Blessings Documentation

Release 1.7

Erik Rose

Jun 21, 2018

Contents

CHAPTER 1

Read The Readme First

This is the API documentation for the Blessings terminal library. Because Blessings uses quite a bit of dynamism, you should read the readme first, as not all the useful stuff shows up in these autogenerated docs. However, if you're looking for seldom-used keyword args or a peek at the internals, you're in the right place.

CHAPTER 2

Then Read This If You Want

class blessings.**Terminal**(*kind=None*, *stream=None*, *force_styling=False*)

An abstraction around terminal capabilities

Unlike curses, this doesn't require clearing the screen before doing anything, and it's friendlier to use. It keeps the endless calls to `tigetstr()` and `tparm()` out of your code, and it acts intelligently when somebody pipes your output to a non-terminal.

Instance attributes:

stream The stream the terminal outputs to. It's convenient to pass the stream around with the terminal; it's almost always needed when the terminal is and saves sticking lots of extra args on client functions in practice.

__getattr__(*attr*)

Return a terminal capability, like bold.

For example, you can say `term.bold` to get the string that turns on bold formatting and `term.normal` to get the string that turns it off again. Or you can take a shortcut: `term.bold('hi')` bolds its argument and sets everything to normal afterward. You can even combine things: `term.bold_underline_red_on_bright_green('yowzers!')`.

For a parametrized capability like `cup`, pass the parameters too: `some_term.cup(line, column)`.

`man terminfo` for a complete list of capabilities.

Return values are always Unicode.

__init__(*kind=None*, *stream=None*, *force_styling=False*)

Initialize the terminal.

If `stream` is not a tty, I will default to returning an empty Unicode string for all capability values, so things like piping your output to a file won't strew escape sequences all over the place. The `ls` command sets a precedent for this: it defaults to columnar output when being sent to a tty and one-item-per-line when not.

Parameters

- **kind** – A terminal string as taken by `setupterm()`. Defaults to the value of the `TERM` environment variable.
- **stream** – A file-like object representing the terminal. Defaults to the original value of stdout, like `curses.initscr()` does.
- **force_styling** – Whether to force the emission of capabilities, even if we don't seem to be in a terminal. This comes in handy if users are trying to pipe your output through something like `less -r`, which supports terminal codes just fine but doesn't appear itself to be a terminal. Just expose a command-line option, and set `force_styling` based on it. Terminal initialization sequences will be sent to `stream` if it has a file descriptor and to `sys.__stdout__` otherwise. (`setupterm()` demands to send them somewhere, and stdout is probably where the output is ultimately headed. If not, stderr is probably bound to the same terminal.)

  If you want to force styling to not happen, pass `force_styling=None`.

color
Return a capability that sets the foreground color.

The capability is unparametrized until called and passed a number (0-15), at which point it returns another string which represents a specific color change. This second string can further be called to color a piece of text and set everything back to normal afterward.

Parameters **num** – The number, 0-15, of the color

fullscreen(***kwds*)
Return a context manager that enters fullscreen mode while inside it and restores normal mode on leaving.

height
The height of the terminal in characters

If no stream or a stream not representing a terminal was passed in at construction, return the dimension of the controlling terminal so piping to things that eventually display on the terminal (like `less -R`) work. If a stream representing a terminal was passed in, return the dimensions of that terminal. If there somehow is no controlling terminal, return `None`. (Thus, you should check that the property `is_a_tty` is true before doing any math on the result.)

hidden_cursor(***kwds*)
Return a context manager that hides the cursor while inside it and makes it visible on leaving.

location(***kwds*)
Return a context manager for temporarily moving the cursor.

Move the cursor to a certain position on entry, let you print stuff there, then return the cursor to its original position:

```
term = Terminal()
with term.location(2, 5):
    print 'Hello, world!'
    for x in xrange(10):
        print 'I can do it %i times!' % x
```

Specify `x` to move to a certain column, `y` to move to a certain row, both, or neither. If you specify neither, only the saving and restoration of cursor position will happen. This can be useful if you simply want to restore your place after doing some manual cursor movement.

number_of_colors
Return the number of colors the terminal supports.

Common values are 0, 8, 16, 88, and 256.

Though the underlying capability returns -1 when there is no color support, we return 0. This lets you test more Pythonically:

```
if term.number_of_colors:
    ...
```

We also return 0 if the terminal won't tell us how many colors it supports, which I think is rare.

on_color
Return a capability that sets the background color.

See `color()`.

width
The width of the terminal in characters

See `height()` for some corner cases.

Index

Symbols

C

F

H

L

N

O

T

W