PyBoard Editor: Simple Onboard Editor for fast fixes

Recently I stumbled over PyBoard, purchased it, and started to work with it. It's really nice, but the sequence of developing code for it was a little by annoying. Yes, you can work in safe mode, but editing the code on the board's flash from the host was not reliable. Keeping and editing the code on the host and copying it to the board works better, but you have to carefully wait for the files to be sync'ed. If not, they may be corrupted. BTW: a jumper or switch that causes the board to boot in safe mode by default would be nice. And not working in safe mode is even more cumbersome, because you have to switch states, and yes, the interface number may change if you go through reset. I'm a little bit nervous about pressing the switches bringing the board under mechanical stress all the time. So I though, that having a simple editor on board which I could use for small fixes (like the line of code that failed) would be nice. I made my way through the Micropython forum and found pfalkon's Python editor code, which I took and ported it to PyBoard. It's really impressive how few lines of code pfalkon needed to implement a reasonable amount of functionality. Since the code looked clean, and it seemed so easy to add features, I could not resist adding a little bit, and yes, it got a little bit larger. The file size increased by a factor of 3 (only some of that caused by commenting), and the footprint in memory by a factor of 1,5 to 2 (now 12k), depending on what you keep. It still contains the code for the Linux environment, so you can run it in Linux MicroPython (if you install the os module) or Python3 for testing. I sprayed C Preprocessor statements in it (arrgh!), so you can use cpp to remove the stuff which is not needed for PyBoard. So, what did I change and add:

- Use USB_VCP or UART for input and output.
- Changed the read keyboard function to comply with slow char-by-char input on serial lines.
- Added support for TAB, BACKTAB, SAVE, Find, Replace, Goto Line, Yank (delete line into buffer), Zap (insert buffer)
- Join Lines by Delete char at the end or Backspace at the beginning, Autoindent for Enter
- Moved main into a function with some optional parameters
- Added a status line and single line prompts for Quit, Save, Find, Replace and Goto. The status line can be turned (almost) off for slow connections.

The editor works in Insert mode. Cursor Keys, Home, End, PgUp and PgDn work as you would expect. Most functions are available with Ctrl-Keys too, if a keyboard mapping is not available. Keyboard Mapping:

| Keys | Alternative | Function |
|---|---|---|
| Up | Ctrl-K | Move the cursor up one line |
| Down | Ctrl-J | Move the cursor down one line |
| Left | Ctrl-H | Move the cursor left by one char. The terminal should have the BACKSPACE key defined as \x7f avoiding conflicts with Backspace |
| PgUp | Ctrl-O | Move the cursor up by one screen height |
| PgDn | Ctrl-P | Move the cursor down by one screen height |
| Home | Ctrl-W | Move the cursor towards the start of the line. If the cursor is on the right of the first non-space, it moves to the first non-space, otherwise it moves to the start of the line |
| End | Ctrl-E | Move to the End of the Line |
| Enter | | Insert a line break at the cursor position. Auto-indent is supported. |
| Backspace | | Delete the char left hand to the cursor. At the beginning of the line Backspace joins the current line with the previous line. |

| | | |
|---|---|---|
| Del | Ctrl-Y | Delete char under cursor- If the cursor is at the end of the line, join the next line. (Vi lovers might prefer Ctrl-X instead: just change KEYMAP) |
| Ctrl-Q | Ctr-C | Quit the editor or the line edit mode. If the edited text was changed, it will prompt for confirmation. |
| Ctrl-S | | Save to file. The file name will be prompted. Saving to internal flash is really slow, so be warned. |
| Ctrl-F | | Find text. The search is not case sensitive. The last search string is memorized. Search stops at the end. |
| Ctrl-N | | Repeat find starting at the column right to the cursor. |
| Ctrl-R | | Find and replace. The find part is not case sensitive, but replace is. |
| Ctrl-G | | Go to Line. The line number will be prompted. |
| Ctrl-B | | Go to last line (kind of obsolete, goto line 1 will do the same). |
| Ctrl-T | | Go to first line (kind of obsolete, goto a huge line number will do the same) |
| Ctrl-X | Ctrl-Del | Delete Line and keep it. A sequence of these will keep all lines in the order they were deleted. Any other key will start over the game. |
| Ctrl-V | | Insert the deleted lines before the actual line. Together with the Ctrl-X this implements a very basic Cut & Paste feature, to be treated with care. |
| Tab | | Insert a tab, context sensitive. If the cursor is left hand to the first non-space, it shifts the line to the right, such that the first non-space is at a tab position (the editor targets Python code), and the cursor does not move. If at or after the first non-space, it inserts spaces at the cursor position up to the next tab location, moving the cursor. |
| BackTab | Ctrl-U | Backtab, context sensitive. If the cursor is left hand to the first non-space, it shifts the line to the left, such that the first non-space is at a tab position, and the cursor does not move. If at or after the first non-space, it remove spaces left to the cursor position up to the next tab location, if possible, and moves the cursor. |
| Ctrl-A | | Toggles auto-indent. |

The editor is contained in the file pye.py. Start Pye from the REPL prompt e.g. with

```
Import pye
pye.pye([file [,content]])
```

Instead of giving a file name, you can supply a list of strings as the second (or optional) parameter and supply an empty string instead. Pye.pye() will give you an empty screen to fill with your thoughts. Optional parameters:

| | |
|---|---|
| content=list | The content to be edited is supplied in a list of strings. No type checking!. The default is None |
| tabsize=x | Tab step. The default is 4 |
| status=True/False | Toggle visibility of the status line (True/False). The default is True. But even w/o status line, it is shown once after Goto, Find, Save or any error message. |
| device=n | Device to be used for screen/keyboard. 0 means USB_VCP. 1 means UART 1, and so on. The default is 0 (USB_VCP). |
| baud=n | Line's baud rate. This setting is effective only for UART. The default is 38400 |

