

# pbook.el

Luke Gorrie, with modifications by Paul Khuong

February 3, 2007

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>‘pbook’</b>	<b>1</b>
<b>3</b>	<b>Prelude</b>	<b>1</b>
<b>4</b>	<b>Emacs commands</b>	<b>2</b>
<b>5</b>	<b>Configurable variables</b>	<b>3</b>
5.1	Configuration variables for code formatting . . . . .	4
5.2	Configuration for the translation of properties to Latex . . . . .	5
<b>6</b>	<b>Top-level logic</b>	<b>9</b>
<b>7</b>	<b>Escaping special characters</b>	<b>11</b>
<b>8</b>	<b>Processing engine</b>	<b>12</b>
8.1	Heading formatting . . . . .	13
8.2	Commentary formatting . . . . .	14
8.3	Source code formatting . . . . .	15
<b>9</b>	<b>Prologue and file variables</b>	<b>19</b>

## 1 Introduction

Have you ever printed out a program and read it on paper?

It is an interesting exercise to try with one of your own programs, one that you think is well-written. The first few times you will probably find that it’s torture to try and read in a straight line. What seemed so nice in Emacs is riddled with glaring problems on paper.

How a program reads on paper may not be very important in itself, but there is wonderful upside to this. If you go through the program with a red pen and fix all the mind-bendingly obvious problems you see, what happens is that the program greatly improves – not just on paper, but also in Emacs!

This is a marvellously effective way to make programs better.

Let’s explore the idea some more!

## 2 ‘pbook’

This program, `pbook`, is a tool for making readable programs by generating LaTeX'ified program listings. Its purpose is to help you improve your programs by making them read well on paper. It serves this end by generating pretty-looking PDF output for you to print out and attack with a red pen, and perhaps use the medium to trick your mind into seeking the clarity of a technical paper and bringing your prose-editing skills to bear on your source code.

`pbook` is aware of three things: headings, top-level comments, and code. Headings become LaTeX sections, and have entries in a table of contents. Top-level comments become plain text in a nice variable-width font. Other source code is listed as-is in a fixed-width font.

These different elements are distinguished in the source using maximally unobtrusive markup, which you can see at work in the `pbook.el` source code.

Read on to see the program and how it works.

## 3 Prelude

(I have successfully tested this program with GNU Emacs versions 20.7 and 21.3, and with XEmacs version 21.5.)

This is actually not true anymore. I have tested this in GNU Emacs 22.??? and 21.???. While I don't expect there to be any portability problem, it has not been tested in XEmacs at all.

For some tiny luxuries and portability help we use the Common Lisp compatibility library:

```
1 (require 'cl)
```

## 4 Emacs commands

A handful of Emacs commands make up the `pbook` user-interface. The most fundamental is to render a `pbook`-formatted Emacs buffer as LaTeX.

```
1 (defun PBOOK-BUFFER ()
2   "Generate LaTeX from the current (pbook-formatted) buffer.
3   The resulting source is displayed in a buffer called *pbook*."
4   (interactive)
5   (pbook-process-buffer))
```

A very handy utility is to display a summary of the buffer's structure and use it to jump to an appropriate section. I've always enjoyed being able to do this in `texinfo`-mode. Happily, `pbook` gets this for free using the `occur` function, which lists all lines in the buffer that match some regular expression.

```
1 (defun PBOOK-SHOW-STRUCTURE ()
2   "Display the pbook heading structure of the current buffer."
3   (interactive)
4   (occur pbook-heading-regexp))
```

To avoid a lot of mucking about in the shell there is also a command to generate and display a PDF file. This function is a quick hack to make experimentation easy.

I should add a function to do the same with dvis, and to output to non-temporary files.

```
1 (defun PBOOK-BUFFER-VIEW-PDF ()
2   "Generate and display PDF from the current buffer.
3   The intermediate files are created in the standard temporary
4   directory."
5   (interactive)
6   (save-window-excursion
7     (pbook-buffer)
8     (with-current-buffer "*pbook*"
9       (let ((texfile (pbook-tmpfile "pbook" "tex"))
10             (pdffile (pbook-tmpfile "pbook" "pdf"))
11             (idxfile (pbook-tmpfile "pbook" "idx"))))
12         (write-region (point-min) (point-max) texfile)
13         ;; Possibly there is a better way to ensure that LaTeX generates
14         ;; the table of contents correctly than to run it more than
15         ;; once, but I don't know one.
16         (shell-command (format "\
17 cd /tmp; latex %s && \
18 makeindex %s && \
19 pdflatex %s && acroread %s &"
20                         texfile
21                         idxfile
22                         texfile pdffile))))))

24 (defun PBOOK-TMPFILE (name extension)
25   "Return the full path to a temporary file called NAME and with EXTENSION.
26   An appropriate directory is chosen and the PID of Emacs is inserted
27   before the extension."
28   (format "%s%s-%S.%s"
29           (if (boundp 'temporary-file-directory)
30               temporary-file-directory
31               ;; XEmacs does it this way instead:
32               (temp-directory))
33           name (emacs-pid) extension))
```

