

# Contents

<b>1</b>	<b>Startup</b>	<b>2</b>
1.1	talcum.el . . . . .	2
<b>2</b>	<b>If things go wrong...</b>	<b>4</b>
2.1	talcum-bugreport.el . . . . .	4
<b>3</b>	<b>In command of your commands</b>	<b>6</b>
3.1	talcum-newcmd.el . . . . .	6
<b>4</b>	<b>Better completion for a better tomorrow</b>	<b>11</b>
4.1	talcum-tabdance.el . . . . .	11
<b>5</b>	<b>Ain't gotta compile them all!</b>	<b>16</b>
5.1	talcum-pikeman.el . . . . .	16
<b>6</b>	<b>Où sont les dvi d'antan?</b>	<b>23</b>
6.1	talcum-prod.el . . . . .	23
<b>7</b>	<b>Rendering</b>	<b>26</b>
7.1	talcum-render.el . . . . .	26
7.2	talcum-render-ui.el . . . . .	34
7.3	talcum-render-rules.el . . . . .	40
<b>8</b>	<b>It's all been done before.</b>	<b>43</b>
8.1	trex-7bit.el . . . . .	43
8.2	trex-lists.el . . . . .	43
8.3	trex-markup.el . . . . .	46
8.4	trex-math.el . . . . .	46
8.5	trex-sectioning.el . . . . .	48
8.6	trex-unicode.el . . . . .	48
8.7	trex-verbatim.el . . . . .	53
8.8	trex-xref.el . . . . .	53
<b>9</b>	<b>Mixed implementation cruft</b>	<b>54</b>
9.1	talcum-compatible.el . . . . .	54
9.2	talcum-tools.el . . . . .	55
9.3	talcum-custom.el . . . . .	61
9.4	talcum-menu.el . . . . .	66

# 1 Startup

## 1.1 talcum.el

```
1 (defconst talcum-version "0.5.0")
2
3 ;; Path setup.
4 ;;
5 ;; Technically, all that we want is all the .el files where we can
6 ;; find them. This used to be done via talcum-path; it happens
7 ;; automatically if Talcum is installed to the magic site-lisp
8 ;; directories; and if at least this file is in the load-path already,
9 ;; we will just assume that
10 ;; 1. main Talcum files are in the same directory
11 ;; 2. TREXes live in the subdir /trex/
12 ;; since they fall out the tar.gz that way.
13
14 (let* ((thisfile (locate-library "talcum")))
15   (cond
16     ((and thisfile (not (boundp 'talcum-path)))
17      ;; We are in the load-path already, and we need not honour
18      ;; talcum-path.
19      (setq talcum-path (file-name-directory thisfile))
20      ;; If we are in the magic trees, we are done.
21      (if (not (string-match "/site-lisp/" thisfile))
22          ;; Otherwise, add the trex subdir.
23          (add-to-list 'load-path (concat talcum-path "/trex/")))))
24     (t
25      ;; OK, we were called with absolute path the old-fashioned way.
26      ;; Unfortunately, there seems to be no way of getting our position
27      ;; while we're being loaded.
28      (when (not (boundp 'talcum-path))
29        ;; Yes well, what am I supposed to do?
30        (setq talcum-path "/home/ulmi/Projekte/Emacs/Talcum/talcum/src/")
31        (message "Warning: Could not figure out talcum-path, using %s" talcum-path))
32      ;; Anyway, now we're ready.
33      (add-to-list 'load-path talcum-path)
34      (add-to-list 'load-path (concat talcum-path "/trex/")))))
35
36 ;; Nothing works without these three.
37 (require 'talcum-tools)
38 (require 'talcum-compatible)
39 (require 'talcum-custom)
40
41 (talcum-namespace
42  (~~sourcefile-versionstamp "$RCSfile: talcum.el,v $" "$Revision: 2.11 $"))
43
44 (autoload 'talcum-add-render-feature
45  "talcum-render" 'interactive)
46
47 Customize-Setup.
48
49 (defgroup talcum
50   nil ;; members
```

```

49 "A mixed bag of things that make LaTeX easier."
50 :group 'tex)
51
52 (let ((unless-forbidden
53       (lambda (flag sym)
54         (if (and (boundp flag) (null (symbol-value flag)))
55             nil
56             (list sym)))))
57 ;; We used to have talcum-use-foo-flags here. That's unfortunate,
58 ;; optimally, we don't want to change this file when a new sublibrary
59 ;; is added. Now, we have a list of features that will be required.
60 (defcustom talcum-desired-features
61   ;; append hack here: if those were customized to nil, obey that.
62   (append
63    (funcall unless-forbidden 'talcum-use-render-flag 'talcum-render)
64    (funcall unless-forbidden 'talcum-use-prod-flag 'talcum-prod)
65    (funcall unless-forbidden 'talcum-use-newcmd-flag 'talcum-newcmd)
66    '(talcum-pikeman talcum-tabdance))
67   "Features that should be loaded when Talcum is started"
68   :group 'talcum
69   :type '(repeat symbol) ;; fix this for something clickable
70   ))
71
72 ;; Two hooks provided for the proper work-doing code. We could do
73 ;; without the -enter hook, but it's here for symmetry. Since we
74 ;; dabble in idle-timers etc. pp., we want to provide a way to disable
75 ;; them.
76 (defvar talcum-enter-mode-hook nil)
77 (defvar talcum-leave-mode-hook nil)
78
79 Settings of keys.
80
81 (define-prefix-command 'talcum-keymap)
82
83 ;;
84 ;; Main entry point for all things talcum
85 ;;
86 (defconst talcum-lighter "Talcum")
87
88 (defcustom talcum-mode-chord (vector ?\C-c ?#)
89   "The key combination prefix for all Talcum commands."
90   :group 'talcum
91   :type 'sexp)
92
93 ###autoload
94
95 (define-minor-mode talcum-mode
96   "A minor mode to enhance both LaTeX- and latex-mode."
97   nil ; initially off
98   talcum-lighter ; mode indicator
99   (list ; keymap
100    (cons talcum-mode-chord 'talcum-keymap))
101   ;; init body
102
103   ;; Hmmm... I think this should work. The docs are very unclear on

```

```

100 ;; whether the autoload cookie will load the whole file. Let's make
101 ;; sure, because without namespaces, we shan't get far.
102 (require 'talcum)
103 (require 'talcum-menu)
104
105 (when talcum-mode ;; activating!
106   (talcum-namespace
107     (cond
108       ((talcum-latex-p) (~debug 1 "Classic latex mode, ok.))
109       ((talcum-auctex-p) (~debug 1 "AucTeX LaTeX mode, ok.))
110       (t (~debug 9 (concat "Unsupported major mode \"%S\". "
111                           "Things might go wrong.")
112            major-mode)))
113
114   ;; more activation stuff goes here
115   (dolist (feat talcum-desired-features)
116     (~debug 1 "Requiring %s" feat)
117     (require feat))
118   (require 'talcum-bugreport)
119   (run-hooks 'talcum-enter-mode-hook)
120   (talcum-menu-regenerate)
121   t))
122 (unless talcum-mode ;; deactivating!
123   (run-hooks 'talcum-leave-mode-hook)
124   t)
125 )
126
127 (provide 'talcum)

```

talcum.el ends here.

## 2 If things go wrong...

### 2.1 talcum-bugreport.el

Bug reporting.

It's difficult for users to know what's important and what's not. Also, I tend to give out an erroneous email address.

```

130
131 (require 'talcum-tools)
132 (require 'talcum-menu)
133
134 (talcum-namespace
135   (~sourcefile-versionstamp "$RCSFile:" "$Revision:"))
136
137 (defun talcum-report-bug (subj)
138   (interactive "sOne-line description: ")
139   (compose-mail "ulmi@users.sarovar.org"
140     (concat "[Talcum "
141             talcum-version
142             " Bug] ")

```

```

143         subj) nil t)
144     (insert "
145
146 Please describe your problem here. ulmi speaks English, German and
147 very little French. In many cases (e.g. display bugs), it is helpful
148 if a small sample file is included on which the bug is reproducible.
149
150 "
151
152 ;; Please read and acknowledge the following legalese:
153
154 ;; The bug report includes information about your system (most
155 ;; importantly, operating system and Emacs version, and whether you use
156 ;; AucTeX). This information is included in plain text, at the end of
157 ;; this report.
158
159 ;; You give permission for this mail to be published and incorporated,
160 ;; whole or in parts, free of charge and obligation to the maintainer, in
161 ;; future releases of Talcum (e.g. if you propose a fix) and/or on the
162 ;; Talcum homepage (probably in the Bugs and Patches sections). The
163 ;; maintainer will give credit where it's due, unless he forgets to (in
164 ;; which case, please submit a bug about forgetting to credit).
165
166 "\nThank you for your help!\n    Ulrich"
167
168 "\nInternal information:\n")
169 (mapc
170   #'(lambda (sym)
171       (insert
172         (concat (symbol-name sym) "'s value: "
173                 (if (boundp sym)
174                     (format "%s" (symbol-value sym))
175                     "(void)"))
176         "\n")))
177   '(
178     emacs-version
179     system-type
180     system-configuration
181     AucTeX-version
182     talcum-version
183     talcum--versions
184     talcum-path
185     talcum-desired-features
186   ))
187 t)
188
189 (defalias 'talcum-curse-at-programmer 'talcum-report-bug)
190
191 (defun talcum-bug-generate-menu ()
192   (list "--"
193         '["Report a bug" talcum-curse-at-programmer]))
194
195 (add-to-list 'talcum-menu-functions

```

```

196      #'talcum-bug-generate-menu
197      :at-end)
198
199 (provide 'talcum-bugreport)

```

## 3 In command of your commands

### 3.1 talcum-newcmd.el

\$Id: talcum-newcmd.el,v 2.15 2005/08/27 12:41:08 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, [ulmiusers.sarovar.org](mailto:ulmiusers.sarovar.org).

```

200
201 (require 'talcum-tools)
202 (require 'talcum-compatible)
203 (require 'talcum-menu)
204
205
206 (talcum-namespace
207   (~sourcefile-versionstamp "$RCSfile: talcum-newcmd.el,v $" "$Revision: 2.15 $")
208
209   (defun talcum-newcmd-generate-menu ()
210     (list "--"
211       ["Insert \\newcommand in preamble" talcum-insert-newcommand]
212       ["Insert \\newenvironment in preamble" talcum-insert-newenvironment]
213       ["Insert \\usepackage in preamble" talcum-insert-usepackage]))
214
215   (add-to-list 'talcum-menu-functions #'talcum-newcmd-generate-menu)
216
217   (defvar talcum-newfoo-before-insertion-hooks nil)
218   (defvar talcum-newfoo-after-insertion-hooks nil)
219
220   (defun ~newfoo-find-suitable-place ()
221     (let* ((talcum-definitions-file talcum-definitions-file)
222            (goodbuffer
223              ;; 2005AUG25: jumbling the order.
224              ;; Doing functionp earlier obliterates the need for
225              ;; :latexmode and :auctex flags -- auctex does not
226              ;; provide a tex-main-file.
227              (cond
228                ((stringp talcum-definitions-file)
229                 talcum-definitions-file)
230                ((functionp talcum-definitions-file)
231                 (funcall talcum-definitions-file))
232                ((and (equal talcum-definitions-file ':latexmode)
233                      (boundp 'tex-main-file)
234                      (stringp tex-main-file))
235                 tex-main-file)
236                ((and (equal talcum-definitions-file ':auctex)

```

```

237         (talcum-auctex-p))
238         (TeX-active-master t))
239         (t (current-buffer))
240         )))
241     (when (stringp goodbuffer)
242       (switch-to-buffer (find-file-noselect goodbuffer) 'secretly))
243     (if buffer-read-only
244       (vc-toggle-read-only))
245     ;;(switch-to-buffer goodbuffer 'secretly)
246     (~newfoo-find-suitable-position)))
247
248 (defun ~newfoo-find-suitable-position (&optional type)
249   ;; This is hopeless. What _is_ a good position?
250   ;;
251   ;; Optimally, we would have all usepackages first, but their setup
252   ;; commands (\hypersetup et al.) just after them, and all commands
253   ;; after that.
254   (goto-char (point-min))
255   (search-forward "\\begin{document}" nil t)
256   (forward-line 0)
257   ;; Now just before begin{document}.
258   ;; Always skip over author, title, date, etc.
259   (dolist (textinfo talcum-textinfo-commands)
260     (search-backward textinfo nil t))
261   (when (eq type :package)
262     ;; This misbehaves if no usepackage is present, but other
263     ;; commands. Oh c'mon, that's bound to be rare.
264     (when (search-backward "\\usepackage" nil t)
265       (forward-char (length "\\usepackage"))
266       (when (looking-at "[[:space:]]*\\[")
267         ;; skip package args
268         (skip-syntax-forward " ")
269         (~fwd-arg))
270       (~fwd-arg)
271       (when (looking-at "[[:space:]]*\\[")
272         ;; skip package release date
273         (skip-syntax-forward " ")
274         (~fwd-arg))
275       (forward-line 1))
276     t)
277   t)
278
279 (defmacro ~at-suitable-place (&rest cmds)
280   '(save-excursion
281     (save-window-excursion
282       (~newfoo-find-suitable-place)
283       ,@cmds)))
284
285
286 (defun talcum-excurse-to-preamble ()
287   (interactive)
288   (~at-suitable-place
289     (message "Press M-C-c when finished."))