The file pye.py is pretty large. As told, it contains C pre-processor statements allowing trimming it down a little bit. For that reason, comments start with ## instead of #. So for PyBoard, you might run:

cpp –D PYBOARD –D DEFINES pye.py >pe.py

That will result in a file with all functions supplied, but smaller footprint when loaded. The directive DEFINES will replace symbolic key names with numeric constants, reducing the demand for symbol space. You may strip down the file size (not the P-code footprint) by removing comments and empty lines (that's what I do), e.g. by:

cpp -D PYBOARD -D DEFINES pye.py | sed "s/#.*$//" | sed "/^$/d" >pe.py

Doing that will removes dead code like the one for the Linux environment

If the footprint is still too large, you may choose:

cpp -D BASIC -D PYBOARD -D DEFINES pye.py | sed "s/#.*$//" | sed "/^$/d" >pe.py

That removes the code for Delete Line, Insert buffer, Tab, Backtab, Go to first Line, go to last line, and auto-indent toggle. There are still lines left as “if sys.platform == “pyboard”. If you do not like these, delete them manually (and take care of the indents). If you just want to get rid of the cpp preprocessor stuff, run:

cpp -D PYBOARD -D DEFINES pye.py | grep -v "^#.*$" >pe.py

**Backup:**

**The keyboard mapping assumes VT100. For those interested, I collected a the key codes issue by terminal emulators, all claiming VT100 compatible:**

| Key | Putty VT100 & Xterm | Putty esc-[~ | Putty Linux | Minicom | GtkTerm | Picocom | Linux Terminal |
|---|---|---|---|---|---|---|---|
| **Up** | \e[A | \e[A | \e[A | \e[A | \e[A | \e[A | \e[A |
| **Down** | \e[B | \e[B | \e[B | \e[B | \e[B | \e[B | \e[B |
| **Left** | \e[D | \e[D | \e[D | \e[D | \e[D | \e[D | \e[D |
| **Right** | \e[C | \e[C | \e[C | \e[C | \e[C | \e[C | \e[C |
| **Home** | \e[1~ | \e[1~ | \e[1~ | \e[1~ | \eOH | \eOH | \e[H |
| **End** | \e[4~ | \e[4~ | \e[4~ | \eOF | \eOF | \eOF | \e[F |
| **Ins** | \e[2~ | \e[2~ | \e[2~ | \e[2~ | \e[2~ | \e[2~ | \e[2~ |
| **Del** | \e[3~ | \e[3~ | \e[3~ | \e[3~ | \e[3~ | \e[3~ | \e[3~ |
| **PgUp** | \e[5~ | \e[5~ | \e[5~ | \e[5~ | \e[5~ | \e[5~ | \e[5~ |
| **PgDn** | \e[6~ | \e[6~ | \e[6~ | \e[6~ | \e[6~ | \e[6~ | \e[6~ |
| **Backspace** | \x7f | \x7f | \x7f | \x7f | \x08 | \x7f | \x7f |
| **Ctrl-Home** | | | | \e[1;5H | | \e[1;5H | \e[1;5H |
| **Ctrl-End** | | | | \e[1;5H | | \e[1;5F | \e[1;5F |
| **Ctrl-Del** | | | | \e[3;5~ | \e[3;5~ | \e[3;5~ | \e[3;5~ |
| **Tab** | \x09 | \x09 | \x09 | \x09 | \x09 | \x09 | \x09 |
| **BackTab** | \e[Z | \e[Z | \e[Z | \e[Z | \e[Z | \e[Z | \e[Z |
| **F1** | \eOP | \e[11~ | \e[[A | -- Calls Linux Terminal Help | | | |
| **F2** | \eOQ | \e[12~ | \e[[B | \eOQ | \eOQ | \eOQ | \eOQ |

Picocom seems sometimes to send the Linux Terminal codes. Nevertheless, I’m using it most of the times. If the KEYMAP is too large, and you know which terminal you are working on, delete or comment out the obsolete lines. If your terminal is different, just change the control codes.

**Notes:**

- For those who wonder why sending data to the screen on pyboard is more than a simple write(): for USB_VCP, it stumbles over a large amount of data to be sent in short time. The

difference is, that USB.write() waits internally until all has been sent, whereas USB_VCP.write() stops when it cannot send more data. So we have to see what's coming back. And, b.t.w., UART.write() does not like empty strings, which in turn is accepted by USB_VCP.write().

- Ctrl-H is mapped to cursor left. There are terminal who have the backspace key unchangeable mapped to Ctrl-H. For these, you may change KEYMAP accordingly.
- For Delete Line/Insert Line the key mapping is now Ctrl-X/Ctrl-V. I played a while with Ctrl-Y/Ctrl-Z, which is also kind of natural. And it keeps Ctrl-X free for Delete Char.
- When reading files, tab characters (\x09) in the text are replaced by spaces, tab size 8, and tab characters at the end of a line are discarded. When you save the file, it will NOT contain any tab characters. So be careful when editing files which need tab character. I added an overly complicated looking version of expandtabs, which hopefully needs less dynamic stack than a simple one.
- There is a junkyard at the end of the file with some code pieces that you might find useful. If you drop working code, just put it there. At the moment you'll find:
  a) A version of find supporting regular expression. If you prefer this one, just swap with the other find_in_file() definition above.
  b) Main code for starting it from the command line. I'm beginning to use it instead of nano whenever I have to do short edits in text mode.

**To Do:**

- Resist adding more features! Better think of what can be dropped (like help), slimmed or cleaned. And obviously bug fixing.
- Any suggestions?