## 5 Configurable variables

These are variables that can be customized to affect pbook's behaviour. The default regular expressions assume Lisp-style comment characters, but they can be overridden with buffer-local bindings from hooks for other programming modes. The other variables that control formatting are best configured with Emacs's magic "file variables" (see down the very bottom for an example).

```

1 (defvar PBOOK-COMMENTARY-REGEXP "^;;;\\(\$\\|\\[\\^#]\\)"
2   "Regular expression matching lines of high-level commentary.")

4 (defvar PBOOK-HEADING-REGEXP "^;;;\\(#+\\)"
5   "Regular expression matching heading lines of chapters/sections/headings.")

7 (defvar PBOOK-HEADING-LEVEL-SUBEXP 1
8   "The subexpression of 'pbook-heading-regexp' whose length indicates nesting.")

10 (defvar PBOOK-INCLUDE-TOC t
11   "When true include a table of contents.")

13 (defvar PBOOK-STYLE 'article
14   "Style of output. Either article (small) or book (large).")

16 (defvar PBOOK-AUTHOR (user-full-name)
17   "The name to use in the \\author LaTeX command.")

```

## 5.1 Configuration variables for code formatting

```

1 (defvar PBOOK-CODE-PROLOGUE "\
2  \\vspace{1pc}
3  \\begin{adjustwidth}{0in}{-1.5in}
4  \\begin{flushleft}
5  "
6   "Tex string to prepend to code listings")

8 (defvar PBOOK-CODE-EPILOGUE "\
9  \\end{flushleft}
10 \\end{adjustwidth}
11 \\vspace{1pc}
12 "
13   "Tex string to append to code listings")

15 (defvar PBOOK-CURRENT-LINE nil
16   "Holds the line number being processed. Note that this
17 is reset for every new section of code. This variable
18 is only accessible while processing code lines, obviously.")

20 (defvar PBOOK-CURRENT-TOTAL-LINES nil
21   "Holds the total number of lines in the section of
22 code that's being processed.")

```



```

1 (defvar PBOOK-CURRENT-TEXT-PROPERTIES nil
2   "text-properties of the current region of text. Maybe be used
3   by the property transformation functions to fine-tune their
4   output.")

```

The most common customisation will be to change the way faces are shown on paper. By default, they are translated faithfully: color, slantedness and boldness are directly translated. While a value of `nil` means don't care (and defers to other faces properties or default property values), `:no`, by default, disables the associated property.

```

1 (defvar PBOOK-FACE-OVERRIDE '(font-lock-keyword-face :bold t :index :no)
2   (font-lock-builtin-face :bold t :index :no)
3   (font-lock-function-name-face
4     :sc t :tt :no
5     :index function)
6   (font-lock-variable-name-face
7     :sc t :tt :no
8     :index variable)
9   (font-lock-warning-face :bold t :index :no)
10  (font-lock-comment-face :italic t :tt :no :index :no)
11  (font-lock-doc-face :italic t :tt :no :index :no)
12  (font-lock-constant-face :italic :no :index :no)
13  (font-lock-string-face :index :no)
14  (default :italic :no))
15  "Alist that associates a face with a set of default properties.")

17 (defvar PBOOK-ESCAPING-REGEXPS '(("<" . "\\\textless{")
18   (">" . "\\\textgreater{")
19   ("\\\\" . "\\\textbackslash{")
20   ("~" . "\\\textasciitilde{")
21   ("\\^" . "\\\textasciicircum{")
22   ("[#%&$_{}]" . "\\\&")) ;; space added as needed in pbook-latex-es
23  "alist of regexp -> replacement (passed to re-search-forward and replace-match)
24  A simple way to index FIXMEs would be to add a regex for that in this list.")

