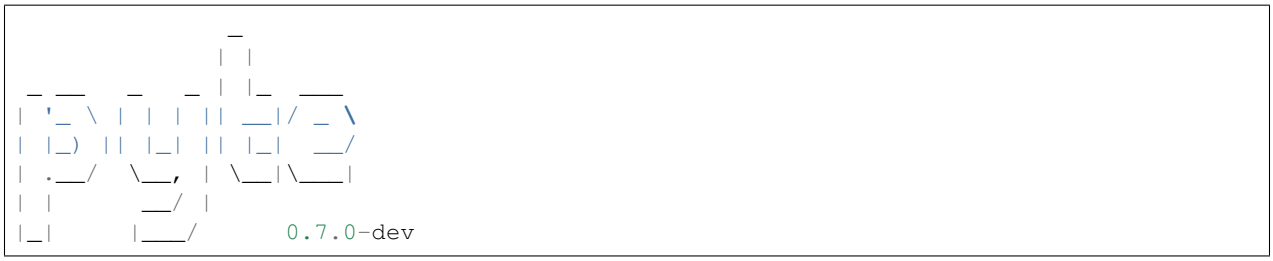
# pyte

*Release 0.7.0-dev*

**Jun 19, 2017**

# Contents




It's an in memory VTXXX-compatible terminal emulator. *XXX* stands for a series of video terminals, developed by DEC between 1970 and 1995. The first, and probably the most famous one, was VT100 terminal, which is now a de-facto standard for all virtual terminal emulators. `pyte` follows the suit.

So, why would one need a terminal emulator library?

- To screen scrape terminal apps, for example `htop` or `aptitude`.
- To write cross platform terminal emulators; either with a graphical (xterm, rxvt) or a web interface, like AjaxTerm.
- To have fun, hacking on the ancient, poorly documented technologies.

**Note**: `pyte` started as a fork of vt102, which is an incomplete pure Python implementation of VT100 terminal.

CHAPTER 1

# Installation

If you have pip you can do the usual:

```
pip install pyte
```

Otherwise, download the source from GitHub and run:

```
python setup.py install
```

CHAPTER 2

# Similar projects

`pyte` is not alone in the weird world of terminal emulator libraries, here's a few other options worth checking out: Termemulator, pyqonsole, webtty, AjaxTerm and of course vt102.

# CHAPTER 3

## pyte users

Believe it or not, there're projects which actually need a terminal emulator library. Not many of them use pyte, though. Here's a shortlist the ones that do:

- Ajenti – a webadmin panel for Linux and BSD, which uses pyte for its terminal plugin.
- Pymux – a terminal multiplexor.
- BastionSSH – a tool for protecting, monitoring and accessing multiple SSH resources.
- Jumpserver – an open source springboard machine(fortress machine): authentication, authorization, audit, automated operation and maintenance.

**Note:** Using pyte? Add yourself to this list and submit a pull request.

# CHAPTER 4

## Show me the code!

Head over to our brief *Tutorial* or, if you're feeling brave, dive right into the *API reference*; `pyte` also has a couple of examples in the examples directory.

## Tutorial

There are two important classes in `pyte`: *`Screen`* and `Stream`. The *Screen* is the terminal screen emulator. It maintains an in-memory buffer of text and text-attributes to display. The *Stream* is the stream processor. It processes the input and dispatches events. Events are things like `LINEFEED`, `DRAW "a"`, or `CURSOR_POSITION 10 10`. See the *API reference* for more details.

In general, if you just want to know what's being displayed on screen you can do something like the following:

```
>>> from __future__ import unicode_literals
>>> import pyte
>>> screen = pyte.Screen(80, 24)
>>> stream = pyte.Stream(screen)
>>> stream.feed(b"Hello World!")
>>> screen.display
    ['Hello World!
↪',
     '
↪',
     '
↪',
     '
↪',
     '
↪',
     '
↪',
     '
↪',
     '
↪',
```

```
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪',
      '
↪']
```

**Note**: `Screen` has no idea what is the source of bytes fed into `Stream`, so, obviously, it **can't read** or **change** environment variables, which implies that:

- it doesn't adjust *LINES* and *COLUMNS* on `"resize"` event;
- it doesn't use locale settings (*LC_** and *LANG*);
- it doesn't use *TERM* value and expects it to be *"linux"* and only *"linux"*.

And that's it for Hello World! Head over to the examples for more.

# API reference

## pyte.streams

This module provides three stream implementations with different features; for starters, here's a quick example of how streams are typically used:

```
>>> import pyte
>>> screen = pyte.Screen(80, 24)
>>> stream = pyte.Stream(screen)
>>> stream.feed("[5B")  # Move the cursor down 5 rows.
>>> screen.cursor.y
5
```

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

### pyte.Stream

**class** `pyte.`**`Stream`**(*screen=None*, *strict=True*)

A stream is a state machine that parses a stream of bytes and dispatches events based on what it sees.

**Parameters**

- **`screen`** (*`pyte.screens.Screen`*) – a screen to dispatch events to.
- **`strict`** (*`bool`*) – check if a given screen implements all required events.

---

**Note:** Stream only accepts text as input, but if for some reason you need to feed it with bytes, consider using `ByteStream` instead.

---

**See also:**

**man console_codes** For details on console codes listed bellow in `basic`, `escape`, `csi`, `sharp`.

### pyte.ByteStream

**class** `pyte.`**`ByteStream`**(**args*, ***kwargs*)

A stream which takes bytes as input.