```

```

290     (recursive-edit)
291     (save-buffer)))
292
293 (defun talcum-insert-newcommand (pfx macname)
294   "Jump to a suitable point and generate a skeleton for a LaTeX macro.
295   Where to jump is determined by ltx-newfoo-find-suitable-place.
296   With a prefix argument, assume that many parameters (0 otherwise).
297   With transient-mark-mode and active region, use that region as body."
298   (interactive "P\nsName of macro: \\\")
299
300   (let* ((argnumber
301          (cond
302            ((null pfx) 0)
303            ((integerp pfx) pfx)
304            ((listp pfx) (car pfx)) ; empty list caught above
305            ((eq pfx '-') -1))) ; whoopsie! Think of something.
306     (beg (and transient-mark-mode mark-active (region-beginning)))
307     (end (and transient-mark-mode mark-active (region-end)))
308     (body (if beg (delete-and-extract-region beg end)
309              "%\n"))))
310
311   (when beg
312     (deactivate-mark)
313     (insert (format "\\s%s"
314                   macname
315                   (if (and pfx (= 0 argnumber))
316                       "\\ "
317                       (apply 'concat
318                             (make-list argnumber "{}")))))
319     t)
320   (~at-suitable-place
321    (run-hook-with-args 'talcum-newfoo-before-insertion-hooks
322                        ':macro macname argnumber)
323    (insert (format
324            "\\newcommand\\s%s{s{%%\n\n}\n\n"
325            macname
326            (if (= 0 argnumber) ""
327                (format "[%s]" argnumber))
328            body))
329    (forward-line -3)
330    (message "Press M-C-c when finished.")
331    (recursive-edit)
332    (run-hook-with-args 'talcum-newfoo-after-insertion-hooks
333                        ':macro macname argnumber)
334    (save-buffer)
335    t)
336   t))
337
338 (defun talcum-insert-newenvironment (pfx envname)
339   "Jump to a suitable point and generate a skeleton for a LaTeX environment.
340   Where to jump is determined by ltx-newfoo-find-suitable-place. The
341   prefix, if any, specifies the number of arguments."
342   (interactive "P\nsName of environment: ")

```



```

343
344 (let* ((argnumber
345        (cond
346          ((null pfx) 0)
347          ((integerp pfx) pfx)
348          ((listp pfx) (car pfx)) ; empty list caught above
349          ((eq pfx '-') -1)))    ; whoopsie! Think of something.
350
351  (~at-suitable-place
352   (run-hook-with-args 'talcum-newfoo-before-insertion-hooks
353     ':environment envname argnumber)
354   (insert (format "\\newenvironment{%s}%s{%%\n%%%\n}{%\n%%%\n}\n\n"
355     envname
356     (if (= 0 argnumber) ""
357         (format "[%s]" argnumber))))
358   (forward-line -5)
359   (message "Press M-C-c when finished.")
360   (recursive-edit)
361   (run-hook-with-args 'talcum-newfoo-after-insertion-hooks
362     ':environment envname argnumber)
363   (save-buffer))))
364
365
366 (defun talcum-insert-usepackage (package &optional options)
367   (interactive "sPackage name: \n")
368   (unless options
369     (setq options
370       (read-from-minibuffer
371         "Options: "
372         (cdr-safe (assoc package talcum-package-option-hints)))))
373   (~at-suitable-place
374    (~newfoo-find-suitable-position :package)
375    (insert "\\usepackage"
376      (if (< 0 (length options))
377        (concat "[" options "]")
378        ""))
379    "{" package "}\n")
380    (sit-for 1.0)))
381
382 ) ;;namespace

Init and deinit functions, for completeness.

384 (defun talcum-newcmd-init ()
385   t)
386 (defun talcum-newcmd-exit ()
387   t)

Key bindings

389 (define-key talcum-keymap "c" 'talcum-insert-newcommand) ; c for command
390 (define-key talcum-keymap "e" 'talcum-insert-newenvironment) ; e for envir
391 (define-key talcum-keymap "p" 'talcum-insert-usepackage) ; p for package

```

Customizations

```

394
395 (defgroup talcum-newcmd nil "Inserting new commands" :group 'talcum)
396
397 (defcustom talcum-textinfo-commands
398   ' (; LaTeX2e classes and everywhere:
399     "\\author" "\\date" "\\ title "
400     ;; KOMA-Script:
401     "\\publishers" "\\subject" "\\titlehead"
402     "\\dedication" "\\uppertitleback" "\\lowertitleback" "\\ extratitle "
403     ;; beamer:
404     "\\ subtitle " "\\ institute " "\\subject" "\\keywords" "\\titlegraphic"
405   )
406   "Commands that are technically in the preamble, but should not be
407   seperated from begin{document}."
408   :group 'talcum-newcmd
409   :type '(repeat :tag "Macro name" string))
410
411 (defcustom talcum-use-newcmd-flag t
412   "Enable the whole newcommand subsystem."
413   :group 'talcum
414   :type 'boolean)
415
416
417 (defcustom talcum-definitions-file nil
418   "Where Talcum will put new definitions (newcommands, newenvs)."
419   :group 'talcum-newcmd
420   :tag "Talcum definitions file"
421   :type '(choice
422     (const :tag "the current buffer" nil)
423     (const :tag "LaTeX-mode's master file (deprecated)" ':latexmode)
424     (const :tag "AucTeX' master file (deprecated)" ':auctex)
425     (string :tag "filename" :value "preamble.ltx")
426     (function :tag "returned by function" :value (lambda () "preamble.tex"))))
427
428
429
430 (defcustom talcum-package-option-hints
431   ' (("inputenc" . "latin9")
432     ("fontenc" . "T1")
433     ("helvet" . "scaled"))
434   "Talcum can suggest options for packages."
435   :group 'talcum-newcmd
436   :tag "Package option suggestions"
437   :type '(repeat
438     (cons (string :tag "package name" :value "package")
439           (string :tag "option string" :value ""))))
440
441
442 (add-hook 'talcum-enter-mode-hook #'talcum-newcmd-init)
443 (add-hook 'talcum-leave-mode-hook #'talcum-newcmd-exit)
444
445 (provide 'talcum-newcmd)

```



```

478 (setq talcum-tabdance-key chord))
479
480 (defcustom talcum-tabdance-auto-prefer-history t
481   "Make a two-step tabdance out of history and completion list
482   automatically, in all applications that use completing-read"
483   :group 'talcum-tabdance
484   :type 'boolean)
485
486 (defcustom talcum-tabdance-auto-escalate t
487   "If no completion exists at the current completion level,
488   automatically try a more choiceful level."
489   :group 'talcum-tabdance
490   :type 'boolean)

```

In intermediate CVS versions, this was not documented and checked with `~value-safe`. This will probably become obsolete if the default `tex-latex-block` becomes history-aware itself, unless by then they've completely gone over the border with asking for parameters etc.

```

492 (defcustom talcum-tabdance-no-hijacking t
493   "Do not replace LaTeX-mode's tex-latex-block with a history-enabled
494   variant."
495   :group 'talcum-tabdance
496   :type 'boolean)
497
498 (talcum-namespace
499
500 (defun talcum-tabdance-generate-menu ()
501   (list "--"
502     (~~menu-flag-toggle talcum-use-tabdance-flag
503       "Improved Minibuffer")

```

Oh well, I think they're stable enough. Use the customization.

```

504 ;;      (~~menu-flag-toggle talcum-tabdance-auto-prefer-history
505 ;;      "Automatic History")
506 ;;      (~~menu-flag-toggle talcum-tabdance-auto-escalate
507 ;;      "Automatic Escalation")
508 ;;    ))
509
510 (add-to-list 'talcum-menu-functions #'talcum-tabdance-generate-menu)
511
512
513 (defun talcum-tabdance-escalate ()
514   "Go to the next more choicy level of completion, if there is one.
515   Return a non-nil value if that is a real increase."
516   (interactive)
517   (when (boundp 'tabdance-level)
518     (unless (= tabdance-level (~~pseudolength tabdance-choices))
519       (if (interactive-p)
520         (message "completion level %s" (1+ tabdance-level)))
521       (setq tabdance-level (1+ tabdance-level))))))
522
523 (defun talcum-tabdance-descalate ()
524   "Go to the previous, more restricted level of completion if there is one.
525   Return a non-nil value if that is a real decrease"

```

```

526 (interactive)
527 (when (boundp 'tabdance-level)
528   (unless (= 1 tabdance-level)
529     (if (interactive-p)
530       (message "completion level %s" tabdance-level))
531     (setq tabdance-level (1- tabdance-level)))))

```

This is the one the user will want to call. Question: should the directions be inverted? With automatic escalation, maybe going back down will be more useful by default.

```

533 (defun talcum-tabdance-scalate (direction)
534   "Without prefix argument, go to a more choiceful completion level,
535   with prefix argument, go back one level."
536   (interactive "P")

```

We call-interactively because we want the message.

```

537   (call-interactively
538     (if direction
539       #'talcum-tabdance-desalate
540       #'talcum-tabdance-escalate)))

```

Remove doubles from a list. This is likely to be phased out at some point, because it looks like GNU Emacs 22+ will have an option to remove doubles on its own. One way or another, this is not critical, but a completion window that lists equation five times looks... strange.

```

542 (defun ~~sparsed (xs)
543   (let (ys)
544     (dolist (x xs ys)
545       (add-to-list 'ys x))))

```

Somebody set us up the bomb!

```

547 (defun talcum-tabdance-init ()

```

Be extra paranoid about the key. If auto-escalation is off and there is no escalation key, completion is effectively only selecting from the history.

```

548   (if (not talcum-tabdance-key)
549     (message "Tabdance not started: %s"
550       "you have not defined an escalation key.")
551     (define-key minibuffer-local-completion-map
552       talcum-tabdance-key (quote talcum-tabdance-scalate))
553
554     (when (and (talcum-latex-p)
555       (not talcum-tabdance-no-hijacking))
556       (~~tabdance-hijack)
557       t))
558

```

```

559 (defun ~~tabdance-hijack ()
560   ;; Vaguely similar to the function from classic LaTeX mode as
561   ;; of GNU Emacs 21.3. No "Options" Question -- who uses that
562   ;; anyway? Display of default choice is standard now, offers
563   ;; history, accommodates both latex-mode 21.3 and latex-mode
564   ;; 22.
565   (define-skeleton tex-latex-block
566     "Create a matching pair of lines \begin{NAME} and \end{NAME} at point.

```

```

567 Puts point on a blank line between them. Offers history."
568 (let ((blockname
569       (completing-read
570         (format "LaTeX environment: (default %s)" latex-block-default)
571         (mapcar 'list (~~sparsed
572                     (append (or
573                               (~~value-safe standard-latex-block-names) ;; --21
574                               (~~value-safe latex-standard-block-names));; 22--
575                             latex-block-names)))
576         (not ':filter-by-predicate)
577         (not ':require-match)
578         (not ':filled-out-already)
579         'latex-block-names ; history
580         latex-block-default ; default
581         )))
582       (setq latex-block-default blockname)
583       blockname)
584 ;; Don't start a new line for environment?
585 ;; \n
586 ;; Without extra >, we misindent for document.
587 "\\begin{" str "}" >
588 \n _ \n
589 "\\end{" str "}" > \n)

```

In latex-mode 22-, C-c C-o points to latex-insert-block.

```

591 (defalias 'latex-insert-block 'tex-latex-block))
592
593
594 (defun talcum-tabdance (str pred allp)
595   (let* ((realchoice (~~nth-filtered (1- tabdance-level)
596                                       tabdance-choices
597                                       'identity)))
598     (cond
599       ((equal t allp)
600        (while (and talcum-tabdance-auto-escalate
601                    (null (all-completions str realchoice pred))
602                    (talcum-tabdance-escalate)
603                    (setq realchoice (~~nth-filtered (1- tabdance-level)
604                                                      tabdance-choices
605                                                      'identity))))
604          (all-completions str realchoice pred))
605       ((null allp)
606        (while (and talcum-tabdance-auto-escalate
607                    (null (try-completion str realchoice pred))
608                    (talcum-tabdance-escalate)
609                    (setq realchoice (~~nth-filtered (1- tabdance-level)
610                                                      tabdance-choices
611                                                      'identity))))
610          (try-completion str realchoice pred))
611       ((equal 'lambda allp)

```

```

619      (not (null (member str
620                    (all-completions str realchoice pred))))))
621    )))
622
623 (defun ~nth-filtered (n xs pred)
624   (nth n (~filter pred xs)))
625
626 (defun ~pseudolength (xs)
627   (length (~filter 'identity xs)))
628
629 (defadvice completing-read
630   (around tabdance (prompt table
631                     &optional predicate reqmatch initial hist def inherit)
632     activate)
633   (let ((tabdance-choices '())
634         (table table)
635         (tabdance-level 1) ; Ugh. Think of something better.
636         )
637     (cond
638      ;; Safeguard: we're not correctly set up, go back to old behaviour.
639      ((or (not talcum-use-tabdance-flag)
640           (not talcum-tabdance-key))
641       t)
642
643      ;; Explicit :tabdance request. Note that in this case, we are not
644      ;; trying to be clever about the history. This is not needed,
645      ;; because the caller already knows about tabdance, so it's
646      ;; his/her responsibility to include history vars at a suitable
647      ;; place.
648      ((and (consp table)
649            (equal (car table) ':tabdance))
650       (setq tabdance-choices (cdr table)
651             table 'talcum-tabdance))
652
653      ;; Prefer the history.
654      ((and talcum-tabdance-auto-prefer-history hist)
655       (setq tabdance-choices (list
656                               ;; We must convert the history from list
657                               ;; to alist.
658                               (mapcar 'list
659                                       ;; The history may be a (symbol .
660                                       ;; offset) cons as well, but we
661                                       ;; always want the symbol's value,
662                                       ;; which is a list.
663                                       (symbol-value
664                                         (if (consp hist)
665                                               (car hist)
666                                               hist)))
667                               table)
668             table 'talcum-tabdance))
669
670      ;; Default case: do nothing fancy.