```

`pbook-properties` defines pbook properties: their name, when they are applied (by default), and how they are translated into Latex. It is a list of triples (`[name]` `[default-value]` `[translator]`). `[name]` is an unique identifier for the property. `[default-value]` is an unary function that, given the face, returns the value to associate with the property (or `nil` to defer). `[translator]` is either an unary function that, given the property's value, returns a list of a string to prepend to the formatted region, a string to append to it, and any number of pairs as in `pbook-escaping-regexps`. Latex code is spliced outside (for the first property) in (for the last property).

```

1 (defvar PBOOK-PROPERTIES
2   '(:color
3     pbook-face-color
4     (lambda (colors)
5       (if (or (null colors)
6             (eq pbook-monochrome t)
7             (every (lambda (color)
8                   (< color 0.01))
9                   colors))
10          nil
11          (let ((components (mapcar (lambda (color)
12                                    (if (< color 0.01)
13                                        "0"
14                                        (number-to-string color)))
15                                    (or pbook-monochrome
16                                        colors))))
17            (list (concat "\\textcolor[rgb]{",
18                    (first components)
19                    ", ",
20                    (second components)
21                    ", ",
22                    (third components)
23                    "}")
24                  "))))))

26 (:bold face-bold-p
27       (lambda (prop)
28         (and (not (member prop '(nil :no)))
29              (looking-at " *[^ ]")
30              '("\\textbf{" " }"))))

32 (:italic face-italic-p
33       (lambda (prop)
34         (and (not (member prop '(nil :no)))
35              (looking-at " *[^ ]")
36              '("\\textit{" " }"))))

38 (:tt (lambda (face)
39        t)
40       (lambda (prop)
41         (when (or (not (looking-at " *[^ ]")) ;; always \tt whitespace.
42                 (not (member prop '(nil :no))))
43           '("\\texttt{" " }"))))

45 (:sc (lambda (face)
46        nil)
47       ("\\textsc{" " }"))

```

```

49   (:index (lambda (face)
50             (and (eq face 'default)
51                  (eq (plist-get pbook-current-text-properties
52                          'syntax-table)
53                          'pbook-identifier)
54                          'use))
55             (lambda (index)
56               (when (and index
57                       (not (eq index :no)))
58                 (let* ((pbook-escaping-regexps (list* (cons "[!@|]" "\\|\\&")
59                                                         pbook-escaping-regexps))
60                       (word (pbook-latex-escape-string (buffer-string))))
61                   (list (format "\\index{%s%s}" word (ecase index
62                                                         ((function) "|bb")
63                                                         ((variable) "|ii")
64                                                         ((use) "")))
65                         "}}))))))
66   )
67   "Complex system. See paragraph above.")

69 (defun PBOOK-FACE-COLOR (face)
70   "Given a face, return a triplet of rgb values. Flips the luminance
71   as needed (to adapt dark background colours to a light background)."
72   (let ((dark-bg-p (or (and (boundp 'face-background-mode)
73                             (eq face-background-mode 'dark))
74                         (member (face-background face)
75                                 pbook-dark-colors))))
76     (and (face-foreground face)
77          (let ((rgb-specs (mapcar (lambda (n)
78                                    (/ n 65535.0))
79                                  (color-values (face-foreground face))))))
80          (and rgb-specs
81               (if dark-bg-p
82                 (let ((yuv-specs (apply 'pbook-rgb-yuv rgb-specs)))
83                   (pbook-yuv-rgb (* 0.25 (- 1 (first yuv-specs)))
84                                   (second yuv-specs)
85                                   (third yuv-specs)))
86                 rgb-specs))))))

```

```

88 (defun PBOOK-RGB-YUV (r g b)
89   "As http://en.wikipedia.org/wiki/YUV - Matrix fixed by mjp."
90   (let* ((y (+ (* 0.299 r)
91                (* 0.587 g)
92                (* 0.114 b)))
93          (u (+ (* -0.168740 r)
94                (* -0.331260 g)
95                (* 0.500000 b)))
96          (v (+ (* 0.500000 r)
97                (* -0.418690 g)
98                (* -0.081310 b))))
99   (list y u v)))

101 (defun PBOOK-YUV-RGB (y u v)
102   "As http://en.wikipedia.org/wiki/YUV - Matrix fixed by mjp."
103   (let ((r (+ y
104              (* 1.40200 v)))
105         (g (+ y
106              (* -0.34413 u)
107              (* -0.71414 v)))
108         (b (+ y
109              (* 1.77200 u))))
110     (mapcar (lambda (x)
111               (cond ((< x 0) 0.0) ;; Need to clamp values for some reason
112                     ((> x 1) 1.0)
113                     (t x)))
114             (list r g b))))
115

```

## 6 Top-level logic

Here we have the top level of the program. Setting up, calling the formatting engine, piecing things together, and putting on the finishing touches.

The real work is done in a new buffer called `*pbook*`. First the source is fontified, then copied into this buffer and from there it is massaged into shape.