Bytes are decoded to text using either UTF-8 (default) or the encoding selected via `select_other_charset()`.

**`use_utf8`**

Assume the input to `feed()` is encoded using UTF-8. Defaults to `True`.

## pyte.screens

This module provides classes for terminal screens, currently it contains three screens with different features:

- *`Screen`* – base screen implementation, which handles all the core escape sequences, recognized by `Stream`.
- If you need a screen to keep track of the changed lines (which you probably do need) – use *`DiffScreen`*.
- If you also want a screen to collect history and allow pagination – `pyte.screen.HistoryScreen` is here for ya ;)

---

**Note:** It would be nice to split those features into mixin classes, rather than subclasses, but it's not obvious how to do – feel free to submit a pull request.

---

**copyright**

3. 2011-2012 by Selectel.

### pyte.screens.Screen

**class** `pyte.screens.`**`Cursor`**(*x*, *y*, *attrs=Char(data=' ', fg='default', bg='default', bold=False, italics=False, underscore=False, strikethrough=False, reverse=False)*)

Screen cursor.

**Parameters**

- **x** (*int*) – 0-based horizontal cursor position.
- **y** (*int*) – 0-based vertical cursor position.
- **attrs** (`pyte.screens.Char`) – cursor attributes (see *select_graphic_rendition()* for details).

**class** `pyte.screens.`**`Char`**

A single styled on-screen character.

**Parameters**

- **data** (*str*) – unicode character. Invariant: `len(data) == 1`.
- **fg** (*str*) – foreground colour. Defaults to `"default"`.
- **bg** (*str*) – background colour. Defaults to `"default"`.
- **bold** (*bool*) – flag for rendering the character using bold font. Defaults to `False`.
- **italics** (*bool*) – flag for rendering the character using italic font. Defaults to `False`.
- **underline** (*bool*) – flag for rendering the character underlined. Defaults to `False`.
- **strikethrough** (*bool*) – flag for rendering the character with a strike-through line. Defaults to `False`.
- **reverse** (*bool*) – flag for swapping foreground and background colours during rendering. Defaults to `False`.

**class** `pyte.screens.`**`Screen`**(*columns*, *lines*)

A screen is an in-memory matrix of characters that represents the screen display of the terminal. It can be instantiated on its own and given explicit commands, or it can be attached to a stream and will respond to events.

**`buffer`**

A sparse `lines x columns` *Char* matrix.

**`dirty`**

A set of line numbers, which should be re-drawn. The user is responsible for clearing this set when changes have been applied.

```
>>> screen = Screen(80, 24)
>>> screen.dirty.clear()
>>> screen.draw("!")
>>> list(screen.dirty)
[0]
```

New in version 0.7.0.

**`cursor`**

Reference to the *Cursor* object, holding cursor position and attributes.

**margins**
Margins determine which screen lines move during scrolling (see *index()* and *reverse_index()*). Characters added outside the scrolling region do not make the screen to scroll.

The value is `None` if margins are set to screen boundaries, otherwise – a pair 0-based top and bottom line indices.

**charset**
Current charset number; can be either `0` or `1` for *G0* and *G1* respectively, note that *G0* is activated by default.

---

**Note:** According to `ECMA-48` standard, **lines and columns are 1-indexed**, so, for instance `ESC [ 10;10 f` really means – move cursor to position (9, 9) in the display matrix.

---

Changed in version 0.4.7.

**Warning:** *LNM* is reset by default, to match VT220 specification. Unfortunatelly this makes `pyte` fail `vttest` for cursor movement.

Changed in version 0.4.8.

**Warning:** If *DECAWM* mode is set than a cursor will be wrapped to the **beginning** of the next line, which is the behaviour described in `man console_codes`.

**See also:**

Standard ECMA-48, Section 6.1.1 for a description of the presentational component, implemented by `Screen`.

**default_char = Char(data=' ', fg='default', bg='default', bold=False, italics=False, underscore=False, strikethrough=F**
An empty character with default foreground and background colors.

**display**
A `list()` of screen lines as unicode strings.

**reset**()
Reset the terminal to its initial state.

- Scrolling margins are reset to screen boundaries.
- Cursor is moved to home location – `(0, 0)` and its attributes are set to defaults (see *default_char*).
- Screen is cleared – each character is reset to *default_char*.
- Tabstops are reset to "every eight columns".
- All lines are marked as *dirty*.

---

**Note:** Neither VT220 nor VT102 manuals mention that terminal modes and tabstops should be reset as well, thanks to *xterm* – we now know that.

---

**resize**(*lines=None*, *columns=None*)
Resize the screen to the given size.

If the requested screen size has more lines than the existing screen, lines will be added at the bottom. If the requested size has less lines than the existing screen lines will be clipped at the top of the screen. Similarly, if the existing screen has less columns than the requested screen, columns will be added at the right, and if it has more – columns will be clipped at the right.

---

**Note:** According to *xterm*, we should also reset origin mode and screen margins, see `xterm/screen.c:1761`.

---

**Parameters**

- **lines** (*int*) – number of lines in the new screen.
- **columns** (*int*) – number of columns in the new screen.

Changed in version 0.7.0: If the requested screen size is identical to the current screen size, the method does nothing.