```

```

672     (t
673     t))
674   ad-do-it))
675
676   (add-hook 'talcum-mode-hook (lambda () (if talcum-use-tabdance-flag
677                                           (talcum-tabdance-init))))
678   )
679   (provide 'talcum-tabdance)
680   ;; talcum-tabdance.el ends here.

```

## 5 Ain't gotta compile them all!

### 5.1 talcum-pikeman.el

\$Id: talcum-pikeman.el,v 1.16 2005/08/26 18:36:57 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, [ulmiusers.sarovar.org](mailto:ulmiusers.sarovar.org).

```

681
682   (require 'talcum-tools)
683   (require 'talcum-compat)
684
685   (talcum-namespace
686     (~~sourcefile-versionstamp "$RCSfile: talcum-pikeman.el,v $" "$Revision: 1.16 $"))
687
688   (defgroup talcum-pikeman nil "The compiling options menu" :group 'talcum)
689
690   (talcum-namespace
691
692     (defvar talcum-pikeman-compile-options-functions nil
693       "A list of functions that will be called prior to calling tex-file.
694 Each function will be called with one argument, the name of the main
695 file, and should return a string. All these strings are concatenated,
696 seperated by spaces, and passed as parameters to TeX.")
697
698     (defvar talcum-pikeman-compile-filestart-functions nil
699       "A list of functions that will be called prior to calling tex-file.
700 Each function will be called with one argument, the name of the main
701 file, and should return a string. All these strings are concatenated,
702 and passed to TeX as the first input line, suitably quoted.")
703
704     (defvar talcum-pikeman-menu-creator-functions nil
705       "A list of functions that are called with no parameters to create
706 the Pikeman menu. They should return a list of menu items, since all
707 the results are appended.")
708
709     (defun talcum-pikeman-generate-menu ()
710       (list "--"
711         (~~pikeman-generate-menu)))
712

```



```

713 (add-to-list 'talcum-menu-functions #'talcum-pikeman-generate-menu)
714
715 (defun ~~pikeman-generate-menu ()
716   (list "Pikeman"
717     :filter (lambda (_)
718               (apply 'append
719                     (mapcar
720                      (lambda (mcf) (funcall mcf))
721                      talcum-pikeman-menu-creator-functions))))))
722
723 (defun ~~pikeman-compiler-options (texfile)
724   "Assemble the command-line options for the tex call (by calling
725   talcum-pikeman-compile-options-functions). TEXFILE is the name of the
726   main file, with extension and possibly path."
727   (let ( (options ; we do a little dance to exclude nil entries
728           (mapconcat #'identity ; just to splice in the spaces
729             (~filter #'identity ; to remove the nils
730               (mapcar (lambda (optfun) (funcall optfun texfile))
731                       talcum-pikeman-compile-options-functions))
732             " "))
733     options))
734
735 (defun ~~pikeman-first-line (texfile)
736   "Assemble the first line for the tex call (by calling
737   talcum-pikeman-compile-filestart-functions). TEXFILE is the name of
738   the main file, with extension and possibly path."
739   (let ((firstline (mapconcat (lambda (filifun) (funcall filifun texfile))
740                               talcum-pikeman-compile-filestart-functions "")))
741     firstline))
742
743 (defun ~~pikeman-prepare-compiler-string ()
744   "Set tex-command to include compiler options and the first line."
745   (let* ((texfile (tex-main-file))
746          (options (~~pikeman-compiler-options texfile))
747          (firstline (~~pikeman-first-line texfile)))
748     (setq tex-command
749           (concat latex-run-command " "
750                   ;;--jobname * " ;; * not quoted, replaced by tex-file
751                   (~value-safe tex-start-options-string) ;; removed in 21--
752                   (~value-safe tex-start-options) ;; 22+
753                   " "
754                   options " "
755                   " ,"
756                   ;; somewhat ugly bugfix: what do we do about *
757                   ;; generated by tex-start-commands, tpc-filestart
758                   ;; et al.? If allowed, the \input below will fail,
759                   ;; since at most one * expansion is done.
760                   ;; * in options can still give replacement.
761                   ;; We assume existence of \def\@gobble#1{} (i.e. LaTeX 2e)
762                   "\csname @gobble\endcsname{*}"
763                   firstline
764                   (~value-safe tex-start-commands) ;; 22+
765                   "\input{"))

```

```

766         texfile
767         "}")
768     ))
769
770 ;; Only needed for classic LaTeX mode, but who cares?
771 (defadvice tex-file (before talcum-pikeman activate)
772   (~~pikeman-prepare-compiler-string))
773
774 ;; Only for AucTeX. Can't check for AucTeX version since that is
775 ;; sometimes a date, sometimes not.
776 (when (boundp 'TeX-expand-list)
777   (add-to-list 'TeX-expand-list
778     (list "%(Talcum-options)"
779       (lambda () (~~pikeman-compiler-options (TeX-active-master t))))))
780   (add-to-list 'TeX-expand-list
781     (list "%(Talcum-line1)"
782       (lambda () (~~pikeman-first-line (TeX-active-master t))))))
783
784 ); namespace

```

That should be sufficient for the framework.

Now, the canonical examples (to be out-sourced à la TREX)

Ex. 1: Includes.

```

787 (talcum-namespace
788
789 ;; to be made buffer-local? we want one per main file...
790 (defvar ~~pikeman-selected-chapters nil)
791 (defvar ~~pikeman-all-chapters nil)
792
793
794 (defun ~~pikeman-include-menu ()
795
796   (setq ~~pikeman-all-chapters
797     (~~pikeman-find-includes
798       (or (and (boundp 'tex-main-file)
799         tex-main-file)
800         (and (talcum-auctex-p)
801           (TeX-active-master t))
802         nil)))
803   (if ~~pikeman-all-chapters
804     ;; if nothing to select, don't offer menu.
805     ;; Caveat: we look at version on disk.
806     (identity ;; was list for submenu, but that's a lot of mousing.
807       (append '("___" "___"
808         ["Included chapters:" nil]
809         )
810         (mapcar (quote (~~pikeman-menu-item) ~~pikeman-all-chapters)
811           '("___"
812             ["Include none"
813               (talcum-namespace
814                 (setq ~~pikeman-selected-chapters nil))])

```

```

815
816     ["Include all"
817      (talcum-namespace
818       (setq ~~pikeman-selected-chapters ~~pikeman-all-chapters))]
819
820     ["Rescan file"
821      (talcum-namespace
822       (setq ~~pikeman-menu (~~pikeman-make-menu))))]
823   ))))
824
825 (defun ~~pikeman-menu-item (chapter)
826   '[,chapter
827    (talcum-namespace
828     (setq ~~pikeman-selected-chapters
829      (if (member ,chapter ~~pikeman-selected-chapters)
830          (remove ,chapter ~~pikeman-selected-chapters)
831          (cons ,chapter ~~pikeman-selected-chapters))))
832    :key-sequence nil          ;sorry, how would that work?
833    :style toggle
834    :selected (member ,chapter (quote ,(talcum-namespace ~~pikeman-selected-chapters)))
835    ])
836
837 (defun ~~pikeman-find-includes (&optional buffer-or-file)
838   (interactive "bMain buffer")
839   (let ((buffer (or (and (bufferp buffer-or-file) buffer-or-file)
840                     (and (stringp buffer-or-file)
841                          (let* ((bu (generate-new-buffer "Talcum temp master"))
842                                (save-excursion
843                                 (set-buffer bu)
844                                 (insert-file-contents buffer-or-file nil nil nil t)
845                                 bu)))
846                          (current-buffer))))
847     (rst nil)
848     (inhibit-point-motion-hooks t)
849     (inhibit-redisplay t))
850   (save-excursion
851     (set-buffer buffer)
852     (goto-char (point-min))
853     (while (search-forward-regexp
854             "^[[:space:]]*\\\\\\\\ include {\\([\\^]+\\)} "
855             (point-max) t)
856       (push (match-string-no-properties 1) rst))
857     (setq rst (nreverse rst))
858     ;;(message "%s" rst)
859     (if (stringp buffer-or-file)
860         (kill-buffer buffer))
861     rst)))
862
863 (defun ~~pikeman-make-includeonly (file)
864   ;; if we use all chapters, give no includeonly at all.
865   (unless (= (length ~~pikeman-all-chapters)
866              (length ~~pikeman-selected-chapters))
867     (concat "\\includeonly{"

```

```

868         (mapconcat 'identity
869                     ~~pikeman-selected-chapters
870                     ",")
871     "}")
872
873
874 (add-to-list 'talcum-pikeman-compile-filestart-functions
875             (quote ~~pikeman-make-includeonly))
876 (add-to-list 'talcum-pikeman-menu-creator-functions
877             (quote ~~pikeman-include-menu))
878
879 ) ; namespace
880
881 ;; Ex. 2: setting the interaction mode
882 (talcum-namespace
883
884 (defvar ~~pikeman-interaction-mode nil)
885
886 (defun ~~pikeman-interaction-menu ()
887   (list (cons "Interaction mode"
888              (~~menu-choose-one ~~pikeman-interaction-mode
889                                '(("(default)" nil)
890                                  ("batch" "batchmode")
891                                  ("non stop" "nonstopmode")
892                                  ("scroll" "scrollmode")
893                                  ("errorstop" "errorstopmode"))))))
894
895 (defun ~~pikeman-interaction-option (file)
896   (if ~~pikeman-interaction-mode
897       (concat "--interaction " ~~pikeman-interaction-mode)))
898
899 (add-to-list 'talcum-pikeman-compile-options-functions
900             (quote ~~pikeman-interaction-option))
901 (add-to-list 'talcum-pikeman-menu-creator-functions
902             (quote ~~pikeman-interaction-menu))
903 ); namespace
904
905 ;; Ex. 3: setting the format
906 (talcum-namespace
907
908 (defcustom talcum-pikeman-formats
909   '(("(default)" nil)
910     ("pdfLaTeX" "pdflatex")
911     ("LaTeX" "latex")
912     ("Beamer" "beamer")
913     ("Diplom" "diplom"))
914   "Formats to offer in the Output Format menu. First element is the
915   menu name, second is the fmt file name."
916   :group 'talcum-pikeman
917   :type '(repeat
918          (list (string :tag "Menu name")
919                (choice :tag "Format"
920                        (string :tag "File name"))

```

```

921         (const :tag "(none)" nil))))))
922
923 (defvar ~~pikeman-format nil)
924
925 (defun ~~pikeman-format-menu ()
926   (list (cons "Output format"
927     ;(cons "——"
928       (~menu-choose-one ~~pikeman-format
929         talcum-pikeman-formats))))))
930
931 (defun ~~pikeman-format-option (file)
932   (if ~~pikeman-format
933     (concat "&" ~~pikeman-format)))
934
935 (add-to-list 'talcum-pikeman-compile-options-functions
936   (quote ~~pikeman-format-option))
937 (add-to-list 'talcum-pikeman-menu-creator-functions
938   (quote ~~pikeman-format-menu))
939 );namespace
940
941 ;; Ex. 4: \ listfiles .
942 (talcum-namespace
943   (defvar ~~pikeman-listfiles nil)
944   (defun ~~pikeman-listfiles-menu ()
945     (list ;"——"
946       '["\ listfiles "
947         (talcum-namespace (~negate ~~pikeman-listfiles))
948         :style toggle
949         :selected (talcum-namespace ~~pikeman-listfiles)]))
950   (defun ~~pikeman-make-listfiles (file)
951     (if ~~pikeman-listfiles "\\listfiles"))
952
953   (add-to-list 'talcum-pikeman-menu-creator-functions
954     (quote ~~pikeman-listfiles-menu))
955   (add-to-list 'talcum-pikeman-compile-filestart-functions
956     (quote ~~pikeman-make-listfiles))
957 ); namespace
958
959 ;; Ex. 5: draft or final
960 (talcum-namespace
961   (defvar ~~pikeman-draftmode nil)
962   (defun ~~pikeman-draftmode-menu ()
963     (list (cons "Draft/Final"
964       (~menu-choose-one ~~pikeman-draftmode
965         '(("default)" nil)
966         ("draft" "draft")
967         ("final" "final"))))))))
968 (defun ~~pikeman-make-draftmode (file)
969   (when ~~pikeman-draftmode
970     (save-excursion
971       (let ((bu (generate-new-buffer "Talcum temp master")))
972         (set-buffer bu)
973         (insert-file-contents file nil nil nil t)

