direction).
CSI 61[**:***v*]**m** CJK double underline (on the right side if vertical character progression direction).
CSI 62[**:***v*]**m** CJK overline (on the left side if vertical character progression direction).
CSI 63[**:***v*]**m** CJK double overline (on the left side if vertical character progression direction).
CSI 64[**:***v*]**m** CJK stress marks (dot placed under/over (right/left side if vertical writing)).
CSI 65m Cancel the effect of the renditions established by parameter values 60 to 64.

CSI 66[**:***v*]**m** Display uppercase letters as small caps (near x-height capitals).
CSI 67[**:***v*]**m** Display letters as normal (default).

CSI 68[**:0**]**m** Reset text crossed-out/strike-through and string framing colour to follow foreground colour.
CSI 68:1m Fully transparent overstrike and framing lines. (Not recommended.)
CSI 68:2:[*i*]**:***r***:***g***:***b*[**:***t*[**:***a***:0**]]**m** Text overstrike and string framing colour as RGB(T).
CSI 68:3:[*i*]**:***c***:***m***:***y*[**:***t*[**:***a***:0**]]**m** Text overstrike and string framing colour as CMY(T).
CSI 68:4:[*i*]**:***c***:***m***:***y***:***k*[**:***a***:0**]**m** Text overstrike and string framing colour as CMYK.
However, if *i* is negative (including negative zero, **=0**), there is no change in overstrike/framing colour, but there is an assignment to the colour palette (unless the position is write protected).
CSI 68:5:*p*[**:***t*]**m** Text overstrike and string framing colour from colour palette.

CSI 69m Lowercase digits, if available in the font.
CSI 70m Proportional width uppercase digits, if available in the font.
CSI 71m Fixed width uppercase digits (default).

CSI 72:*u***:***s***m** Font size.
CSI 73:*a***:***b***m** Font magnification.
CSI 74:*u***:***s***m** Line spacing.
CSI 75:*u***:***v*[**:***s*]**m** First line BOL (beginning of line) indent.
CSI 76:*u***:***v*[**:***s*]**m** Non-first line BOL line indent (after auto-line-break).
CSI 77:*u***:***v*[**:***j*]**m** EOL (end of line) line indent. Also sets paragraph justification.
CSI 78[**:**[*v*][**:***p*]]**m** Line in the margin before the beginning-of-line position.
CSI 79[**:**[*v*][**:***p*]]**m** Line in the margin after end-of-line position.

CSI 80[**:0**]**m** Cancel all-around shadow, directional shadow and its "counter-shadow" (if any).
CSI 80:2:*d***:***r***:***g***:***b*[**:***t*[**:***f*]]**m** All-around shadow colour as RGB(T).
CSI 80:3:*d***:***c***:***m***:***y*[**:***t*[**:***f*]]**m** All-around shadow colour as CMY(T).
CSI 80:4:*d***:***c***:***m***:***y***:***k*[**:***f*]**m** All-around shadow colour as CMYK. Fully opaque.
CSI 80:5:*d***:***p*[**:**[*t*][**:***f*]]**m** All-around shadow colour from palette. *p* is palette index.
CSI 80:6:*d*[**:***f*]**m** Copy current foreground colour to all-around shadow colour. Not recommended.
CSI 80:9:*d***:***b***:***p1***:**[*t1*]**:***p2*[**:**[*t2*][**:***f*]]**m** Gradient all-around glyph shadow colouring.
CSI 81[**:0**]**m** Cancel directional shadow and its "counter-shadow" (if any).
CSI 81:*e***:2:***d***:***r***:***g***:***b*[**:***t*[**:***f*]]**m** Directional shadow colour as RGB(T).
CSI 81:*e***:3:***d***:***c***:***m***:***y*[**:***t*[**:***f*]]**m** Directional shadow colour as CMY(T).
CSI 81:*e***:4:***d***:***c***:***m***:***y***:***k*[**:***f*]**m** Directional shadow colour as CMYK.
CSI 81:*e***:5:***d***:***p*[**:***f*]**m** Directional shadow colour from palette.
CSI 81:*e***:6:***d*[**:***f*]**m** Copy current foreground colour to directed shadow colour. Not recommended.
CSI 81:*e***:9:***d***:***b***:***p1***:**[*t1*]**:***p2*[**:**[*t2*][**:***f*]]**m** Gradient directional glyph shadow colouring.
CSI 82:…**m** "Counter-shadow" to the current directional shadow.
CSI 83m, …, **CSI 89m** Reserved (possibly for short forms for "popular" shadow combinations or maybe semi-counter-shadows).