**set_margins**(*top=None*, *bottom=None*)
Select top and bottom margins for the scrolling region.

**Parameters**

- **top** (*int*) – the smallest line number that is scrolled.
- **bottom** (*int*) – the biggest line number that is scrolled.

**set_mode**(**modes*, ***kwargs*)
Set (enable) a given list of modes.

**Parameters** **modes** (*list*) – modes to set, where each mode is a constant from *pyte.modes*.

**reset_mode**(**modes*, ***kwargs*)
Reset (disable) a given list of modes.

**Parameters** **modes** (*list*) – modes to reset – hopefully, each mode is a constant from *pyte.modes*.

**define_charset**(*code*, *mode*)
Define `G0` or `G1` charset.

**Parameters**

- **code** (*str*) – character set code, should be a character from `"B0UK"`, otherwise ignored.
- **mode** (*str*) – if `"("` `G0` charset is defined, if `")"` – we operate on `G1`.

**Warning:** User-defined charsets are currently not supported.

**shift_in**()
Select `G0` character set.

**shift_out**()
Select `G1` character set.

**draw**(*data*)
Display decoded characters at the current cursor position and advances the cursor if *DECAWM* is set.

**Parameters** **data** (*str*) – text to display.

Changed in version 0.5.0: Character width is taken into account. Specifically, zero-width and unprintable characters do not affect screen state. Full-width characters are rendered into two consecutive character containers.

**set_title**(*param*)
Set terminal title.

---

**Note:** This is an XTerm extension supported by the Linux terminal.

---

**set_icon_name**(*param*)
Set icon name.

---

**Note:** This is an XTerm extension supported by the Linux terminal.

---

**carriage_return**()
Move the cursor to the beginning of the current line.

**index**()
Move the cursor down one line in the same column. If the cursor is at the last line, create a new line at the bottom.

**reverse_index**()
Move the cursor up one line in the same column. If the cursor is at the first line, create a new line at the top.

**linefeed**()
Perform an index and, if *LNM* is set, a carriage return.

**tab**()
Move to the next tab space, or the end of the screen if there aren't anymore left.

**backspace**()
Move cursor to the left one or keep it in its position if it's at the beginning of the line already.

**save_cursor**()
Push the current cursor position onto the stack.

**restore_cursor**()
Set the current cursor position to whatever cursor is on top of the stack.

**insert_lines**(*count=None*)
Insert the indicated # of lines at line with cursor. Lines displayed **at** and below the cursor move down. Lines moved past the bottom margin are lost.

**Parameters** **count** – number of lines to insert.

**delete_lines**(*count=None*)
Delete the indicated # of lines, starting at line with cursor. As lines are deleted, lines displayed below cursor move up. Lines added to bottom of screen have spaces with same character attributes as last line moved up.

**Parameters** **count** (*int*) – number of lines to delete.

**insert_characters**(*count=None*)
Insert the indicated # of blank characters at the cursor position. The cursor does not move and remains at the beginning of the inserted blank characters. Data on the line is shifted forward.

**Parameters** **count** (*int*) – number of characters to insert.

**delete_characters**(*count=None*)
Delete the indicated # of characters, starting with the character at cursor position. When a character is deleted, all characters to the right of cursor move left. Character attributes move with the characters.

**Parameters** **count** (*int*) – number of characters to delete.

**erase_characters**(*count=None*)
Erase the indicated # of characters, starting with the character at cursor position. Character attributes are set cursor attributes. The cursor remains in the same position.

**Parameters** **count** (*int*) – number of characters to erase.

---

**Note:** Using cursor attributes for character attributes may seem illogical, but if recall that a terminal emulator emulates a type writer, it starts to make sense. The only way a type writer could erase a character is by typing over it.

---

**erase_in_line**(*how=0*, *private=False*)
Erase a line in a specific way.

Character attributes are set to cursor attributes.

**Parameters**

- **how** (*int*) – defines the way the line should be erased in:
  - `0` – Erases from cursor to end of line, including cursor position.
  - `1` – Erases from beginning of line to cursor, including cursor position.
  - `2` – Erases complete line.
- **private** (*bool*) – when `True` only characters marked as eraseable are affected **not implemented**.

**erase_in_display**(*how=0*, *private=False*)
Erases display in a specific way.

Character attributes are set to cursor attributes.

**Parameters**

- **how** (*int*) – defines the way the line should be erased in:
  - `0` – Erases from cursor to end of screen, including cursor position.
  - `1` – Erases from beginning of screen to cursor, including cursor position.
  - `2` and `3` – Erases complete display. All lines are erased and changed to single-width. Cursor does not move.
- **private** (*bool*) – when `True` only characters marked as eraseable are affected **not implemented**.

**set_tab_stop**()
Set a horizontal tab stop at cursor position.

**clear_tab_stop**(*how=0*)
Clear a horizontal tab stop.

**Parameters** **how** (*int*) – defines a way the tab stop should be cleared:

- `0` or nothing – Clears a horizontal tab stop at cursor position.
- `3` – Clears all horizontal tab stops.

**ensure_hbounds**()
Ensure the cursor is within horizontal screen bounds.

**ensure_vbounds**(*use_margins=None*)
Ensure the cursor is within vertical screen bounds.

**Parameters** **use_margins** (*bool*) – when `True` or when *DECOM* is set, cursor is bounded by top and and bottom margins, instead of `[0; lines - 1]`.