Most of this is mundane, but there is one tricky part: the source buffer may have buffer-local values for some pbook settings, and we have to be careful or we'd lose them when switching into the `*pbook*` buffer. This is taken care of by moving the correct values of all the relevant customizable settings into new dynamic bindings.

```

1 (defun PBOOK-PROCESS-BUFFER ()
2   "Generate pbook output for the current buffer
3   The output is put in the buffer *pbook* and displayed."
4   (interactive)
5   (let ((font-lock-syntactic-keywords
6         (append font-lock-syntactic-keywords
7                 pbook-font-lock-override)))
8     (setq font-lock-fontified nil) ;; pretend buffer isn't fontified
9     (font-lock-default-fontify-buffer) ;; HACK!!! Looks like an internal...
10    )
11    (let ((buffer (current-buffer))
12          (beginning (pbook-tex-beginning))
13          (ending (pbook-tex-ending))
14          (text (buffer-string)))
15      (with-current-buffer (get-buffer-create "*pbook*")
16        ;; Setup,
17        (pbook-inherit-buffer-locals buffer
18          '(pbook-commentary-regexp
19            pbook-heading-regexp
20            pbook-style
21            pbook-heading-level-subexp
22            pbook-include-toc
23            pbook-monochrome
24            pbook-font-lock-override))
25        (erase-buffer)
26        (insert text)
27        ;; Reformat as LaTeX,
28        (pbook-preprocess)
29        (pbook-format-buffer)
30        ;; Insert header & footer.
31        (goto-char (point-min))
32        (insert beginning)
33        (goto-char (point-max))
34        (insert ending)
35        (display-buffer (current-buffer))))))

37 (defun PBOOK-INHERIT-BUFFER-LOCALS (buffer variables)
38   "Make buffer-local bindings of VARIABLES using the values in BUFFER."
39   (dolist (v variables)
40     (set (make-local-variable v)
41         (with-current-buffer buffer (symbol-value v))))))

```

```

43 (defun PBOOK-PREPROCESS ()
44   "Cleanup the buffer to prepare for formatting."
45   (goto-char (point-min))
46   ;; FIXME: Currently we just zap all pagebreak characters.
47   (save-excursion
48     (while (re-search-forward "\C-1" nil t)
49       (replace-match "")))
50   (unless (re-search-forward pbook-heading-regexp nil t)
51     (error "File must have at least one heading."))
52   (beginning-of-line)
53   ;; Delete everything before the first heading.
54   (delete-region (point-min) (point)))

56 (defun PBOOK-TEX-BEGINNING ()
57   "Return the beginning prelude for the LaTeX output."
58   (format "\
59 \\documentclass[notitlepage,a4paper]{%s}
60 \\usepackage[nohead,nofoot]{geometry}
61 \\usepackage{color}
62 \\usepackage{bold-extra}
63 \\usepackage{chngepage}
64 \\usepackage{index}
65 \\newcommand{\\ii}[1]{\\it #1}
66 \\newcommand{\\bb}[1]{\\bf #1}
67 \\newcommand{\\t}[1]{\\tt #1}
68 \\makeindex
69 \\title{%s}
70 \\author{%s}
71 \\begin{document}
72 \\maketitle
73 %s\n"
74         (symbol-name pbook-style)
75         (pbook-latex-escape-string (buffer-name))
76         (pbook-latex-escape-string pbook-author)
77         (if pbook-include-toc "\\tableofcontents" "")))

79 (defun PBOOK-TEX-ENDING ()
80   "Return the ending of the LaTeX output."
81   "\
82 \\printindex
84 \\end{document}\n")

```

## 7 Escaping special characters

We have to escape characters that LaTeX treats specially. This is done based on `pbook-escaping-regexps`, which is defined according to the rules in the `Special Characters` node of the LaTeX2e info

manual. (CHECKME)

```
1 (defun PBOOK-LATEX-ESCAPE-STRING (string &optional space)
2   (with-temp-buffer
3     (insert string)
4     (pbook-latex-escape (point-min) (point-max) space)
5     (buffer-string)))

7 (defun PBOOK-LATEX-ESCAPE (start end &optional space)
8   "LaTeX-escape special characters in the region from START to END."
9   (when (or space
10         pbook-escaping-regexps)
11     (let* ((pbook-escaping-regexps (if space
12                                       (append pbook-escaping-regexps
13                                               (list (cons " " "\\\\\\\&"))
14                                               pbook-escaping-regexps))
15           (scan-regex (apply 'concat
16                             (car (first pbook-escaping-regexps))
17                             (mapcan (lambda (entry)
18                                     (list "\\|"
19                                           (car entry)))
20                                     (rest pbook-escaping-regexps))))))
21       (save-excursion
22         (save-restriction
23           (narrow-to-region start end)
24           (goto-char start)
25           (while (re-search-forward scan-regex
26                                   nil t)
27                 (goto-char (match-beginning 0))
28                 (dolist (entry pbook-escaping-regexps)
29                   (let ((test (car entry))
30                         (replace (cdr entry)))
31                     (when (looking-at test)
32                       (replace-match replace))))))))))
```

## 8 Processing engine

The main loop scans through the source buffer piece by piece and converts each one to LaTeX as it goes. There are three sorts of pieces: headings, top-level commentary, and code.