```

```

974 (goto-char (point-min))
975 (if (search-forward-regexp "\\documentclass\\(\\[[^]]*\\]\\)?{\\(\\[[^]]+\\)}" nil t)
976 ;; XXX: this fails for {...} within class options. Does anybody do that?
977 (format "\\PassOptionsToClass{%s}{%s}"
978 ~~~pikeman-draftmode
979 (or (match-string 2) ;; in case of options
980 (match-string 1)) ;; in case of no options
981 ))))
982
983 (add-to-list 'talcum-pikeman-menu-creator-functions
984 (quote ~~~pikeman-draftmode-menu))
985 (add-to-list 'talcum-pikeman-compile-filestart-functions
986 (quote ~~~pikeman-make-draftmode))
987 );namespace
988
989 ;; Ex. 6: force non-embedding of base fonts into pdf.
990 (talcum-namespace
991 (defvar ~~~pikeman-nonembed-fonts nil)
992 (defun ~~~pikeman-nonembed-menu ()
993 (list ;"--"
994 '["Don't embed base 14 fonts (PDF only)"
995 (talcum-namespace (~~negate ~~~pikeman-nonembed-fonts))
996 :style toggle
997 :selected (talcum-namespace ~~~pikeman-nonembed-fonts)]))
998 (defun ~~~pikeman-make-nonembed (file)
999 ;; Yeah, I know, this is not going to work for anybody else.
1000 (if ~~~pikeman-nonembed-fonts "\\pdfmapfile{pdfTeX_ndl14}"))
1001
1002 (add-to-list 'talcum-pikeman-menu-creator-functions
1003 (quote ~~~pikeman-nonembed-menu))
1004 (add-to-list 'talcum-pikeman-compile-filestart-functions
1005 (quote ~~~pikeman-make-nonembed))
1006 );namespace
1007
1008 ;; Ex. 7: source specials.
1009 (talcum-namespace
1010 (defvar ~~~pikeman-srcspc nil)
1011 (defun ~~~pikeman-srcspc-menu ()
1012 (list ;"--"
1013 '["Source specials"
1014 (talcum-namespace (~~negate ~~~pikeman-srcspc))
1015 :style toggle
1016 :selected (talcum-namespace ~~~pikeman-srcspc)]))
1017 (defun ~~~pikeman-srcspc-option (file)
1018 (if ~~~pikeman-srcspc
1019 "-src-specials"))
1020
1021 (add-to-list 'talcum-pikeman-compile-options-functions
1022 (quote ~~~pikeman-srcspc-option))
1023 (add-to-list 'talcum-pikeman-menu-creator-functions
1024 (quote ~~~pikeman-srcspc-menu))
1025
1026 ); namespace

```

```

1027
1028 (provide 'talcum-pikeman)
talcum-pikeman.el ends here.

```

## 6 Où sont les dvi d'antan?

### 6.1 talcum-prod.el

\$Id: talcum-prod.el,v 2.5 2005/08/21 19:48:55 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, [ulmiusers.sarovar.org](http://ulmiusers.sarovar.org).

```

1030
1031 (require 'talcum-tools)
1032 (require 'talcum-compatible)
1033 (require 'talcum-menu)
1034 (require 'advice)
1035
1036 (defgroup talcum-prod nil "Prodding your viewer" :group 'talcum)
1037
1038 (defcustom talcum-use-prod-flag t
1039   "«Enable the whole prodding subsystem."
1040   :group 'talcum
1041   :type 'boolean)
1042
1043 ;; current syntax is:
1044 ;; (predicate command)
1045 (defcustom talcum-prod-commands
1046   '(((lambda () (string-match "xdvi" tex-dvi-view-command))
1047     "pkill -USR1 xdvi.bin")
1048     (ignore
1049      lower-frame))
1049
1050   "A list of things that might be executed after LaTeXing.
1051   Each element is a list (DOTHIS-P DOWHAT), where:
1052   DOTHIS-P is t, nil, or a function. nil means do not execute this,
1053   anything else means execute this.
1054   DOWHAT is either a string or a function. If a function, it is called,
1055   if a string, the string is fed into the shell that runs LaTeX."
1056   :group 'talcum-prod
1057   :type '(repeat
1058           (list
1059            (choice (const t) (function :value ignore))
1060            (choice (string :value "/bin/true") (function :value ignore))
1061            )))
1062
1063 (defcustom talcum-prod-cycle 1
1064   "Talcum checks if it can run the next prod command every n seconds."
1065   :group 'talcum-prod
1066   :type '(choice integer (number :value 0.3)))

```

```

1067
1068 (defvar talcum-prod-tick 0)
1069
1070 (talcum-namespace
1071
1072 (defun talcum-prod-generate-menu ()
1073   (list "--"
1074     (~menu-flag-toggle talcum-use-prod-flag
1075       "Auto-refresh Viewer"))))
1076
1077 (add-to-list 'talcum-menu-functions #'talcum-prod-generate-menu)
1078
1079 (defun ~work-prod-queue (whatnext &optional docu)
1080   (when talcum-use-prod-flag
1081     ;; if delay is high, talcum might not be active anymore
1082     ;; when this triggers.
1083     (~debug 0 "talcum-prod: advice entered, talcum mode is %s" talcum-mode)
1084     (when (or talcum-mode docu) ; evil hack: AucTeX moves to some other buffer
1085       (let ((dothis-p (nth 0 (car whatnext)))
1086             (dowhat (nth 1 (car whatnext)))
1087             (restjobs (cdr whatnext)))
1088         (proc (cond
1089               ((talcum-latex-p) (tex-shell-proc))
1090               ((talcum-auctex-p) (TeX-process docu))))))
1091     (when (or
1092           (equal dothis-p t) ; always
1093           (and (functionp dothis-p) (funcall dothis-p))) ; pred says go!
1094       (~debug 0 "Yes, I'm doing something: %s" dowhat)
1095       ;; next, check if we have a shell string or a fun.
1096       (cond
1097         ((functionp dowhat)
1098          (~debug 0 "Calling fun: %S" dowhat)
1099          (funcall dowhat))
1100         ((stringp dowhat)
1101          ;; big ugly hack: assume we can send command if
1102          ;; no new output is coming for 0.1s. Maybe this
1103          ;; will fail for files with lengthy calculations?
1104          ;; (fp comes to mind, as does beamer)
1105          ;; OTOH, on all but the first call, the other delay
1106          ;; is added to it as well.
1107          (if (or
1108              (and (talcum-latex-p)
1109                  (/= talcum-prod-tick
1110                    (setq talcum-prod-tick
1111                      (buffer-modified-tick (tex-shell-buf)))))
1112              ;;(accept-process-output proc 0 100))
1113              (and (or docu (talcum-auctex-p)) compilation-in-progress))
1114            (progn
1115              (~debug 0 "no, not yet.")
1116              (setq restjobs whatnext))
1117            (~debug 1 "Executing: %s" dowhat)
1118            ;; note: it's the user's responsibility to add "&"
1119            ;; if they want a background process here.

```



```

1120      (cond
1121        ((talcum-latex-p)
1122         (tex-send-command dowhat nil nil))
1123        ((or docu (talcum-auctex-p))
1124         (~debug 0 "TeX-run-command %s %s %s -> %s" docu dowhat ""
1125                  (TeX-run-command docu dowhat ""))))))
1126      (t (error "Unknown kind of dowhat: %s" dowhat))))
1127    (if restjobs
1128      (run-at-time talcum-prod-cycle nil
1129        (quote ~work-prod-queue) restjobs docu))))))
1130
1131    ;; This is called upon activation of the mode.
1132    (defun talcum-prod-init ()
1133      ;; This is easy: intercept tex-file.
1134      (cond
1135        ((talcum-latex-p)
1136         (defadvice tex-file (after talcum-prod-viewer) ;disable)
1137           "Prod viewers to refresh and show the new file."
1138           (~work-prod-queue talcum-prod-commands))
1139         (ad-enable-advice 'tex-file 'after 'talcum-prod-viewer)
1140         (ad-activate 'tex-file)
1141         t)
1142        )
1143      ;; This is tricky. Intercept the plethora of commands AucTeX provides.
1144      ((talcum-auctex-p)
1145       (defadvice TeX-run-format (after talcum-prod-viewer-AucTeX) ;disable)
1146         "Prod viewers to refresh and show the new file."
1147         (~work-prod-queue talcum-prod-commands (ad-get-arg 0)))
1148       (ad-enable-advice 'TeX-run-format 'after 'talcum-prod-viewer-AucTeX)
1149       (ad-activate 'TeX-run-format)
1150       t)))
1151
1152    ;; This is called upon deactivation of the mode.
1153    (defun talcum-prod-exit ()
1154      (cond
1155        ((talcum-latex-p)
1156         (ad-disable-advice 'tex-file 'after 'talcum-prod-viewer)
1157         ;;(ad-activate 'tex-file)
1158         t)
1159        ((talcum-auctex-p)
1160         (ad-disable-advice 'TeX-run-format 'after 'talcum-prod-viewer-AucTeX)
1161         ;;(ad-activate 'TeX-run-format)
1162         t)))
1163
1164    (~sourcefile-versionstamp "$RCSfile: talcum-prod.el,v $"
1165      "$Revision: 2.5 $")
1166
1167    ) ;;namespace
1168
1169    (add-hook 'talcum-enter-mode-hook #'talcum-prod-init)
1170    (add-hook 'talcum-leave-mode-hook #'talcum-prod-exit)
1171
1172

```

```

1173 (provide 'talcum-prod)
talcum-prod.el ends here

```

## 7 Rendering

### 7.1 talcum-render.el

\$Id: talcum-render.el,v 2.20 2005/08/28 09:00:49 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

1175
1176 (require 'talcum-tools)
1177 (require 'talcum-render-ui)
1178 (require 'talcum-render-rules)
1179
1180 (talcum-namespace
1181   (~~sourcefile-versionstamp "$RCSfile: talcum-render.el,v $" "$Revision: 2.20 $"))
1182
1183 (defvar talcum-default-actions
1184   '(:property category
1185     talcum-rendered
1186     :property evaporate t
1187     :property priority 500 ;; default for show-paren-prio is 1000
1188     ; :opener talcum-make-render-opener
1189     ))
1190
1191 (defvar talcum-rex-init-hook nil
1192   "Hooks called at init time, before the rendering structures are
1193   initialized.")
1194
1195 (talcum-namespace
1196   (defvar ~~match-any-re)
1197
1198   (defvar ~~rendermap (~~mmap-new))
1199
1200   (defun talcum-render-string (str)
1201     "Add renderings to a string and return the result."
1202     (if (or (null str)
1203             (string= str ""))
1204         ""
1205         (with-temp-buffer
1206           (with-syntax-table ~~mode-syntax-table
1207             (insert str)
1208             (talcum-render-region (point-min) (point-max) 'REPLACE)
1209             ;; We're actually carrying font-lock info all along, so don't
1210             ;; throw it away now :-D
1211             ;; (buffer-substring-no-properties (point-min) (point-max))
1212             (buffer-string))))))

```

```

1213
1214
1215 ;; Go through the region and call ~-maybe-apply-rule-at-point for
1216 ;; each match of any item according to the current ~-match-any-re.
1217 ;; ~-maybe-apply-rule-at-point must check itself whether the rule is
1218 ;; applicable. We leave point just after the match, i.e. after the
1219 ;; closing brace of a \begin{} or after the last char of a command.
1220 ;;
1221 ;; We save match data and excursion, so the rules are free to move
1222 ;; around and do stuff.
1223 ;;
1224 ;; The region to render is actually extended to the next following
1225 ;; non-word character (or end of buffer) for technical reasons.
1226 (defun talcum-render-region (bor eor &optional replace?)
1227   (interactive "r")
1228
1229   ;; Init thingy: if render-list is dirty, rebuild.
1230   (if (get (quote ~-rendermap) 'talcum-map-is-dirty)
1231       (talcum-render-reinit))
1232
1233   (when (and talcum-use-render-flag
1234             (< bor eor))
1235     (~save (excursion restriction)
1236            (~without-touching-buffer
1237
1238              (unless (~value-safe ~-user-point)
1239                    (setq ~-user-point (point)))
1240
1241              (let* ((bor bor)
1242                     ;; make sure we can see the character after a macro end, to
1243                     ;; check if it's to be rendered.
1244                     (eor (save-excursion
1245                           (goto-char eor)
1246                           (skip-syntax-forward "w")
1247                           (or (eobp)
1248                               (forward-char 1))
1249                           (point)))
1250                     (final-point (copy-marker bor)) ;; always move forward
1251
1252                     new-end-or-nil
1253                     matching-rules)
1254
1255              (goto-char bor)
1256
1257              (while (search-forward-regexp
1258                     ~-match-any-re
1259                     eor 'end-and-nil)
1260
1261                (setq matching-rules
1262                      (~mmap-get (concat (match-string 1)
1263                                          (match-string 2))
1264                                ~-rendermap)
1265                      new-end-or-nil nil)

```