**cursor_up**(*count=None*)
Move cursor up the indicated # of lines in same column. Cursor stops at top margin.

**Parameters** **count** (*int*) – number of lines to skip.

**cursor_up1**(*count=None*)
Move cursor up the indicated # of lines to column 1. Cursor stops at bottom margin.

**Parameters** **count** (*int*) – number of lines to skip.

**cursor_down**(*count=None*)
Move cursor down the indicated # of lines in same column. Cursor stops at bottom margin.

**Parameters** **count** (*int*) – number of lines to skip.

**cursor_down1**(*count=None*)
Move cursor down the indicated # of lines to column 1. Cursor stops at bottom margin.

**Parameters** **count** (*int*) – number of lines to skip.

**cursor_back**(*count=None*)
Move cursor left the indicated # of columns. Cursor stops at left margin.

**Parameters** **count** (*int*) – number of columns to skip.

**cursor_forward**(*count=None*)
Move cursor right the indicated # of columns. Cursor stops at right margin.

**Parameters** **count** (*int*) – number of columns to skip.

**cursor_position**(*line=None*, *column=None*)
Set the cursor to a specific *line* and *column*.

Cursor is allowed to move out of the scrolling region only when *DECOM* is reset, otherwise – the position doesn't change.

**Parameters**

- **line** (*int*) – line number to move the cursor to.
- **column** (*int*) – column number to move the cursor to.

**cursor_to_column**(*column=None*)
Move cursor to a specific column in the current line.

**Parameters** **column** (*int*) – column number to move the cursor to.

**cursor_to_line**(*line=None*)
Move cursor to a specific line in the current column.

**Parameters** **line** (*int*) – line number to move the cursor to.

**bell**(**args*)
Bell stub – the actual implementation should probably be provided by the end-user.

**alignment_display**()
Fills screen with uppercase E's for screen focus and alignment.

**select_graphic_rendition**(**attrs*)
Set display attributes.

**Parameters** **attrs** (*list*) – a list of display attributes to set.

**report_device_attributes**(*mode=0*, ***kwargs*)
Report terminal identity.

New in version 0.5.0.

Changed in version 0.7.0: If `private` keyword argument is set, the method does nothing. This behaviour is consistent with VT220 manual.

**report_device_status**(*mode*)
Report terminal status or cursor position.

**Parameters** **mode** (*int*) – if 5 – terminal status, 6 – cursor position, otherwise a noop.

New in version 0.5.0.

**write_process_input**(*data*)
Write data to the process running inside the terminal.

By default is a noop.

**Parameters** **data** (*str*) – text to write to the process `stdin`.

New in version 0.5.0.

**debug**(**args*, ***kwargs*)
Endpoint for unrecognized escape sequences.

By default is a noop.

### pyte.screens.DiffScreen

**class** pyte.screens.**DiffScreen**(**args*, ***kwargs*)
A screen subclass, which maintains a set of dirty lines in its `dirty` attribute. The end user is responsible for emptying a set, when a diff is applied.

Deprecated since version 0.7.0: The functionality contained in this class has been merged into *Screen* and will be removed in 0.8.0. Please update your code accordingly.

### pyte.screens.HistoryScreen

**class** pyte.screens.**History**(*top*, *bottom*, *ratio*, *size*, *position*)

**class** pyte.screens.**HistoryScreen**(*columns*, *lines*, *history=100*, *ratio=0.5*)
A :class:~‘pyte.screens.Screen‘ subclass, which keeps track of screen history and allows pagination. This is not linux-specific, but still useful; see page 462 of VT520 User’s Manual.

**Parameters**

- **history** (*int*) – total number of history lines to keep; is split between top and bottom queues.
- **ratio** (*int*) – defines how much lines to scroll on *next_page()* and *prev_page()* calls.

**history**
A pair of history queues for top and bottom margins accordingly; here’s the overall screen structure:

```
[ 1: .......]
[ 2: .......]  <- top history
[ 3: .......]
------------
[ 4: .......]  s
[ 5: .......]  c
[ 6: .......]  r
[ 7: .......]  e
[ 8: .......]  e
[ 9: .......]  n
------------
[10: .......]
[11: .......]  <- bottom history
[12: .......]
```

**Note:** Don't forget to update `Stream` class with appropriate escape sequences – you can use any, since pagination protocol is not standardized, for example:

```
Stream.escape["N"] = "next_page"
Stream.escape["P"] = "prev_page"
```

**before_event**(*event*)
Ensure a screen is at the bottom of the history buffer.

**Parameters** **event** (*str*) – event name, for example `"linefeed"`.

**after_event**(*event*)
Ensure all lines on a screen have proper width (`columns`).

Extra characters are truncated, missing characters are filled with whitespace.

**Parameters** **event** (*str*) – event name, for example `"linefeed"`.

**reset**()
Overloaded to reset screen history state: history position is reset to bottom of both queues; queues themselves are emptied.

**erase_in_display**(*how=0*)
Overloaded to reset history state.

**index**()
Overloaded to update top history with the removed lines.

**reverse_index**()
Overloaded to update bottom history with the removed lines.

**prev_page**()
Move the screen page up through the history buffer. Page size is defined by `history.ratio`, so for instance `ratio = .5` means that half the screen is restored from history on page switch.