This loop recognises what type of piece is at the point and then calls the appropriate subroutine. The subroutines are responsible for determining where their piece finishes and for advancing the point beyond the region they have formatted.

```

1 (defun PBOOK-FORMAT-BUFFER ()
2   (while (not (eobp))
3     (if (looking-at "^\\s *$")
4       ;; Skip blank lines.
5       (forward-line)
6       (cond ((looking-at pbook-heading-regexp)
7             (pbook-do-heading))
8             ((looking-at pbook-commentary-regexp)
9             (pbook-do-commentary))
10            (t
11            (pbook-do-code))))))

```

## 8.1 Heading formatting

Each heading line is converted to a LaTeX sectioning command. The heading text is escaped.

```

1 (defun PBOOK-DO-HEADING ()
2   ;; NB: 'looking-at' sets the Emacs match data (for match-string, etc)
3   (assert (looking-at pbook-heading-regexp))
4   (let ((depth (length (match-string-no-properties pbook-heading-level-subexp))))
5     ;; Strip off the comment characters and whitespace.
6     (replace-match "")
7     (when (looking-at "\\s +")
8       (replace-match ""))
9     (pbook-latex-escape (line-beginning-position) (line-end-position))
10    (wrap-line (format "\\%s{" (pbook-nth-sectioning-command depth)
11              "}))
12    (forward-line))

14 (defun WRAP-LINE (prefix suffix)
15   "Insert PREFIX at the start of the current line and SUFFIX at the end."
16   (save-excursion
17     (goto-char (line-beginning-position))
18     (insert prefix)
19     (goto-char (line-end-position))
20     (insert suffix))

```

LaTeX has different sectioning commands for articles and books, so we have to choose from the right set. These variables define the sets in order of nesting – the first element is top-level, etc.

```

1 (defconst PBOOK-ARTICLE-SECTIONING-COMMANDS
2   '("section" "subsection" "subsubsection")
3   "LaTeX commands for sectioning articles.")

```

```

5 (defconst PBOOK-BOOK-SECTIONING-COMMANDS
6   (cons "chapter" pbook-article-sectioning-commands)
7   "LaTeX commands for sectioning books.")

9 (defun PBOOK-NTH-SECTIONING-COMMAND (n)
10  "Return the sectioning command for nesting level N (top-level is 1)."
```

```

11  (let ((commands (ecase pbook-style
12                  (article pbook-article-sectioning-commands)
13                  (book    pbook-book-sectioning-commands))))
14    (nth (min (1- n) (1- (length commands))) commands)))
```

## 8.2 Commentary formatting

Top-level commentary is stripped of its comment characters and we escape all characters that LaTeX treats specially.

```

1 (defun PBOOK-DO-COMMENTARY ()
2   "Format one or more lines of commentary into LaTeX."
3   (assert (looking-at pbook-commentary-regexp))
4   (let ((start (point)))
5     ;; Strip off comment characters line-by-line until end of section.
6     (while (or (looking-at pbook-commentary-regexp)
7               (and (looking-at "^\\s *$")
                     (not (eobp))))
9       (replace-match ""))
10    (delete-horizontal-space)
11    (forward-line)
12    (save-excursion
13      (pbook-latex-escape start (point))
14      (pbook-pretty-commentary start (point))))))
```

These functions define a simple Wiki-like markup language for basic formatting.

```

1 (defun PBOOK-PRETTY-COMMENTARY (start end)
2   "Make commentary prettier."
3   (save-restriction
4     (narrow-to-region start end)
5     (goto-char (point-min))
6     (save-excursion (pbook-pretty-tt))
7     (save-excursion (pbook-pretty-doublequotes))))

9 (defun PBOOK-PRETTY-TT ()
10  "Format 'single quoted' text with a typewriter font."
11  (while (re-search-forward "[^']*" nil t)
12    (replace-match "{\\\\tt \\1}" t)))
```

```

14 (defun PBOOK-PRETTY-DOUBLEQUOTES ()
15   "Format \"double quoted\" text with \"double single quotes\"."
16   (while (re-search-forward "\\([^\"]*\\)\"" nil t)
17     (replace-match "'\1'")))

```

### 8.3 Source code formatting

Source text is rendered as defined in pbook-properties.

```

1 (defun PBOOK-DO-CODE ()
2   (assert (and (not (looking-at pbook-commentary-regexp))
3               (not (looking-at pbook-heading-regexp))))
4   (let ((start (point))
5         (end   (progn
6                 (pbook-goto-end-of-code)
7                 (point))))
8     (save-restriction
9       (narrow-to-region start end)
10      (pbook-convert-tabs-to-spaces start end)
11      ;; delete trailing newlines and spaces
12      (goto-char (point-max))
13      (while (or (equal (char-syntax (char-before)) " ")
14                (bolp))
15        (delete-char -1))
16      (pbook-format-code start (point-max)
17                        (count-lines start (point-max)))
18      (goto-char (point-min))
19      (insert pbook-code-prologue)
20      (goto-char (point-max))
21      (insert "\n" pbook-code-epilogue "\n"))))

23 (defun PBOOK-GOTO-END-OF-CODE ()
24   "Goto the end of the current section of code."
25   (if (re-search-forward (format "\\(%s\\)\\|\\(%s\\)"
26                           pbook-heading-regexp
27                           pbook-commentary-regexp)
28       nil t)
29       (beginning-of-line)
30       (goto-char (point-max))))

32 (defun PBOOK-CONVERT-TABS-TO-SPACES (start end)
33   "Replace tab characters with spaces."
34   (save-excursion
35     (save-restriction
36       (narrow-to-region start end)
37       (untabify start end))))