```

1266      (move-marker final-point (or (match-end 1) (match-end 2)))
1267
1268      (dolist (rule matching-rules)
1269        (unless new-end-or-nil
1270          (~save (excursion match-data)
1271                (setq new-end-or-nil
1272                      (~maybe-apply-rule-at-point rule eor replace?))
1273                (if new-end-or-nil
1274                  (move-marker final-point new-end-or-nil))))))
1275      (goto-char final-point))
1276
1277      (setq ~user-point nil)
1278      (set-marker final-point nil)
1279      t))))))
1280
1281  ;; Check and apply RULE at this point.
1282  ;;
1283  ;; When called, we have point just after the matcher string,
1284  ;; match-data slot MATCH points at the match. We may move point and trash
1285  ;; match-data. By setting premature-abort to non-nil, we may prevent
1286  ;; other rules from looking at this as well.
1287  ;;
1288  ;; return what we think is a good point for rendering to continue.
1289  (defun ~maybe-apply-rule-at-point (rule scopelimit &optional replace?)
1290    (let* (args
1291           envlimit
1292           (pattern (~pattern-part rule))
1293           (argspec (~pattern-args pattern))
1294           (MATCH (if (equal ':macro (~pattern-type pattern))
1295                      1
1296                      2))
1297           (startofpattern (match-beginning MATCH))
1298           (endofpattern (copy-marker (match-end MATCH))))
1299      (goto-char endofpattern)
1300      (if
1301        (and
1302          ;; This is cheap. Maybe some context-sensitive rule already prevents us.
1303          (~rule-desired-p rule)
1304
1305          ;; parse-arg. returns string if failure, eg. unclosed braces
1306          (not (stringp (setq args
1307                           (~parse-arguments argspec scopelimit))))))
1308        (set-marker endofpattern (point))
1309
1310        ;; this would mess with the user's point
1311        (not (and (not replace?)
1312                  (~value-safe ~user-point)
1313                  (>= ~user-point startofpattern)
1314                  (< ~user-point endofpattern))))
1315
1316      ;; if not environment, succeed, otherwise see if we can get the
1317      ;; end-of-environment, and then put it into the args.

```

```

1319     (if (not (equal 'environment (pattern-type pattern)))
1320         t
1321         (setq envlimit
1322             (~find-end-of-environment
1323              (~pattern-name pattern) scopelimit))
1324         (unless (stringp envlimit)
1325             ;;end of env in scope! (implicit nil otherwise)
1326             (set-marker endofpattern (point))
1327             ;;(message "considering '%s'" (buffer-substring startofpattern endofpattern))
1328             (~append! args (list envlimit)))))) ;implicit t
1329
1330 ;; Yes! Go ahead and do it!
1331 ;; now, if somebody matched, always abort.
1332 (prog1 endofpattern
1333     (save-excursion
1334         (~really-apply-rule-at-point rule args
1335                                     (cons startofpattern
1336                                             (marker-position endofpattern))
1337                                     replace?)))
1338 ;; otherwise, hope for some other rule.
1339 nil)
1340 ))
1341
1342 ;; Really apply RULE at point.
1343 ;;
1344 ;; We know it's applicable and all the arguments are there.
1345 ;; PATTERNPOS is a cons with car just before and cdr just after the
1346 ;; whole thing we will hide. ARGS contains the arguments as strings,
1347 ;; and buffer positions for the environment body.
1348 (defun ~really-apply-rule-at-point (rule args patternpos &optional replace?)
1349   (let* ((actions (append talcum-default-actions (~action-part rule)))
1350          (localvars nil)
1351          (ov (make-overlay (car patternpos) (cdr patternpos)))
1352          (talcum-make-render-opener 'talcum-make-render-opener)
1353          action arg1 arg2 ;; current action and its args
1354          no-display-flag
1355          )
1356   ;; run over the actions and do what they say.
1357   ;;
1358   (dolist (subov (overlays-in (car patternpos) (cdr patternpos)))
1359     (if (overlay-get subov 'display)
1360         (delete-overlay subov)))
1361   (while actions
1362     (setq action (pop actions))
1363     (cond
1364       ((run-hook-with-args-until-success
1365        'talcum-apply-action-functions action 'actions ov))
1366       ((eq action 'var) ;; Make a local var known.

```

```

1372 (setq arg1 (pop actions))
1373 (if (not (consp arg1))
1374     (~var-scope-start arg1)
1375     (~var-scope-start (car arg1))
1376     (set (car arg1) (cdr arg1))
1377     (setq arg1 (car arg1)))
1378 (push arg1 localvars))
1379
1380 ((eq action ':eval) ;; eval the next element
1381  (eval (pop actions)))
1382
1383 ((eq action ':opener)
1384  (setq talcum-make-render-opener (pop actions)))
1385
1386 ((eq action ':face)
1387  (~with-secretly-touching-buffer
1388   (add-text-properties
1389    (overlay-start ov) (overlay-end ov)
1390    (list 'face
1391          (cons (pop actions)
1392                (get-text-property (overlay-start ov)
1393                                   'face)))))
1394
1395 ((eq action ':property) ;; set the (overlay) property
1396  (overlay-put ov (pop actions) (pop actions)))
1397
1398 ((eq action ':display) ;; set the display string
1399  (if replace?
1400      (~with-secretly-touching-buffer
1401       (goto-char (car patternpos))
1402       (insert (~eval-display-spec (pop actions) args))
1403       (delete-region (point) (min (+ (point)
1404                                       (cdr patternpos)
1405                                       (- (car patternpos)
1406                                           (point-max)))))
1407       (overlay-put ov 'display
1408                    (format "%s" (~eval-display-spec (pop actions)
1409                                                         args)))))
1410
1411 ((eq action ':recurse) ;; just render the body, without generating overlay
1412  (setq no-display-flag t)
1413  (let ((body (car (last args))))
1414    (talcum-render-region (car body) (cadr body))))
1415
1416 ((eq action ':help) ;; set the flyover text
1417  (if replace? ;; doesn't make much sense if we only get the string.
1418      (pop actions)
1419      (overlay-put ov 'help-echo
1420                   (format "%s" (~eval-display-spec (pop actions)
1421                                                       args)))))
1422 (t (message "unknown action! %s" action))
1423 ))
1424

```

```

1425 (if (string= (overlay-get ov 'display) "")
1426     (overlay-put ov 'display "<empty>"))
1427
1428 (unless (or replace? no-display-flag)
1429     (funcall talcum-make-render-opener ov patternpos))
1430 (if no-display-flag
1431     (delete-overlay ov))
1432
1433 ;; all done, remove local scope.
1434 (dolist (lv localvars)
1435     (~~var-scope-end lv))
1436 ))
1437
1438 ;; turn a display spec into a string with text properties.
1439 (defun ~~eval-display-spec (spec patternargs)
1440 (let* (result
1441       (render? (eq (car-safe spec) ':rendered))
1442       (spec (if render? (cadr spec)
1443                 spec)))
1444 (setq result
1445       (cond
1446         ((run-hook-with-args-until-success
1447          'talcum-display-spec-functions spec patternargs))
1448         ((stringp spec) spec)
1449         ((integerp spec) (~~maybe-strip-parens (nth spec patternargs)))
1450         ((eq spec 'body)
1451          (let ((bodypos (car (last patternargs))))
1452            (buffer-substring (car bodypos)
1453                              (cadr bodypos))))
1454         ((functionp spec) (funcall spec patternargs))
1455         (t (error "Unknown result specifier %s" spec))
1456       ))
1457 (if render?
1458     (setq result (talcum-render-string result)))
1459 result))
1460
1461 ;; The cheap, safe variant.
1462 (defun talcum-make-render-opener (ov pos)
1463 (talcum-namespace
1464  (~~with-secretly-touching-buffer
1465   (let ((beg (overlay-start ov))
1466         (end (overlay-end ov)))
1467     (add-text-properties
1468      beg end
1469      '(point-entered (lambda (_ __)
1470                        (let* ((a (overlay-start ,ov))
1471                              (o (overlay-end ,ov)))
1472                          (when (and a o)
1473                            ;(message "a&o %s %s, point %s" a o (point))
1474                            (talcum-unrender-region a o)
1475                            ;(message "ok.")
1476                            t)
1477                          t))))))

```

```

1478         )))))
1479
1480 (defun ~render-open/close (ov enter-or-leave)
1481   (~without-touching-buffer
1482     (~with-secretly-touching-buffer
1483       (let* ((beg (overlay-start ov))
1484              (end (overlay-end ov)))
1485         (cond
1486           ((equal enter-or-leave 'enter)
1487            (overlay-put ov 'display nil)
1488            ;; (overlay-put ov 'face nil)
1489            (font-lock-fontify-region beg end)
1490            (remove-text-properties
1491             beg end
1492             (list 'point-entered nil)))
1493           ((and (equal enter-or-leave 'leave)
1494                (null (overlay-get ov 'display))
1495                (overlay-start ov)
1496                (overlay-end ov)
1497                (eq (overlay-buffer ov) (current-buffer))
1498                (or (< (point) (overlay-start ov))
1499                    (> (point) (overlay-end ov))))
1500            )
1501           (delete-overlay ov)
1502           (remove-text-properties
1503            beg end
1504            (list 'point-entered nil 'point-left nil))
1505           (talcum-render-region beg end))))))
1506
1507
1508
1509
1510 (defun ~parse-arguments (argspec end)
1511   "Parse options ARGSPEC in current buffer. Each opening brace matches a
1512   whole braced sexp, all but { are optional. * is optional. Any other
1513   char matches itself. Return a list of what was parsed, or a string
1514   detailing what went wrong. END limits how far in the buffer we will
1515   go."
1516   (catch 'mismatch
1517     (let (result found)
1518       (dolist (wanted (append argspec nil)
1519                             (if (> (point) end)
1520                                 (throw 'mismatch "Exceeded")
1521                                 result)))
1522         ;; we can always have whitespace between args.
1523         (skip-syntax-forward "-" end)
1524         (setq found (char-after))
1525         (if (null found) (throw 'mismatch "Premature EOB."))
1526         (~append! result
1527                   (list
1528                    (cond
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

```



```

1531         'talcum-parse-argument-functions wanted))
1532
1533     ((or (equal wanted ':mandatory)
1534          (equal wanted ?{}))
1535      (with-syntax-table ~~sanitized-mode-syntax-table
1536        (~~scantoken)))
1537
1538     ;; [ gets nothing or a [sexp]
1539     ((or (equal wanted ':optional)
1540          (equal wanted ?\[ ))
1541      (if (= found ?\[)
1542          (catch 'notclosed
1543              (with-syntax-table ~~sanitized-mode-syntax-table
1544                (let ((s (point))
1545                      (forward-sexp-function nil))
1546                  (while (not (looking-at "[^{}]*"))
1547                      (if (> (point) end)
1548                          (throw 'notclosed ""))
1549                      (forward-sexp 1))
1550                  (search-forward "]")
1551                  (if (> (point) end)
1552                      (throw 'notclosed ""))
1553                  (buffer-substring
1554                     s (point))))))
1555          ""))
1556
1557     ;; special hardcoded hack: * gets itself, optionally
1558     ((or (equal wanted ':star)
1559          (equal wanted ?*))
1560      (if (= found ?*)
1561          (buffer-substring
1562             (point) (~~point-after (forward-char 1)))
1563          ""))
1564
1565
1566     ;; better not: more complaints = less bugs :D
1567     ;; ;; any other char gets itself
1568     ;; ((= wanted found)
1569     ;; (buffer-substring
1570     ;; (point) (~~point-after (forward-char 1))))
1571
1572     ;; if all else fails, complain and exit
1573     (t (throw 'mismatch
1574           (format "looking at %c and no rule matches."
1575                  found))))))
1576
1577     nil))))))
1578
1579     (defun ~maybe-strip-parens (str)
1580       "Remove one level of enclosing parentheses from string, if any. We
1581       also strip leading and trailing whitespace in that case. Note: we do
1582       not check the parens match each other!"
1583       (if (and str (string-match "^[:space:]]*\\s(\\([^\000]*\\)\\s)[[:space:]]*$" str))
          ;; evil hack there, but not handling NUL is better than not handling \n.

```

```

1584      (match-string 1 str)
1585      str))
1586
1587 (defun ~find-end-of-environment (envname end)
1588   "Find the end of the environment called envname.
1589
1590   We assume point after the \\begin, since somebody else might have
1591   parsed arguments already. (Technically, point after the \\ is
1592   sufficient.) We return a list with the position of the first char in
1593   and not in the env. body anymore. We leave point just after the
1594   closing brace of the \\end We only check for nesting of this
1595   environment, not any other. (This means we probably don't DTRT in the
1596   preamble, if the begin-part of a newenvironment does a \\begin{foo}).
1597   Returns a string if we run out of buffer before the env is closed."
1598   (catch 'endofbuffer
1599     (let ((depth 1)
1600           (re (concat "\\(\\\\begin\\\\\\\\end\\\\{" envname "\\}"))
1601               (a (point))
1602                 o)
1603           (while (and (not (zerop depth))
1604                      (or (search-forward-regexp re end t)
                          (throw 'endofbuffer "Exceeded scope"))))
1605             ;(message "d%s, %s" depth (match-string 1))
1606             (if (string= (match-string 1) "\\begin")
1607                 (setq depth (1+ depth))
1608                 (setq depth (1- depth))))
1609           (list a
1610                 (save-excursion
1611                   (search-backward "\\end{")
1612                   (point))))
1613     )))
1614
1615
1616 ) ;namespace
1617
1618 (provide 'talcum-render)

```

## 7.2 talcum-render-ui.el

\$Id: talcum-render-ui.el,v 2.18 2005/08/28 09:00:49 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

1619
1620 (require 'talcum-tools)
1621 (require 'talcum-render-rules)
1622 (require 'talcum-custom)
1623 (require 'talcum-menu)
1624
1625 (talcum-namespace
1626   (~sourcefile-versionstamp "$RCSfile: talcum-render-ui.el,v $" "$Revision: 2.18 $"))