**next_page**()
Move the screen page down through the history buffer.

### pyte.screens.DebugScreen

**class** pyte.screens.**DebugScreen**(*to=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *only=()*)

A screen which dumps a subset of the received events to a file.

```
>>> import io
>>> with io.StringIO() as buf:
...     stream = Stream(DebugScreen(to=buf))
...     stream.feed("\x1b[1;24r\x1b[4l\x1b[24;1H\x1b[0;10m")
...     print(buf.getvalue())
...
...
["set_margins", [1, 24], {}]
["reset_mode", [4], {}]
["cursor_position", [24, 1], {}]
["select_graphic_rendition", [0, 10], {}]
```

**Parameters**

- **to** (*file*) – a file-like object to write debug information to.
- **only** (*list*) – a list of events you want to debug (empty by default, which means – debug all events).

**Warning:** This is developer API with no backward compatibility guarantees. Use at your own risk!

## pyte.modes

This module defines terminal mode switches, used by *Screen*. There're two types of terminal modes:

- *non-private* which should be set with `ESC [ N h`, where `N` is an integer, representing mode being set; and
- *private* which should be set with `ESC [ ? N h`.

The latter are shifted 5 times to the right, to be easily distinguishable from the former ones; for example *Origin Mode* – *DECOM* is `192` not `6`.

```
>>> DECOM
192
```

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

pyte.modes.**LNM = 20**

*Line Feed/New Line Mode*: When enabled, causes a received *LF*, *pyte.control.FF*, or *VT* to move the cursor to the first column of the next line.

pyte.modes.**IRM = 4**

*Insert/Replace Mode*: When enabled, new display characters move old display characters to the right. Characters moved past the right margin are lost. Otherwise, new display characters replace old display characters at the cursor position.

pyte.modes.**DECTCEM = 800**
*Text Cursor Enable Mode*: determines if the text cursor is visible.

pyte.modes.**DECSCNM = 160**
*Screen Mode*: toggles screen-wide reverse-video mode.

pyte.modes.**DECOM = 192**
*Origin Mode*: allows cursor addressing relative to a user-defined origin. This mode resets when the terminal is powered up or reset. It does not affect the erase in display (ED) function.

pyte.modes.**DECAWM = 224**
*Auto Wrap Mode*: selects where received graphic characters appear when the cursor is at the right margin.

pyte.modes.**DECCOLM = 96**
*Column Mode*: selects the number of columns per line (80 or 132) on the screen.

## pyte.control

This module defines simple control sequences, recognized by `Stream`, the set of codes here is for `TERM=linux` which is a superset of VT102.

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

pyte.control.**SP = ' '**
*Space*: Not suprisingly – `" "`.

pyte.control.**NUL = '\x00'**
*Null*: Does nothing.

pyte.control.**BEL = '\x07'**
*Bell*: Beeps.

pyte.control.**BS = '\x08'**
*Backspace*: Backspace one column, but not past the begining of the line.

pyte.control.**HT = '\t'**
*Horizontal tab*: Move cursor to the next tab stop, or to the end of the line if there is no earlier tab stop.

pyte.control.**LF = '\n'**
*Linefeed*: Give a line feed, and, if *pyte.modes.LNM* (new line mode) is set also a carriage return.

pyte.control.**VT = '\x0b'**
*Vertical tab*: Same as *LF*.

pyte.control.**FF = '\x0c'**
*Form feed*: Same as *LF*.

pyte.control.**CR = '\r'**
*Carriage return*: Move cursor to left margin on current line.

pyte.control.**SO = '\x0e'**
*Shift out*: Activate G1 character set.

pyte.control.**SI = '\x0f'**
*Shift in*: Activate G0 character set.

pyte.control.**CAN = '\x18'**
*Cancel*: Interrupt escape sequence. If received during an escape or control sequence, cancels the sequence and displays substitution character.

pyte.control.**SUB = '\x1a'**
*Substitute*: Same as *CAN*.

pyte.control.**ESC = '\x1b'**
*Escape*: Starts an escape sequence.

pyte.control.**DEL = '\x7f'**
*Delete*: Is ignored.

pyte.control.**CSI = '\x9b'**
*Control sequence introducer*: An equivalent for `ESC [`.

pyte.control.**ST = '\x9c'**
*String terminator*.

pyte.control.**OSC = '\x9d'**
*Operating system command*.

## pyte.escape

This module defines both CSI and non-CSI escape sequences, recognized by `Stream` and subclasses.

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

pyte.escape.**RIS = 'c'**
*Reset*.

pyte.escape.**IND = 'D'**
*Index*: Move cursor down one line in same column. If the cursor is at the bottom margin, the screen performs a scroll-up.

pyte.escape.**NEL = 'E'**
*Next line*: Same as *pyte.control.LF*.

pyte.escape.**HTS = 'H'**
Tabulation set: Set a horizontal tab stop at cursor position.

pyte.escape.**RI = 'M'**
*Reverse index*: Move cursor up one line in same column. If the cursor is at the top margin, the screen performs a scroll-down.

pyte.escape.**DECSC = '7'**
Save cursor: Save cursor position, character attribute (graphic rendition), character set, and origin mode selection (see *DECRC*).

pyte.escape.**DECRC = '8'**
*Restore cursor*: Restore previously saved cursor position, character attribute (graphic rendition), character set, and origin mode selection. If none were saved, move cursor to home position.

pyte.escape.**DECALN = '8'**
*Alignment display*: Fill screen with uppercase E's for testing screen focus and alignment.