```

```

40 (defun PBOOK-FORMAT-CODE (start end num-lines)
41   "Format the section of code. The third argument is
42 the total number of line in the section."
43   (save-excursion
44     (save-restriction
45       (narrow-to-region start end)
46       (goto-char (point-min))
47       (let ((cur-line 0))
48         (while (< cur-line num-lines)
49           (pbook-format-line (line-beginning-position) (line-end-position)
50                             cur-line num-lines)
51           (incf cur-line)
52           (beginning-of-line 2))))))

54 (defun PBOOK-GET-INHERITS (face)
55   "Flattens a face's inheritance list, in order."
56   (let ((faces (cond ((eq face 'unspecified) nil)
57                     ((listp face)          face)
58                     (t                      (list face)))))
59     (mapcan (lambda (face)
60              (let ((inherits (face-attribute face :inherit)))
61                (if (and inherits
62                        (not (eq inherits 'unspecified)))
63                    (cons face (pbook-get-inherits inherits))
64                    (list face))))
65              faces)))

67 (defun PBOOK-TRANSLATE-FACE-PROPERTIES (face props)
68   "Updates the list of properties 'props' with those
69 associates with 'face'. 'pbook-face-override' has priority"
70   (setq props (append props
71                       (copy-list (cdr (assoc face pbook-face-override)))))
72   (dolist (defn pbook-properties)
73     (let ((prop-name (first defn))
74           (predicate (second defn)))
75       (unless (plist-get props prop-name)
76         (let ((value (funcall predicate face)))
77           (when value
78             (setq props (plist-put props prop-name value)))))))
79   props)

```

```

81 (defun PBOOK-FACE-PROPERTIES (face)
82   "Finds a face's (and those from which it inherits) pbook
83 properties. Earlier faces in the inheritance list (preorder
84 depth-first) have priority."
85   (let ((faces (append (pbook-get-inherits face)
86                       '(default))))
87     (props nil))
88   (dolist (face faces props)
89     (setq props (pbook-translate-face-properties face props))))

91 (defun PBOOK-PROPERTIES-LATEX-STRINGS (plist)
92   "Given a plist of pbook properties, finds the latex
93 strings with which to wrap the text that is being formatted,
94 and the additional regexps with which to escape it."
95   (let* ((pbook-face-latex-properties plist) ;; special var
96          (prepend nil)
97          (append nil)
98          (regexps nil) ;; escaping-regexp entry
99          )
100    (dolist (property pbook-properties (list (apply 'concat
101                                              (reverse prepend))
102                                             (apply 'concat append)
103                                             regexps))
104      (let* ((name (first property))
105             (transformer (third property))
106             (foundp (plist-member plist name))
107             (prop (plist-get plist name)))
108        (when foundp
109          (let ((wrap (if (and (listp transformer)
110                              (not (eq (first transformer)
111                                        'lambda)))
112                          (and prop
113                              (not (eq prop :no))
114                              transformer)
115                          (funcall transformer prop))))
116            (when wrap
117              (push (first wrap) prepend)
118              (push (second wrap) append)
119              (setq regexps (append (caddr wrap)
120                                    regexps))))))))))