Foreground colour shorts (default transparency is 0, i.e. fully opaque):
CSI 90[**:***t*]**m** is short for **CSI 38:2::105:105:105:***t***m**, dark grey foreground.
CSI 91[**:***t*]**m** is short for **CSI 38:2::255:0:0:***t***m**, clear red foreground.
CSI 92[**:***t*]**m** is short for **CSI 38:2::0:255:0:***t***m**, clear green foreground.
CSI 93[**:***t*]**m** is short for **CSI 38:2::255:255:0:***t***m**, clear yellow foreground.
CSI 94[:*t*]**m** is short for **CSI 38:2::0:0:255:***t***m**, clear blue foreground.
CSI 95[**:***t*]**m** is short for **CSI 38:2::255:0:255:***t***m**, clear magenta foreground.
CSI 96[**:***t*]**m** is short for **CSI 38:2::0:255:255:***t***m**, clear cyan foreground.
CSI 97[**:***t*]**m** is short for **CSI 38:2::220:220:220:***t***m**, light grey foreground.
CSI 98[**:***t*]**m** is short for **CSI 38:2::169:169:169:***t***m**, medium grey foreground.
CSI 99[**:***t*]**m** is short for **CSI 38:2::255:140:0:***t***m**, orange foreground.

Background (or highlight) colour shorts (default transparency is implementation defined):
CSI 100[**:***t*]**m** is short for **CSI 48:2::105:105:105:***t***m**, dark grey background.
CSI 101[**:***t*]**m** is short for **CSI 48:2::255:0:0:***t***m**, clear red background.
CSI 102[**:***t*]**m** is short for **CSI 48:2::0:255:0:***t***m**, clear green background.
CSI 103[**:***t*]**m** is short for **CSI 48:2::255:255:0:***t***m**, clear yellow background.
CSI 104[**:***t*]**m** is short for **CSI 48:2::0:0:255:***t***m**, clear blue background.
CSI 105[**:***t*]**m** is short for **CSI 48:2::255:0:255:***t***m**, clear magenta background.
CSI 106[**:***t*]**m** is short for **CSI 48:2::0:255:255:***t***m**, clear cyan background.
CSI 107[**:***t*]**m** is short for **CSI 48:2::220:220:220:***t***m**, light grey background.
CSI 108[**:***t*]**m** is short for **CSI 48:2::169:169:169:***t***m**, medium grey background.
CSI 109[**:***t*]**m** is short for **CSI 48:2::255:140:0:***t***m**, orange background.

CSI 110[**:***z*]**m** Advancement modification.
CSI 111[**:***x*]**m** Space advancement modification.

CSI 112:*f***m** Fade control factor.
CSI 113:*v***m** Show horizontally halved glyphs. (Only for conversion from ISCII, otherwise deprecated.)

CSI 114[**:**[*v*][**:**[*p1*][**:**[*b*][**:**[*p2*][**:***c*]]]]]**m** Start frame of paragraphs.
CSI 115m End frame of paragraphs.

HTSA

CSI *stoplist***!N** Set tab stops according to the *stoplist*. **CSI !N** clears all HTSA set tab stops.