pyte.escape.**ICH = '@'**
*Insert character*: Insert the indicated # of blank characters.

pyte.escape.**CUU = 'A'**
*Cursor up*: Move cursor up the indicated # of lines in same column. Cursor stops at top margin.

pyte.escape.**CUD = 'B'**
*Cursor down*: Move cursor down the indicated # of lines in same column. Cursor stops at bottom margin.

pyte.escape.**CUF = 'C'**
*Cursor forward*: Move cursor right the indicated # of columns. Cursor stops at right margin.

pyte.escape.**CUB = 'D'**
*Cursor back*: Move cursor left the indicated # of columns. Cursor stops at left margin.

pyte.escape.**CNL = 'E'**
*Cursor next line*: Move cursor down the indicated # of lines to column 1.

pyte.escape.**CPL = 'F'**
*Cursor previous line*: Move cursor up the indicated # of lines to column 1.

pyte.escape.**CHA = 'G'**
*Cursor horizontal align*: Move cursor to the indicated column in current line.

pyte.escape.**CUP = 'H'**
*Cursor position*: Move cursor to the indicated line, column (origin at `1, 1`).

pyte.escape.**ED = 'J'**
*Erase data* (default: from cursor to end of line).

pyte.escape.**EL = 'K'**
*Erase in line* (default: from cursor to end of line).

pyte.escape.**IL = 'L'**
*Insert line*: Insert the indicated # of blank lines, starting from the current line. Lines displayed below cursor move down. Lines moved past the bottom margin are lost.

pyte.escape.**DL = 'M'**
*Delete line*: Delete the indicated # of lines, starting from the current line. As lines are deleted, lines displayed below cursor move up. Lines added to bottom of screen have spaces with same character attributes as last line move up.

pyte.escape.**DCH = 'P'**
*Delete character*: Delete the indicated # of characters on the current line. When character is deleted, all characters to the right of cursor move left.

pyte.escape.**ECH = 'X'**
*Erase character*: Erase the indicated # of characters on the current line.

pyte.escape.**HPR = 'a'**
*Horizontal position relative*: Same as *CUF*.

pyte.escape.**DA = 'c'**
*Device Attributes*.

pyte.escape.**VPA = 'd'**
*Vertical position adjust*: Move cursor to the indicated line, current column.

pyte.escape.**VPR = 'e'**
*Vertical position relative*: Same as *CUD*.

pyte.escape.**HVP = 'f'**
*Horizontal / Vertical position*: Same as *CUP*.

pyte.escape.**TBC = 'g'**
*Tabulation clear*: Clears a horizontal tab stop at cursor position.

pyte.escape.**SM = 'h'**
*Set mode*.

pyte.escape.**RM = 'l'**
*Reset mode*.

pyte.escape.**SGR = 'm'**
*Select graphics rendition*: The terminal can display the following character attributes that change the character display without changing the character (see *pyte.graphics*).

pyte.escape.**DSR = 'n'**
*Device status report*.

pyte.escape.**DECSTBM = 'r'**
*Select top and bottom margins*: Selects margins, defining the scrolling region; parameters are top and bottom line. If called without any arguments, whole screen is used.

pyte.escape.**HPA = '"'**
*Horizontal position adjust*: Same as *CHA*.

## pyte.graphics

This module defines graphic-related constants, mostly taken from *console_codes(4)* and http://pueblo.sourceforge.net/doc/manual/ansi_color_codes.html.

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

pyte.graphics.**TEXT = {27: '-reverse', 1: '+bold', 3: '+italics', 4: '+underscore', 22: '-bold', 7: '+reverse', 24: '-underscore**
A mapping of ANSI text style codes to style names, "+" means the: attribute is set, "-" – reset; example:

```
>>> text[1]
'+bold'
>>> text[9]
'+strikethrough'
```

pyte.graphics.**FG_ANSI = {32: 'green', 33: 'brown', 34: 'blue', 35: 'magenta', 36: 'cyan', 37: 'white', 39: 'default', 30: 'b**
A mapping of ANSI foreground color codes to color names.

```
>>> FG_ANSI[30]
'black'
>>> FG_ANSI[38]
'default'
```

pyte.graphics.**FG = {32: 'green', 33: 'brown', 34: 'blue', 35: 'magenta', 36: 'cyan', 37: 'white', 39: 'default', 30: 'black', 3**
An alias to *FG_ANSI* for compatibility.

pyte.graphics.**FG_AIXTERM = {96: 'cyan', 97: 'white', 90: 'black', 91: 'red', 92: 'green', 93: 'brown', 94: 'blue', 95: 'mag**
A mapping of non-standard `aixterm` foreground color codes to color names. These are high intensity colors and thus should be complemented by `+bold`.

pyte.graphics.**BG_ANSI = {49: 'default', 40: 'black', 41: 'red', 42: 'green', 43: 'brown', 44: 'blue', 45: 'magenta', 46: 'cya**
A mapping of ANSI background color codes to color names.

```
>>> BG_ANSI[40]
'black'
>>> BG_ANSI[48]
'default'
```

pyte.graphics.**BG = {49: 'default', 40: 'black', 41: 'red', 42: 'green', 43: 'brown', 44: 'blue', 45: 'magenta', 46: 'cyan', 47:**
An alias to *BG_ANSI* for compatibility.

pyte.graphics.**BG_AIXTERM = {100: 'black', 101: 'red', 102: 'green', 103: 'brown', 104: 'blue', 105: 'magenta', 106: 'cya**
A mapping of non-standard `aixterm` background color codes to color names. These are high intensity colors and thus should be complemented by `+bold`.

pyte.graphics.**FG_256 = 38**
SGR code for foreground in 256 or True color mode.

pyte.graphics.**BG_256 = 48**
SGR code for background in 256 or True color mode.

pyte.graphics.**FG_BG_256 = ['000000', 'cd0000', '00cd00', 'cdcd00', '0000ee', 'cd00cd', '00cdcd', 'e5e5e5', '7f7f7f', 'ff00**
A table of 256 foreground or background colors.