```

```

122 (defun PBOOK-FORMAT-LINE (start end line-number total-lines)
123   "Format a complete line of code, by spans of constant text property.
124 Also wraps it as per 'pbook-around-code-line'. Each span is only escaped
125 at the very end to facilitate examination of the buffer."
126   (save-excursion
127     (save-restriction
128       (narrow-to-region start end)
129       (goto-char start)
130       (let* ((pbook-current-line line-number)
131              (pbook-current-total-lines total-lines)
132              (substr-beg (point-marker))
133              (substr-end (point-marker))
134              (wrap (pbook-around-code-line line-number total-lines))
135              (pbook-escaping-regexps (append (cddr wrap)
136                                               pbook-escaping-regexps)))
137         (move-marker substr-end
138                    (next-char-property-change (marker-position substr-beg)))
139         (set-marker-insertion-type substr-beg t)
140         (set-marker-insertion-type substr-end t)
141         ;; Main loop: find spans of constant text properties
142         ;; then get the latex trings to wrap around it.
143         (while (not (equal substr-beg substr-end))
144           (goto-char (marker-position substr-beg))
145           (let ((wrap (save-excursion ;; DOCUMENT ME
146                       (save-restriction
147                         (narrow-to-region (marker-position substr-beg)
148                                           (marker-position substr-end))
149                         (let ((pbook-current-text-properties
150                              (text-properties-at (marker-position substr-beg))))
151                          (pbook-properties-latex-strings
152                           (pbook-face-properties
153                            (get-char-property (marker-position substr-beg)
154                                                'face))))))))))
155             (insert (first wrap))
156             (let ((pbook-escaping-regexps (append (third wrap)
157                                                   pbook-escaping-regexps)))
158               (pbook-latex-escape (marker-position substr-beg)
159                                   (marker-position substr-end)
160                                   t))
161             (goto-char (marker-position substr-end))
162             (insert (second wrap))

164             (move-marker substr-beg (marker-position substr-end))
165             (move-marker substr-end
166                        (next-char-property-change
167                         (marker-position substr-beg))))))
168         (wrap-line (first wrap)
169                   (second wrap))))))

```

## 9 Prologue and file variables

```
1 (provide 'pbook)
```

We use Emacs's magic `file variables` to make sure pbook is formatted how it should be:

```
1 ;; Local Variables:  
2 ;; pbook-author: "Luke Gorrie, with modifications by Paul Khuong"  
3 ;; pbook-use-toc: t  
4 ;; pbook-style: article  
5 ;; pbook-monochrome: t  
6 ;; End:
```

## Index

- 0.081310, 9
- 0.168740, 9
- 0.331260, 9
- 0.34413, 9
- 0.418690, 9
- 0.71414, 9
- 1, 15
- &optional, 12
- <=, 5
- 0.0, 9
- 0.01, 7
- 0.114, 9
- 0.25, 8
- 0.299, 9
- 0.500000, 9
- 0.587, 9
- 1+, 5
- 1-, 5, 14
- 1.0, 9
- 1.40200, 9
- 1.77200, 9
- 65535.0, 8
  
- and, 7, 8, 14–17
- append, 10, 12, 16–18
- apply, 8, 12, 17
- article, 4, 14
- assoc, 16
  
- beginning, 10
- beginning-of-line, 11, 15, 16
- bolp, 15
- book, 14
- boundp, 3, 8
- buffer, 10
- buffer-name, 11
- buffer-string, 8, 10, 12
  
- car, 12
- caddr, 17, 18
- cdr, 12, 16
- char-before, 15
- char-syntax, 15
- color, 7
- color-values, 8
- colors, 7
  
- commands, 14
- components, 7
- concat, 5, 7, 12, 17
- cons, 8, 12, 14, 16
- copy-list, 16
- count-lines, 15
- cur-line, 16
- current-buffer, 10
  
- dark, 8
- dark-bg-p, 8
- default, 6, 8, 17
- defn, 16
- delete-char, 15
- delete-horizontal-space, 14
- delete-region, 11
- depth, 13
- display-buffer, 10
  
- emacs-pid, 3
- end, 12, 14–16, 18
- ending, 10
- entry, 12
- eobp, 13, 14
- eq, 7, 8, 16, 17
- equal, 15, 18
- erase-buffer, 10
- every, 7
- extension, 3
  
- face, 7, 8, 16–18
- face-attribute, 16
- face-background, 8
- face-background-mode, 8
- face-bold-p, 7
- face-foreground, 8
- face-italic-p, 7
- faces, 16, 17
- first, 7, 8, 12, 16–18
- font-lock-builtin-face, 6
- font-lock-comment-face, 6
- font-lock-constant-face, 6
- font-lock-default-fontify-buffer, 10
- font-lock-doc-face, 6
- font-lock-fontified, 10
- font-lock-function-name-face, 6

font-lock-keyword-face, 6  
font-lock-string-face, 6  
font-lock-syntactic-keywords, 10  
font-lock-variable-name-face, 6  
font-lock-warning-face, 6  
format, 3, 8, 11, 13, 15  
forward-line, 13, 14  
foundp, 17  
funcall, 16, 17  
function, 6, 8  
  
get-buffer-create, 10  
get-char-property, 18  
goto-char, 10–16, 18  
  
idxfile, 3  
incf, 16  
index, 8  
inherits, 16  
insert, 10, 12, 13, 15, 18  
interactive, 2, 3, 10  
  
keep, 5  
  
lambda, 17  
length, 5, 13, 14  
line-beginning-position, 13, 16  
line-end-position, 13, 16  
line-number, 5, 18  
list, 5, 7–9, 12, 16, 17  
list\*, 8  
listp, 16, 17  
looking-at, 5, 7, 12–15  
  