SPD

CSI 0!V Character progression left-to-right, no bidi processing. Common default; also used for *visual order*.
CSI 1!V CJK/Hangul vertical, line progression right to left, no bidi processing.
CSI 2!V Mongolian vertical, line progression left to right, no bidi processing.
CSI 3!V Character progression right-to-left, with Unicode bidi processing; actually sets paragraph direction.
CSI 4!V, …, **CSI 7!V** Reserved.
CSI 8!V Character progression left-to-right, with Unicode bidi processing; actually sets paragraph direction.
CSI 10!V Boustrophedon. No bidi processing, but right-to-left lines get mirrored glyphs.
CSI 11!V Reserved.
CSI 99!V Reset to default directions, per preference setting.

UI text emphasis marking

For temporary text styling for UI reasons, like marking text selection, spell errors, pattern matches, etc. for terminal emulator-oriented programs (other programs are likely to use other mechanisms for UI text marking). These control sequences should not occur in "**.txtf**" text documents.

CSI 0!m End all UI text marks. Stop all UI text markings. Handy for things like beginning of prompts, where the current state of UI text marking is unknown.
CSI 1:*n***!m** Start UI background colour mark to the colour of index *n* in the palette.
CSI 2:*n***!m** End UI background colour mark that started by using index *n* in the palette.
CSI 3:*n*[**:***v*]**!m** Start UI underline, variant *v*, having the colour of index *n* in the palette.
CSI 4:*n***!m** End UI underline that started by using by index *n* in the palette.

In addition, UI text foreground colour change and UI bold and oblique may be considered.

PTX as table row

CSI 1[**:***u***:***s*]\ Beginning of first text of the list of parallel texts. Beginning of table row, or beginning of base text for Ruby annotation.
CSI 2[**:***n*[**:***v*]]\ End of previous text (cell) and beginning of next text (cell) (i.e., cell boundary).
CSI 0[**:***n*[**:***v*]]\ End of list of parallel texts (i.e. end of row).

PTX as Ruby text

CSI 1\ Beginning of principal text to be phonetically annotated. Beginning of table row, or beginning of base text for Ruby annotation. Any **:***u***:***s* is ignored.
CSI 3\ or **CSI 4\** End of principal text and beginning of Japanese (3) or Chinese (4) phonetic annotation text (Ruby text). There should be only one of these between **CSI 1\** and **CSI 5\**.
CSI 5\ End of Chinese/Japanese phonetic annotation.

Push/Pop

CSI ![push a copy of all currently explicitly set rendition attributes.
CSI *n***!]** pop the rendition attributes stack till it has *n* levels. If *n* is omitted, *n* is current stack size minus one. If *n* is **=1** (and push/pop is implemented), this is a “super-reset” to system defaults.

Other ECMA-48 related suggestions not detailed in this paper

SOS charset=*charset-name* **ST** Declare character encoding (akin to XML/HTML/CSS), non-ISO2022.

SOS lang=*IANA language tag* **ST** Language tagging.

APC hpgl:<*HP-GL commands or* `.hgl` *filename***>** **ST** Embed HP-GL/2 graphics. (Allow SGR styling for “labels”, with implicit **CSI 0m** before each label, as well as implicit styling push/pop surrounding the **APC … ST**.)

APC svg:<*SVG ‘commands’ or* `.svg` *filename***>** **ST** Embed SVG graphics.

Clickable links; like HTML’s <a href="…">…</a>.

Embedded images (.png, .jpg, …); like HTML’s <img ref="…"/>.

Defining page header and page footer for paginated uses.

SOS ST closes such non-CSI ECMA-48-based commands without “executing” them. (Note that they do not nest.) Useful for contexts where an APC/DCS/OSC/PM/SOS command string might not be terminated properly, such as at beginning of prompts, or used by the ‘reset’ command. Plain **ST** may also be used for such closings, but that may “execute” some of the command string (such as HP-GL commands). So a good start of prompt, for a terminal emulator oriented program, might be **ST CSI 0!m CSI 0m** if “style stacking” is not implemented, if it is: **ST CSI 0!m CSI =1!]**, but that also resets the colour and font palette, line indents and tab stops. (Spaces there are part of the notation, not explicit.)