## pyte.charsets

This module defines `G0` and `G1` charset mappings the same way they are defined for linux terminal, see `linux/drivers/tty/consolemap.c` @ http://git.kernel.org

---

**Note:** `VT100_MAP` and `IBMPC_MAP` were taken unchanged from linux kernel source and therefore are licensed under **GPL**.

---

**copyright**

3. 2011-2012 by Selectel.

**copyright** (c) 2012-2017 by pyte authors and contributors, see AUTHORS for details.

**license** LGPL, see LICENSE for more details.

pyte.charsets.**LAT1_MAP = '\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17**
Latin1.

pyte.charsets.**VT100_MAP = '\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x1**
VT100 graphic character set.

pyte.charsets.**IBMPC_MAP = '\x00•♪¶§↑↓→← !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUV**
IBM Codepage 437.

pyte.charsets.**VAX42_MAP = '\x00•♪¶§↑↓→← "#$%&\'()*+,-./0123456789:;<=>@ABCDEFGHIJKLMNOPQRSTUVW**
VAX42 character set.

# pyte Changelog

Here you can see the full list of changes between each pyte release.

## Version 0.7.0-dev

- Removed deprecated `only` parameter of `Stream.attach`.
- Removed deprecated `encoding` parameter of `ByteStream`.
- Fixed `how == 3` handling in `DiffScreen.erase_in_display`.
- Deprecated `DiffScreen`. Its functionality has been backported to the base `Screen` class.
- Fixed a bug in `DiffScreen.draw` which incorrectly handled the case when the input of `draw` required several lines.
- Fixed a bug in `Screen` which did not ignore `ESC (` argument in UTF8 mode. See issue #88 on GitHub.
- Changed `Screen.resize` to do nothing if the requested size matches the current one.
- Disallowed private mode for `Screen.report_device_attributes`. This was causing an infinite loop in Emacs and Vim. See issue #81 on GitHub.
- Fixed a bug in *OSC* parsing, which caused `Stream` to hang upon receiving a palette reset request `ESC ] R`.

## Version 0.6.0

Released on May 28th 2017

This release is NOT backward compatible with 0.5.X branch!

- Optimized `Stream.feed` for plain-text input. The code was backported from `pymux` project by Jonathan Slenders.
- Optimized `Screen` by changing `Screen.buffer` to dict-of-dicts. The idea was borrowed from `pymux` project by Jonathan Slenders. The implementation was done by @istarion.
- Further optimized `Stream._parser_fsm` by forcing static binding between `Stream` events and `Screen` methods. The code was backported from `pmux` project by Jonathan Slenders.
- Restricted `Stream` to a single listener and deprecated `attach` and `detach`. The old logic can be emulated by a fanout proxy, forwarding events to a list of its listeners.
- Replaced `DebugStream` with `DebugScreen` to workaround the single listener limitation (previsouly `DebugStream` implicitly added a listener when instantiated). Unlike other screens `DebugScreen` does not maintain any state.
- Changed `DebugScreen` to emit JSON instead of custom text format.
- Removed overly generic `Screen.__before__` and `Screen.__after__`.
- Renamed `Screen.set_charset` to a more appropriate `Screen.define_charset`.
- Added support for ECMA-035 *DOCS* command to `ByteStream` which no longer accepts `encoding` as an argument and instead sets it as instructed by *DOCS*. The default encoding is assumed to be UTF-8.
- Added support for OSC sequences allowing to set terminal title and icon name.
- Allowed 256 and 24bit colours in `Screen.select_graphic_rendition`.
- Added support for `aixterm` colours in `Screen.select_graphic_rendition`, see issue #57 on GitHub.
- Changed `Screen.select_graphic_rendition` to ignore 0 if it is given along with other attributes, ie `"0;1;2"` is now equivalent to `"1;2"`.

- Fixed rendering of multicolumn characters at ‘‘Screen‘ boundaries. Thanks to @shaform! See PR #55 on GitHub.
- Fixed `Screen.display` in the case of multicolumn characters. See issue #52 on GitHub.
- Fixed *DECSTBM* handling in case of missing arguments. See issue #61 on GitHub.
- Fixed the way `Screen.cursor_up` and `Screen.cursor_down` interact with the scrolling region. See #63 on GitHub.
- Added a minimal web terminal example by @demiurg906. For a faster and more fully-featured version, see demiurg906/pyte_gui.
- Fixed `Screen.cursor_back` when called after the draw in the last column.
- Fixed `Screen.inser_characters` when called with an argument larger than the number of columns. Thanks to @istarion! See PR #74 on GitHub.
- Fixed `Screen.erase_in_display` which did not handle all values supported by `TERM=linux`. See #80 on GitHub.