make-local-variable, 10  
mapcan, 12, 16  
mapcar, 7–9  
marker-position, 18  
match-beginning, 12  
match-string-no-properties, 13  
max, 5  
member, 7, 8  
min, 14  
move-marker, 18  
  
name, 3, 17  
narrow-to-region, 12, 14–16, 18  
next-char-property-change, 18  
nil, 4–7, 10–12, 14–17  
not, 7, 8, 13–18  
  
nth, 14  
null, 7  
num, 5  
num-lines, 16  
number-to-string, 5, 7  
  
occur, 2  
or, 7, 8, 12, 14, 15  
  
pbook-around-code-line, **5**, 18  
pbook-article-sectioning-commands, 13, 14  
pbook-author, 4, 11  
pbook-book-sectioning-commands, 14, 14  
pbook-buffer, **2**, 3  
pbook-buffer-view-pdf, **3**  
pbook-code-epilogue, 4, 15  
pbook-code-prologue, 4, 15  
pbook-commentary-regexp, 3, 10, 13–15  
pbook-convert-tabs-to-spaces, 15, **15**  
pbook-current-line, 4, 18  
pbook-current-text-properties, 6, 8, 18  
pbook-current-total-lines, 4, 18  
pbook-dark-colors, 5, 8  
pbook-do-code, 13, **15**  
pbook-do-commentary, 13, **14**  
pbook-do-heading, 13, **13**  
pbook-escaping-regexps, 6, 8, 12, 18  
pbook-face-color, 7, **8**  
pbook-face-latex-properties, 5, 17  
pbook-face-override, 6, 16  
pbook-face-properties, **17**, 18  
pbook-font-lock-override, 5, 10  
pbook-format-buffer, 10, **13**  
pbook-format-code, 15, **16**  
pbook-format-line, 16, **18**  
pbook-get-inherits, 16, **16**, 17  
pbook-goto-end-of-code, 15, **15**  
pbook-heading-level-subexp, 4, 10, 13  
pbook-heading-regexp, 2, 4, 10, 11, 13, 15  
pbook-identifier, 5, 8  
pbook-include-toc, 4, 10, 11  
pbook-inherit-buffer-locals, 10, **10**  
pbook-latex-escape, 12, **12**, 13, 14, 18  
pbook-latex-escape-string, 8, 11, **12**  
pbook-monochrome, 5, 7, 10  
pbook-nth-sectioning-command, 13, **14**  
pbook-preprocess, 10, **11**  
pbook-pretty-commentary, 14, **14**  
pbook-pretty-doublequotes, 14, **15**

pbook-pretty-tt, 14, **14**  
 pbook-process-buffer, 2, **10**  
 pbook-properties, 7, 16, 17  
 pbook-properties-latex-strings, **17**, 18  
 pbook-rgb-yuv, 8, **9**  
 pbook-show-structure, **2**  
 pbook-style, 4, 10, 11, 14  
 pbook-tex-beginning, 10, **11**  
 pbook-tex-ending, 10, **11**  
 pbook-tmpfile, 3, **3**  
 pbook-translate-face-properties, **16**, 17  
 pbook-yuv-rgb, 8, **9**  
 pdffile, 3  
 plist, 17  
 plist-get, 8, 16, 17  
 plist-member, 17  
 plist-put, 16  
 point, 11, 14, 15  
 point-marker, 18  
 point-max, 3, 10, 12, 15  
 point-min, 3, 10–12, 14–16  
 predicate, 16  
 prefix, 13  
 prepend, 17  
 prop, 7, 17  
 prop-name, 16  
 property, 17  
 props, 16, 17  
 push, 17  
  
 re-search-forward, 11, 12, 14, 15  
 regexps, 17  
 repeat, 5  
 replace, 12  
 replace-match, 11–15  
 rest, 12  
 reverse, 17  
 rgb-specs, 8  
  
 scan-regexp, 12  
 second, 7, 8, 16–18  
 set, 10  
 set-marker-insertion-type, 18  
 setq, 10, 16, 17  
 shell-command, 3  
 space, 12  
 start, 12, 14–16, 18  
 str, 5  
 string, 5, 12  
  
 substr-beg, 18  
 substr-end, 18  
 suffix, 13  
 symbol-name, 11  
 symbol-value, 10  
 syntax-table, 8  
  
 temp-directory, 3  
 temporary-file-directory, 3  
 test, 12  
 texfile, 3  
 text, 10  
 text-properties-at, 18  
 third, 7, 8, 17, 18  
 total-lines, 5, 18  
 transformer, 17  
  
 unspecified, 16  
 untabify, 15  
 use, 8  
 user-full-name, 4  
  
 value, 16  
 variable, 6, 8  
 variables, 10  
  
 word, 8  
 wrap, 17, 18  
 wrap-line, 13, **13**, 18  
 write-region, 3  
  
 yuv-specs, 8