```

```

1627
1628 (defcustom talcum-use-render-flag t
1629   "Enable the whole rendering subsystem."
1630   :group 'talcum
1631   :type 'boolean)
1632
1633 (defvar talcum-render-list nil
1634   "The true render list.")
1635
1636 (talcum-namespace
1637   (defvar ~~mode-syntax-table nil)
1638   (defvar ~~sanitized-mode-syntax-table nil))
1639
1640 (defgroup talcum-render nil
1641   "Displaying beautiful letters instead of ugly macros"
1642   :group 'talcum)
1643
1644 (defcustom talcum-user-render-list nil
1645   "Local rendering rules"
1646   :group 'talcum-render
1647   :get 'talcum-render-rule-getter
1648   :set 'talcum-render-rule-setter
1649   :initialize 'custom-initialize-default
1650   :type talcum-render-rule-custom-type)
1651
1652 (defvar talcum-render-features 'nil)
1653
1654 (defvar talcum-apply-action-functions nil
1655   "Abnormal hook to introduce new action commands.
1656
1657 Each function is called with the overlay, the current action and the
1658 list of rest-args-and-further-actions. The function should not alter
1659 these unless it has verified it is applicable. The function should
1660 return a non-nil value only if no other rules shall apply to the
1661 action.")
1662
1663 (defvar talcum-display-spec-functions nil
1664   "Abnormal hook to introduce new display specifiers.
1665
1666 Each function is called with two parameters: the spec and a list
1667 containing the macro's arguments. The function returns a string with
1668 text properties as appropriate, or nil if it is not the right one.
1669 Note that you cannot prevent the :rendered syntax from working with
1670 this — :rendered is hidden from you beforehand. You may, as a dirty
1671 hack, set the value of render? to nil to override :rendered, though,
1672 and to non-nil to simulate :rendered.")
1673
1674 (defvar talcum-parse-argument-functions nil
1675   "Abnormal hook used to introduce new kinds of macro parameter style.
1676
1677 Each function is called with one argument: the character that
1678 identifies the parameter style. Functions must return a non-nil value
1679 and leave point at the end of the parameter only if they have handled

```

```

1680 that parameter. Otherwise, they may not move point. If you have found
1681 your parameter to be there, but empty (like missing *, return the
1682 empty string "\\").")
1683
1684 (defvar talcum-init-specpdl 6000
1685   "If you get \"exceeds max-specpdl-size\" errors on startup, set this
1686   to a higher value. (As comparison: GNU Emacs' default is 600.)")
1687
1688 (defvar talcum-semantic-braces
1689   '(?( ?) ?[ ?] ?« ?» ?< ?> ?\\")
1690   "Not all braces should be made to match each other. For example, a)
1691   does not need an opening brace.")
1692
1693 (defgroup trexes
1694   nil ;members
1695   "Customization options for Talcum Rendering EXtensions."
1696   :group 'talcum)
1697
1698 (defcustom talcum-render-trexes
1699   '((trex-math :math)
1700     (trex-unicode :unicode)
1701     (trex-markup :markup)
1702     (trex-xref :xref)
1703     (trex-sectioning :sectioning))
1704   "Trexes that will be loaded."
1705   :group 'trexes
1706   :type 'sexp)
1707
1708 (talcum-namespace
1709 ;; This used to be in talcum-render. Unfortunately, if font-lock
1710 ;; kicks in before talcum-render is completely loaded, we would get a
1711 ;; byte-compile warning: assignment to free var. I wouldn't give a
1712 ;; flying shit about that if it didn't switch-window away an actual
1713 ;; useful buffer with actual information in it. Thank you Emacs!
1714 (defvar ~~user-point nil)
1715 (make-variable-buffer-local (quote ~~user-point)))
1716
1717 ;; XXX HACK XXX should use the proper functions.
1718 ;;
1719 ;; 2005AUG21: Nevermind, from what I can see, all other font-lock
1720 ;; methods disappear in Emacs 22.
1721 (talcum-namespace
1722 (defadvice jit-lock-fontify-now
1723   (before talcum-user-point-setter activate)
1724   "Save user's point."
1725
1726   (unless (~~value-safe ~~user-point)
1727     (setq ~~user-point (point))))))
1728
1729
1730 (talcum-namespace
1731
1732 ;; initialize internal stuff.

```

```

1733 ;;
1734 ;; currently, this mainly recreates the internal hash structures and
1735 ;; regexpen. This should be called when entering the mode and
1736 ;; whenever talcum-render-list or talcum-render-features are changed.
1737 (defun talcum-render-reinit ()
1738   (interactive)
1739   (let* ((max-specpdl-size (max max-specpdl-size
1740                                talcum-init-specpdl))
1741          matcher macro-matchers other-matchers)
1742
1743     (setq ~~rendermap (~~mmap-new))
1744     (with-temp-message "Building rendering structures..."
1745       (dolist (rule (append talcum-render-list
1746                             talcum-user-render-list))
1747         (when (~~rule-desired-p rule)
1748           (setq matcher (~~pattern-matcher (~~pattern-part rule)))
1749           (if (equal ':macro (~~pattern-type (~~pattern-part rule)))
1750               (push matcher macro-matchers)
1751               (push matcher other-matchers))
1752           (~~mmap-put matcher rule ~~rendermap))))
1753
1754     (setq ~~match-any-re
1755           (format "\\(%s\\)\\>\\|\\(%s\\)"
1756                 (if macro-matchers
1757                     (regexp-opt macro-matchers)
1758                     "\\000") ;; XXX hack, because regexp-opt is broken.
1759                 (if other-matchers
1760                     (regexp-opt other-matchers)
1761                     "\\000")))
1762     t)
1763     (message "Building rendering structures...done")
1764     (put (quote ~~rendermap) 'talcum-map-is-dirty nil)
1765     t))
1766
1767 (defun talcum-make-render-map-dirty ()
1768   (put (quote ~~rendermap) 'talcum-map-is-dirty t))
1769
1770 (defun talcum-render-init ()
1771
1772   (dolist (trex talcum-render-trexes)
1773     (require (car trex))
1774     (dolist (feat (cdr trex))
1775       (talcum-add-render-feature feat)))
1776
1777   (message "rex-hook")
1778   (run-hooks 'talcum-rex-init-hook)
1779   (font-lock-add-keywords nil '(talcum-render-fontlock-function) 'atend)
1780   (talcum-render-reinit)
1781   (setq ~~mode-syntax-table (copy-syntax-table))
1782   (setq ~~sanitized-mode-syntax-table (copy-syntax-table ~~mode-syntax-table))
1783   ;; it is stupid of forward-sexp et al. to look at ( and ) which are not
1784   ;; usually syntactical parens: \texttt {}} would take the {} as its argument.
1785   (dolist (ch talcum-semantic-braces)

```

```

1786 ;; How the fuck does putting a R-val into a function have
1787 ;; side effects on the var with that R-val?
1788 (modify-syntax-entry ch "." ~sanitized-mode-syntax-table))
1789 t)
1790
1791 (defun talcum-render-exit ()
1792   ;; no funs use timers anymore.
1793   (font-lock-remove-keywords nil '(talcum-render-fontlock-function))
1794   (talcum-unrender-region (point-min) (point-max))
1795   t)
1796
1797 (defun talcum-unrender-region (bor eor)
1798   "Remove the renderings created by talcum-render."
1799   (interactive "r")
1800   (save-excursion
1801     (~without-touching-buffer
1802      (~with-secretly-touching-buffer
1803       (remove-text-properties bor eor '(point-entered nil
1804                                     point-left nil
1805                                     face nil
1806                                     fontified nil))))
1807     (mapc (lambda (ov)
1808             (if (equal (overlay-get ov 'category) 'talcum-rendered)
                (delete-overlay ov)))
1809           (overlays-in bor eor))))))
1810
1811
1812
1813 (defun talcum-render-find-suitable-start ()
1814   "Return a buffer position backwards from point where
1815 talcum-render-dwim should start."
1816   (~without-touching-buffer
1817    (save-excursion
1818     (search-backward "%%%" (point-min) 'go-limit)
1819     (point))))
1820
1821 (defun talcum-render-find-suitable-end ()
1822   "Return a buffer position forward from point where
1823 talcum-render-dwim should start."
1824   (~without-touching-buffer
1825    (save-excursion
1826     (search-forward "%%%" (point-max) 'go-limit)
1827     (point))))
1828
1829 (defun talcum-render-dwim (pfx)
1830   "Refresh the unclutterings in the current suitable region, as found
1831 by t-r-find-suitable-start, -end, which see. With prefix C-u, remove
1832 all unclutterings from that region."
1833   (interactive "P")
1834   (setq ~user-point (point))
1835   (let ((bos (talcum-render-find-suitable-start))
1836         (eos (talcum-render-find-suitable-end)))
1837     (talcum-unrender-region bos eos)
1838     (unless (and (listp pfx) pfx)

```

```

1839   (talcum-render-region bos eos))))
1840 ) ;namespace
1841
1842
1843 (defconst talcum-render-parsep-re "\\(\\\\\\par[^a-zA-Z\\\\)\\\\\\(\\n\\n\\\\)"
1844   "A regexp to match the beginning or end of a paragraph.")
1845
1846 (defun talcum-render-paragraph ()
1847   (interactive)
1848   "Render the paragraph point is in or after.
1849 May you'll want to advice tex-terminate-paragraph to call this."
1850   (let* ((inhibit-point-motion-hooks t)
1851          (p (point)))
1852     (talcum-render-region
1853      (save-excursion (or (search-backward-regexp talcum-render-parsep-re (point-min) t)
1854                        (point-min)))
1855      (save-excursion (or (search-forward-regexp talcum-render-parsep-re (point-max) t)
1856                        (point-max)))))
1857
1858 (defun talcum-newpar-and-render ()
1859   (interactive)
1860   (let* ((inhibit-point-motion-hooks t)
1861          (save-excursion
1862            (talcum-render-paragraph))
1863          (insert "\\n\\n")))
1864
1865 (defun talcum-render-fontlock-function (limit)
1866   (talcum-render-region (point) limit)
1867   nil)
1868
1869 (defun talcum-render-fontf-function (start)
1870   (let ((end (save-excursion
1871                (catch 'end-of-buffer
1872                  (forward-char 600))
1873                  (or (search-forward-regexp
1874                      talcum-render-parsep-re
1875                      (point-max)
1876                      'silent-go-limit)
1877                    (point-max))))))
1878     (message "rendering from %s to %s" start end)
1879     (talcum-render-region start end)
1880     t))

```

```
1883 (define-key talcum-keymap "#" 'talcum-render-dwim)
1884 (define-key talcum-keymap "\C-M" 'talcum-newpar-and-render)
```

```

1886 (talcum-namespace
1887 (defun talcum-render-make-menu ()
1888   (list
1889     (~menu-flag-toggle talcum-use-render-flag "Rendering"
1890                       (progn

```

```

1891         (if talcum-use-render-flag
1892           (talcum-render-init)
1893           (talcum-render-exit))
1894         (talcum-render-dwim 'on)))
1895   '["Render: DWIM" talcum-render-dwim]
1896   '["Render buffer" (talcum-render-region (point-min) (point-max))
1897     :included talcum-use-render-flag]
1898   '["Render region" (talcum-render-region (region-beginning) (region-end))
1899     :included talcum-use-render-flag]
1900   '["Unrender buffer" (talcum-unrender-region (point-min) (point-max))
1901     :included (not talcum-use-render-flag)]
1902   '["Unrender region" (talcum-unrender-region (region-beginning) (region-end))
1903     :included (not talcum-use-render-flag)]
1904   (let (trexmenu)
1905     ;; A submenu with a checkbox for every loaded TREX.
1906     (append '("TREXes" :active talcum-use-render-flag)
1907       (dolist (trex (talcum-render-features) (nreverse trexmenu))
1908         (push '[, (symbol-name trex)
1909           (progn
1910             (if (talcum-has-render-feature ,trex)
1911               (talcum-remove-render-feature ,trex)
1912               (talcum-add-render-feature ,trex))
1913             (talcum-render-reinit))
1914           :style toggle
1915           :selected (talcum-has-render-feature ,trex)
1916           ]
1917         trexmenu))))
1918   ))
1919   (add-to-list 'talcum-menu-functions #'talcum-render-make-menu)
1920   ) ; namespace
1921
1922   (add-hook 'talcum-enter-mode-hook #'talcum-render-init)
1923   (add-hook 'talcum-leave-mode-hook #'talcum-render-exit)
1924
1925   (provide 'talcum-render-ui)

```

### 7.3 talcum-render-rules.el

\$Id: talcum-render-rules.el,v 2.4 2005/03/06 22:22:43 ulmi RELEASE \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-1.3a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, [ulmiusers.sarovar.org](http://ulmiusers.sarovar.org).

```

1926
1927 ;; '((type name args feature) action-part)
1928
1929 ;; type is one of :macro :environment :active
1930
1931 ;; :macros get a \ prepended, :env.s get a \begin{ }, actives are left
1932 ;; unchanged
1933