## Version 0.5.2

Pi Day bugfix release, released on March 14th, 2016

- Fixed a bug in handling DA request. See issue #46 on GitHub.

## Version 0.5.1

Bugfix release, released on January 10th 2015

- Fixed dependencies in setup.py.

## Version 0.5.0

Released on January 10th 2015

- Deprecated `Stream.consume` in favour of `Stream.feed`. The latter allows for a more efficient implementation because it operates on the whole input string.
- Improved `Stream` performance by converting FSM to a coroutine as suggested by Jonathan Slenders in issue #41 on GitHub.
- Added support for *DA* (device attributes) and *DSR* (device status report). The implementation is based on the code by Jonathan Slenders. See issue #41 on GitHub.
- `Screen.draw` now properly handles full/ambiguous-width characters. Thanks to the excellent wcwidth library by Jeff Quast.
- Removed re-exports of abbreviated modules (e.g. `mo` as a synonym for `modes`) from `pyte`.
- Removed `Screen.size` which misleadingly returned constructor arguments in reverse order. Please use `Screen.columns` and `Screen.lines` instead.
- Fixed a bug in `ByteStream` which suppressed the exception if all of the decoders failed to process the input.

## Version 0.4.10

Bugfix release, released on August 4th 2015

- Fixed a bug in `DiffScreen.draw` which marked the wrong line as changed when *DECAWM* was enabled.
- `Stream` now recognizes ESC % sequences for selecting control character set. However, these operations are no-op in the current version in a sense that `ByteStream` does not handle them to change encoding.

## Version 0.4.9

Bugfix release, released on December 3rd 2014

- Fixed a bug in `Char` initialization, see issue #24 on GitHub for details.
- Updated error message in `Stream`, referencing `str` is relevant for Python 3, but not Python 2.

## Version 0.4.8

Released on January 13th 2014

- `Screen` does NOT inherit from builtin `list`, use `Screen.buffer` to access individual characters directly. This is a backward INCOMPATIBLE change.
- `Char._asdict` was broken on Python 3.3 because of the changes in `namedtuple` implementation.
- `LAT1_MAP` was an iterator because of the change in map semantics in Python 3.
- Changed `Screen` to issues a CR in addition to LF when *DECAWM* mode is set and the cursor is at the right border of the screen. See http://www.vt100.net/docs/vt510-rm/DECAWM and issue #20 on GitHub for details.

## Version 0.4.7

Bugfix release, released on March 28th 2013

- Updated `pyte` and tests suite to work under Python 3.3.
- Changed `Screen` so that *LNM* mode is reset by default, see http://www.vt100.net/docs/vt510-rm/LNM and issue #11 on GitHub for details.

## Version 0.4.6

Bugfix release, released on February 29th 2012

## Version 0.4.5

Technical release, released on September 1st 2011

- Added MANIFEST.in and CenOS spec file

## Version 0.4.4

Bugfix release, released on July 17th 2011

- Removed `pdb` calls, left from `HistoryScreen` debugging – silly, I know :)

## Version 0.4.3

Bugfix release, released on July 12th 2011

- Fixed encoding issues in `DebugStream` – Unicode was not converted to bytes properly.
- Fixed G0-1 charset handling and added VAX42 charset for the ancient stuff to work correctly.

## Version 0.4.2

Bugfix release, released on June 27th 2011

- Added a tiny debugging helper: `python -m pyte your escape codes`
- Added `Screen.__{before,after}__()` hooks to `Screen` – now subclasses can extend more than one command easily.
- Fixed `HistoryScreen` – now not as buggy as it used to be: and allows for custom ratio aspect when browsing history, see `HistoryScreen` documentation for details.
- Fixed *DECTCEM* private mode handling – when the mode is reset `Screen.cursor.hidden` is `True` otherwise it's `False`.

## Version 0.4.1

Bugfix release, released on June 21st 2011

- Minor examples and documentation update before the first public release.

## Version 0.4.0

Released on June 21st 2011

- Improved cursor movement – `Screen` passes all but one tests in *vttest*.
- Changed the way `Stream` interacts with `Screen` – event handlers are now implicitly looked up in screen's `__dict__`, not connected manually.
- Changed cursor API – cursor position and attributes are encapsulated in a separate `Cursor` class.
- Added support for *DECSCNM* – toggle screen-wide reverse-video mode.
- Added a couple of useful `Screen` subclasses: `HistoryScreen` which allows screen pagination and `DiffScreen` which tracks the changed lines.

## Version 0.3.9

Released on May 31st 2011

- Added initial support for G0-1 charsets (mappings taken from `tty` kernel driver) and SI, SO escape sequences.
- Changed `ByteStream` to support fallback encodings – it now takes a list of `(encoding, errors)` pairs and traverses it left to right on `feed()`.
- Switched to `unicode_literals` – one step closer to Python3.

## Version 0.3.8

Released on May 23rd 2011

- Major rewrite of `Screen` internals – highlights: inherits from `list`; each character is represented by `namedtuple` which also holds SGR data.
- Numerous bugfixes, especially in methods, dealing with manipulating character attributes.

## Version 0.3.7

First release after the adoption – skipped a few version to reflect that. Released on May 16th 2011

- Added support for ANSI color codes, as listed in `man console_codes`. Not implemented yet: setting alternate font, setting and resetting mappings, blinking text.
- Added a couple of trivial usage examples in the *examples/* dir.

# Python Module Index

## p

# Index

## V



## W