```



```

1934 (require 'talcum-tools)
1935
1936 (talcum-namespace
1937   (~sourcefile-versionstamp "$RCSfile: talcum-render-rules.el,v $" "$Revision: 2.4 $"))
1938
1939 (talcum-namespace
1940   (defconst ~~backslash "\\")
1941     "Regexp matching a backslash.")
1942
1943 (defun ~~not-after-backslash (re)
1944   (concat "\\(?:^\\\\[" (regexp-quote ~~backslash) "]"\\)" re))
1945
1946
1947 (defun ~~pattern-part (rule)
1948   (car rule))
1949 (defun ~~action-part (rule)
1950   (cdr rule))
1951
1952 (defun ~~pattern-type (pattern)
1953   (nth 0 pattern))
1954 (defun ~~pattern-name (pattern)
1955   (nth 1 pattern))
1956 (defun ~~pattern-args (pattern)
1957   (nth 2 pattern))
1958 (defun ~~pattern-feat (pattern)
1959   (nth 3 pattern))
1960
1961 (defun ~~pattern-matcher (pattern)
1962   "Make a matchable string from the pattern."
1963
1964 Note that we go through a level of regexp-quote before we search in
1965 the buffer."
1966   (let ((name (~~pattern-name pattern))
1967         (type (~~pattern-type pattern)))
1968     (cond
1969      ((eq type ':macro)
1970       (concat ~~backslash name))
1971      ((eq type ':environment)
1972       (concat ~~backslash "begin{" name "}"))
1973      ((eq type ':active)
1974       name)
1975      (t
1976       (~debug 9 "Huh? What's type %s of %s?" type name)
1977       nil))))
1978
1979 ;; check a complex specification rule.
1980 ;; a sr can be conjunctive or disjunctive.
1981 ;; a csr is true iff all elements are fulfilled
1982 ;; a dsr is true iff not all elements are not fulfilled
1983 ;;
1984 ;; elements of a sr can be:
1985 ;; - integers : true iff not larger than talcum::thres
1986 ;; - functions: true iff they return not-nil

```

```

1987 ;; - lists      : lists within a dsr are csrs and vice versa
1988 ;; - keywords : true iff that keyword is in talcum::present-features
1989 (defun ~rule-desired-p (rule &optional disj)
1990   ;; we introduce boolean ops that guarantee to return t or nil to
1991   ;; prevent seeping out of information.
1992   ;; (also, we cannot do (funcall (if pred and or) a b) since and, or are sforms)
1993   (let ((&& (lambda (a b) (if a (if b t))))
1994         (|| (lambda (a b) (if a t (if b t)))))
1995     (~rule-desired-p_ (~pattern-feat (~pattern-part rule)) disj)))
1996
1997 ;; This function does the proper work for ~applicable. Splitting it up
1998 ;; prevents us from defining && and || over and over again.
1999 ;;
2000 ;; disj has a double role: it discriminates csr and dsr, and it must
2001 ;; also be the respective zero, i.e.
2002 ;; - nil for conjunction (since nil && x == nil for all x)
2003 ;; - t for disjunction (since t || x == t for all x)
2004 (defun ~rule-desired-p_ (speclist &optional disj)
2005   (catch 'shortcut
2006     (let* ((applies? (not disj))
2007            (speclist (if (consp speclist) speclist (list speclist)))
2008            (dolist (feat speclist applies?)
2009              (setq applies?
2010                (funcall (if disj || &&)
2011                          (cond
2012                            ((eq t feat) t)
2013                            ((eq nil feat) nil)
2014                            ((integerp feat) (>= talcum-render-threshold feat))
2015                            ((functionp feat) (funcall feat))
2016                            ((listp feat) (~rule-desired-p_ feat (not disj)))
2017                            ((keywordp feat) (talcum-has-render-feature feat))
2018                            (t (error "unknown case of speclist: %s" feat)))
2019                          applies?))
2020            (if (eq applies? disj)
2021                (throw 'shortcut applies?))))))
2022
2023 ;; XXX to be changed to use a hashtable.
2024 (defun talcum-render-features ()
2025   (interactive)
2026   (message "%s" talcum-render-features)
2027   talcum-render-features)
2028
2029 (defun talcum-has-render-feature (feat)
2030   (memq feat talcum-render-features))
2031
2032 (defun talcum-add-render-feature (feat)
2033   (interactive "SFeature keyword:")
2034   (add-to-list 'talcum-render-features feat))
2035
2036 (defun talcum-remove-render-feature (feat)
2037   (interactive "SFeature keyword:")
2038   (setq talcum-render-features
2039         (delq feat talcum-render-features)))
2039

```

```

2040
2041 ); namespace
2042
2043 (provide 'talcum-render-rules)

```

## 8 It's all been done before.

### 8.1 trex-7bit.el

\$Id: trex-7bit.el,v 1.2 2005/03/06 21:37:27 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

2044 (require 'talcum-render)
2045
2046 (defvar trex-7bit-mappings
2047   '(("a" . "ä")
2048     ("o" . "ö")
2049     ("u" . "ü")
2050     ("s" . "ß")
2051     ("A" . "Ä")
2052     ("O" . "Ö")
2053     ("U" . "Ü")))
2054
2055 (dolist (ch trex-7bit-mappings)
2056   (setq talcum-render-list
2057     (cons '(:active ,(car ch) nil (:7bit :german))
2058       :display ,(cdr ch))
2059     talcum-render-list)))
2060
2061 (talcum-make-render-map-dirty)
2062
2063 (provide 'trex-7bit)

```

### 8.2 trex-lists.el

\$Id: trex-lists.el,v 2.5 2005/03/06 21:37:55 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

2064 (require 'talcum-render)
2065
2066 (defun trex-lists-arabic (i) (number-to-string i))
2067 (defun trex-lists-alph (i) (char-to-string (+ 96 i)))
2068 (defun trex-lists-Alph (i) (upcase (trex-lists-alph i)))
2069
2070 ;; Let's hope nobody has lists with more than 39 roman items...

```

```

2071 (defun trex-lists-roman (i)
2072   (let* ((result "")
2073          (i i))
2074     (while (>= i 10)
2075       (setq result (concat result "x")
2076                 i (- i 10)))
2077     (concat result
2078       (nth i '(" " "i" "ii" "iii" "iv" "v" "vi" "vii" "viii" "ix")))))
2079
2080 (defun trex-lists-Roman (i) (upcase (trex-lists-roman i)))
2081
2082 (setq talcum-render-list
2083   (append
2084     talcum-render-list
2085     '(((environment "enumerate" "" (:lists))
2086       :var (trex-lists-nesting . (:enum 0))
2087       :recurse)
2088       ((environment "itemize" "" (:lists))
2089         :var (trex-lists-nesting . (:item 0))
2090         :recurse)
2091       ((environment "description" "" (:lists))
2092         :var (trex-lists-nesting . (:desc 0))
2093         :recurse)
2094       ((macro "item" "[" (:lists))
2095         :display trex-lists-spitzmarke)
2096     )))
2097 (talcum-make-render-map-dirty)
2098
2099
2100 (defun trex-lists-spitzmarke (macparams)
2101
2102   (if (not (boundp 'trex-lists-nesting))
2103       ;; Whoopsie! We're not in scope. Fallback.
2104       "*** "
2105       (if (not (string= "" (nth 0 macparams)))
2106           ;; Explicitly given by user, use that
2107           (talcum-namespace (~maybe-strip-parens (nth 0 macparams)))
2108
2109           ;; all else fails: think for ourselves.
2110           (let* ((list-type (car-safe trex-lists-nesting))
2111                  (nest-depth 0))
2112
2113             (setq nest-depth
2114               (dolist (n (talcum-var-history 'trex-lists-nesting) nest-depth)
2115                 (if (equal list-type (car-safe n))
2116                     (setq nest-depth (1+ nest-depth))))
2117               trex-lists-nesting (list (car trex-lists-nesting)
2118                                         (1+ (cadr trex-lists-nesting))))
2119
2120             (cond
2121              ((equal :enum list-type)
2122               (funcall (nth nest-depth trex-lists-enum-style)
2123                        (cadr trex-lists-nesting)))

```

```

2124
2125      ((equal :item list-type)
2126       (funcall (nth nest-depth trex-lists-item-style)))
2127
2128      ((equal :desc list-type)
2129       ;; An item of a description has an arg.
2130       "[nodesc?]")
2131
2132      (t
2133       "??")))))))
2134
2135
2136 (defgroup trex-lists nil
2137   "Rendering itemize, enumerate and description."
2138   :group 'trexes)
2139
2140 (defcustom trex-lists-enum-style
2141   (list 'ignore ;; dummy element
2142         (lambda (i) (concat (trex-lists-arabic i) "."))
2143         (lambda (i) (concat (trex-lists-alph i) "."))
2144         (lambda (i) (concat (trex-lists-roman i) "."))
2145         (lambda (i) (concat (trex-lists-Alph i) ".")))
2146   )
2147   "Functions that yield a list bullet with a number, one for each
2148 nesting depth."
2149   :group 'trex-lists
2150   :type '(list
2151          (const ignore)
2152          function
2153          function
2154          function
2155          function
2156          ))
2157
2158 (defcustom trex-lists-item-style
2159   '(ignore ;; dummy element
2160     (lambda () "o ")
2161     (lambda () "- ")
2162     (lambda () "* ")
2163     (lambda () ". "))
2164   )
2165   "Functions that yield a list bullet without a number, one for each
2166 nesting depth."
2167   :group 'trex-lists
2168   :type '(list
2169          (const ignore)
2170          function
2171          function
2172          function
2173          function
2174          ))
2175
2176

```

2177 (*provide* 'trex-lists)

### 8.3 trex-markup.el

\$Id: trex-markup.el,v 1.4 2005/03/06 21:38:14 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```
2178 (require 'talcum-tools)
2179 (require 'talcum-render)
2180
2181 (defgroup trex-markup
2182   nil
2183   "Basic markup"
2184   :group 'trexes)
2185
2186 (defcustom trex-markup-macros
2187   '("textbf" "textit" "textsc"
2188     "textrm" "textsf" "texttt"
2189     "emph")
2190   "Basic markup macros. These take one argument, and that will be
2191   displayed."
2192   :group 'trex-markup
2193   :type '(repeat string))
2194
2195 (dolist (rule trex-markup-macros)
2196   (add-to-list 'talcum-render-list
2197     '((:macro ,rule "{" (:markup))
2198       :display (:rendered 0))))
2199 (talcum-make-render-map-dirty)
2200
2201 (provide 'trex-markup)
```

### 8.4 trex-math.el

\$Id: trex-math.el,v 2.6 2005/03/06 21:38:29 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```
2202 (require 'talcum-render)
2203
2204 (defgroup trex-math nil
2205   "Maths-specific rendering"
2206   :group 'trexes)
2207
2208 (defface trex-math-operator-face '((t (:weight bold :foreground "dark slate blue")))
2209   "The face math operators (like sin, cos, etc.) will be displayed in."
```

```

2210 :group 'trex-math)
2211
2212 (setq trex-math-operator-face 'trex-math-operator-face)
2213
2214 (defcustom trex-math-operators
2215   (list
2216     ;; "log-like symbols" from compr. LaTeX symbol list 08OCT2002
2217     "arccos" "arcsin" "arctan" "arg" "cos" "cosh" "cot" "coth"
2218     "csc" "deg" "det" "dim" "exp" "gcd" "hom" "inf"
2219     "ker" "lg" "lim" "liminf" "limsup" "ln" "log" "max"
2220     "min" "Pr" "sec" "sin" "sinh" "sup" "tan" "tanh")
2221   "\\log-like\\ math operators"
2222   :group 'trex-math
2223   :type '(repeat string))
2224
2225 (defcustom trex-math-render-rules
2226   '(((macro "big" "{" (:math))
2227     :property display (:height 1.2)
2228     :display 0)
2229     ((macro "bigg" "{" (:math))
2230       :property display (:height 1.3)
2231       :display 0)
2232     ((macro "Big" "{" (:math))
2233       :property display (:height 1.45)
2234       :display 0)
2235     ((macro "Bigg" "{" (:math))
2236       :property display (:height 1.6)
2237       :display 0)
2238
2239     ((macro "left" "{" (:math))
2240       :property display (:height 1.3)
2241       :display 0)
2242     ((macro "right" "{" (:math))
2243       :property display (:height 1.3)
2244       :display 0)
2245     ((macro "middle" "{" (:math))
2246       :property display (:height 1.3)
2247       :display 0)
2248
2249     ((active "\\{" nil (:math))
2250       :display "{")
2251     ((active "\\}" nil (:math))
2252       :display "}"))
2253   )
2254   "Maths-related rules"
2255   :group 'trex-math
2256   :get 'talcum-render-rule-getter
2257   :set 'talcum-render-rule-setter
2258   :initialize 'custom-initialize-default
2259   :type talcum-render-rule-custom-type)
2260
2261 (setq talcum-render-list
2262   (append talcum-render-list

```

```

2263         trex-math-render-rules
2264         (mapcar
2265         (lambda (op)
2266           '(:macro ,op nil (:math))
2267             :face trex-math-operator-face :display ,op))
2268         trex-math-operators)
2269       ))
2270 (talcum-make-render-map-dirty)
2271
2272 (provide 'trex-math)

```

## 8.5 trex-sectioning.el

\$Id: trex-sectioning.el,v 1.2 2005/03/06 22:23:10 ulmi RELEASE \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

2273 (require 'talcum-render)
2274
2275 (defvar trex-sectioning-section-commands
2276   '("part" "chapter" "section"
2277     "subsection" "subsubsection"
2278     "paragraph" "subparagraph"
2279     ))
2280
2281 (dolist (sec trex-sectioning-section-commands)
2282   (setq talcum-render-list
2283     (cons '(:macro ,sec "*[{" :sectioning)
2284       :display 2)
2285     talcum-render-list)))
2286
2287 (talcum-make-render-map-dirty)
2288
2289 (provide 'trex-sectioning)

```

## 8.6 trex-unicode.el

\$Id: trex-unicode.el,v 2.7 2005/03/06 21:38:45 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

2290 (require 'talcum-tools)
2291 (require 'talcum-render)
2292
2293 (defgroup trex-unicode nil
2294   "Rendering macros as unicode characters"
2295   :group 'trexes)

```



```

2296
2297 (defface trex-unicode-face '((t (:slant normal)))
2298   "A face known to contain many unicode glyphs.
2299
2300 This face is applied to characters by :unicode rules to prevent them
2301 from getting lost in italics , boldface, etc."
2302   :group 'trex-unicode)
2303
2304 (setq trex-unicode-face 'trex-unicode-face)
2305
2306 (defcustom trex-unicode-mappings
2307   '( ( ;; format is (macro . utf16code)
2308       ;; Unicode Math Ops 2200–22FF
2309       ;; Much still missing.
2310       ("forall" . #x2200)
2311       ("complement" . #x2201)
2312       ("partial" . #x2202)
2313       ("exists" . #x2203)
2314       ("emptyset" . #x2205)
2315       ("nabla" . #x2207)
2316       ("in" . #x2208)
2317       ("notin" . #x2209)
2318       ("ni" . #x220b)
2319       ("qedhere" . #x220e)
2320       ("prod" . #x220f)
2321       ("coprod" . #x2210)
2322       ("sum" . #x2211)
2323       ("mp" . #x2213)
2324       ("setminus" . #x2216)
2325       ("circ" . #x2218)
2326       ("cdot" . #x2219)
2327       ("sqrt" . #x221a)
2328       ("infty" . #x221e)
2329       ("land" . #x2227)
2330       ("wedge" . #x2227)
2331       ("lor" . #x2228)
2332       ("vee" . #x2228)
2333       ("cap" . #x2229)
2334       ("cup" . #x222a)
2335       ("int" . #x222b)
2336       ("iint" . #x222c)
2337       ("iiiint" . #x222d)
2338       ("neq" . #x2260)
2339       ("ne" . #x2260)
2340       ("leq" . #x2264)
2341       ("le" . #x2264)
2342       ("geq" . #x2265)
2343       ("ge" . #x2265)
2344       ("prec" . #x227a)
2345       ("succ" . #x227b)
2346       ("subset" . #x2282)
2347       ("supset" . #x2283)
2348       ("subsetq" . #x2286)

```

2349 ("supseteq" . #x2287)  
 2350 ("subsetneq" . #x228a)  
 2351 ("supsetneq" . #x228b)  
 2352 ("unlhd" . #x22b4)  
 2353 ("lhd" . #x22b2)  
 2354 ("unrhd" . #x22b5)  
 2355 ("rhd" . #x22b3)  
 2356 ;; From U27F0–U27FF Supplemental Arrows A  
 2357 ("implies" . #x27f9)  
 2358 ("iff" . #x27fa)  
 2359 ("mapsto" . #x27fc)  
 2360 ;; to is uncertain: function–goes–to is long, limit–to short  
 2361 ("to" . #x27f6)  
 2362 ("longleftarrow" . #x27f5)  
 2363 ("longrightarrow" . #x27f6)  
 2364 ("longlefttrightarrow" . #x27f7)  
 2365 ("Longleftarrow" . #x27f8)  
 2366 ("Longrightarrow" . #x27f9)  
 2367 ;; From U2190–U21FF Arrows  
 2368 ("leftarrow" . #x2190)  
 2369 ("uparrow" . #x2191)  
 2370 ("rightarrow" . #x2192)  
 2371 ("downarrow" . #x2193)  
 2372 ("leftrightharpoon" . #x2194)  
 2373 ("updownarrow" . #x2195)  
 2374 ;; many omitted...  
 2375 ;; From U2000ff General Punctuation  
 2376 ("dots" . #x2026)  
 2377 ("ldots" . #x2026)  
 2378 ("textperthousand" . #x2030)  
 2379 ;; From 27D0ff Misc Math Symb A  
 2380 ;; currently all omitted  
 2381 ;; From U2A00ff Supp Math Op  
 2382 ("bigodot" . #x2a00)  
 2383 ("bigoplus" . #x2a01)  
 2384 ("bigotimes" . #x2a02)  
 2385 ("lneq" . #x2a87)  
 2386 ("gneq" . #x2a88)  
 2387 ;;  
 2388 ("wp" . #x2118)  
 2389 ("ell" . #x2113)  
 2390 ("Im" . #x2111)  
 2391 ("Re" . #x211c)  
 2392 ("Finv" . #x2132)  
 2393 ("Game" . #x2141)  
 2394 ("aleph" . #x2135)  
 2395 ("beth" . #x2136)  
 2396 ("gimel" . #x2137)  
 2397 ("daleth" . #x2138)  
 2398 ;; Greekery. Note it is not technically correct to use  
 2399 ;; the low chars, but who has fonts that have the 1D400  
 2400 ;; Math Alnums? (Italic bold sans–serif fraktur  
 2401 ;; lowercase greek letter chi my ass!)

```

2402 ("alpha" . #x03b1)
2403 ("beta" . #x03b2)
2404 ("gamma" . #x03b3)
2405 ("delta" . #x03b4)
2406 ("epsilon" . #x03f5)
2407 ("varepsilon" . #x03b5)
2408 ("zeta" . #x03b6)
2409 ("eta" . #x03b7)
2410 ("theta" . #x03b8)
2411 ("vartheta" . #x03d1)
2412 ("iota" . #x03b9)
2413 ("kappa" . #x03ba)
2414 ("varkappa" . #x03f0)
2415 ("lambda" . #x03bb)
2416 ("mu" . #x03bc)
2417 ("nu" . #x03bd)
2418 ("xi" . #x03be)
2419 ("pi" . #x03c0)
2420 ("varpi" . #x03d6)
2421 ("rho" . #x03c1)
2422 ("varrho" . #x03f1)
2423 ("sigma" . #x03c3)
2424 ("varsigma" . #x03c2)
2425 ("tau" . #x03c4)
2426 ("upsilon" . #x03c5)
2427 ("varphi" . #x03c6)
2428 ("phi" . #x03d5)
2429 ("chi" . #x03c7)
2430 ("psi" . #x03c8)
2431 ("omega" . #x03c9)
2432 ("digamma" . #x03dd) ;;XXX is \digamma lowercase?
2433 ("Gamma" . #x0393)
2434 ("Delta" . #x0394)
2435 ("Theta" . #x0398)
2436 ("Lambda" . #x039b)
2437 ("Xi" . #x039e)
2438 ("Pi" . #x03a0)
2439 ("Sigma" . #x03a3)
2440 ("Upsilon" . #x03a5)
2441 ("Phi" . #x03a6)
2442 ("Psi" . #x03a8)
2443 ("Omega" . #x03a9)
2444 )
2445 "An alist of macro-to-utf16-code mappings."
2446 :group 'trex-unicode
2447 :type '(repeat
2448   (cons
2449     (string)
2450     (integer)))
2451 )
2452
2453 (defun trex-unicode-make-sample ()
2454   (interactive)

```

```

2455 (dolist (c trex-unicode-mappings)
2456   (insert "\\" (car c)
2457     " ~=" (decode-char 'ucs (cdr c))
2458     "\n"))
2459 (insert "\nand the number sets:\n")
2460 (dolist (c (list "C" "H" "N" "P" "Q" "R" "Z"))
2461   (insert c " ~=" (trex-unicode-mathbb (list c)) "\n"))
2462 t)
2463
2464
2465 (defun trex-unicode-init ()
2466   ;; (font-lock-add-keywords
2467   ;; nil
2468   ;; '("\[ a-zA-Z@]+\" 0 trex-unicode-face prepend))
2469   ;; 'append)
2470 t)
2471 (add-hook 'talcum-rex-init-hook 'trex-unicode-init 'atend)
2472
2473 (defun trex-unicode-display-function (spec args)
2474   (when (eq (car-safe spec) ':unicode)
2475     (propertize
2476       (char-to-string (decode-char 'ucs
2477         (cadr spec)))
2478       ':face 'trex-unicode-face
2479       'fontified t)))
2480
2481 (add-hook 'talcum-display-spec-functions
2482   'trex-unicode-display-function)
2483
2484 (dolist (c trex-unicode-mappings)
2485   (setq talcum-render-list
2486     (cons '(:macro ,(car c) nil (:unicode))
2487       :face trex-unicode-face
2488       :display (:unicode ,(cdr c)))
2489     talcum-render-list)))
2490
2491 (setq talcum-render-list
2492   (cons
2493     '(:macro "mathbb" "{" (:unicode))
2494     :display trex-unicode-mathbb)
2495   talcum-render-list))
2496
2497 (talcum-make-render-map-dirty)
2498
2499
2500 (defun trex-unicode-mathbb (args)
2501   (talcum-namespace
2502     (or
2503       (~switch (~maybe-strip-parens (car-safe args))
2504         ("C" (char-to-string (decode-char 'ucs #x2102)))
2505         ("H" (char-to-string (decode-char 'ucs #x210d)))
2506         ("N" (char-to-string (decode-char 'ucs #x2115)))
2507         ("P" (char-to-string (decode-char 'ucs #x2119)))

```

```

2508         ("Q" (char-to-string (decode-char 'ucs #x211a)))
2509         ("R" (char-to-string (decode-char 'ucs #x211d)))
2510         ("Z" (char-to-string (decode-char 'ucs #x2124))))
2511     (car-safe args)
2512     "?"))))
2513
2514 (provide 'trex-unicode)

```

## 8.7 trex-verbatim.el

\$Id: trex-verbatim.el,v 1.2 2005/03/06 21:39:00 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```

2515 (require 'talcum-render)
2516
2517 (defun trex-verbatim-parse-arg-any-repeated-delimiter (wanted)
2518   (when (equal wanted 'any-repeated)
2519     (let ((delim-char (char-after))
2520           (b (point)))
2521       (forward-char 1)
2522       (if (search-forward (char-to-string delim-char) (point-max) t)
2523           (buffer-substring b (point))
2524           (goto-char b)))))
2525
2526 (add-hook 'talcum-parse-argument-functions
2527           'trex-verbatim-parse-arg-any-repeated-delimiter)
2528
2529 (defvar trex-verbatim-render-rules
2530   '(((macro "verb" (:star any-repeated) :verbatim)
2531     :face font-lock-constant-face
2532     :display (lambda (args) (substring (cadr args) 1 -1)))
2533     (:environment "verbatim" nil :verbatim)
2534     :face font-lock-constant-face
2535     :display body)
2536   ))
2537
2538
2539 (setq talcum-render-list
2540       (append trex-verbatim-render-rules
2541               talcum-render-list))
2542 (talcum-make-render-map-dirty)
2543
2544
2545 (provide 'trex-verbatim)

```

## 8.8 trex-xref.el

\$Id: trex-xref.el,v 1.5 2005/03/06 21:39:38 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```
2546 (require 'talcum-tools)
2547 (require 'talcum-render)
2548 (require 'talcum-custom)
2549
2550 (defgroup trex-xref nil
2551   "Cross-references: index, label, cite, ref and footnote."
2552   :group 'trexes)
2553
2554 (defcustom trex-xref-rules
2555   '(((macro "label" "{" (:xref))
2556     :display "^"
2557     :help 0)
2558     ((macro "cite" "[{" (:xref))
2559       :display "[cit]"
2560       :help 1)
2561     ((macro "ref" "{" (:xref))
2562       :display "/^"
2563       :help 0)
2564     ((macro "eqref" "{" (:xref))
2565       :display "(/)"
2566       :help 0)
2567     ((macro "index" "{" (:xref))
2568       :display "[ix]"
2569       :help 0)
2570     ((macro "footnote" "{" (:xref))
2571       :display "Ž"
2572       :help 0))
2573   "Rules defined by this trex."
2574   :group 'trex-xref
2575   :type talcum-render-rule-custom-type
2576   :get 'talcum-render-rule-getter
2577   :set 'talcum-render-rule-setter
2578   :initialize 'custom-initialize-default)
2579
2580 (setq talcum-render-list (append talcum-render-list
2581                                   trex-xref-rules))
2582
2583 (talcum-make-render-map-dirty)
2584
2585 (provide 'trex-xref)
```

## 9 Mixed implementation cruft

### 9.1 talcum-compat.el

\$Id: talcum-compat.el,v 2.6 2005/02/28 17:23:37 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```
2586
2587 (require 'talcum-tools)
2588
2589 (talcum-namespace
2590  (~sourcefile-versionstamp "$RCSfile: talcum-compat.el,v $" "$Revision: 2.6 $"))
2591
2592
2593 (defun talcum-infidel-p ()
2594   "Returns t for XEmacs, nil otherwise."
2595   (string-match "XEmacs\\|Lucid" (emacs-version)))
2596
2597 (defun talcum-auctex-p ()
2598   "Returns t if AucTeX is running, nil otherwise"
2599   (and (boundp 'AUC-TeX-version) (eq major-mode 'latex-mode)))
2600
2601 (defun talcum-latex-p ()
2602   "Returns t if plain old latex mode is running, nil otherwise"
2603   (and (not (boundp 'AUC-TeX-version)) (eq major-mode 'latex-mode)))
2604
2605 (defcustom talcum-buffer-molestors
2606   '( reftex-view-crossref-when-idle
2607     reftex-view-cr-ref
2608     reftex-view-cr-cite
2609     latex-indent
2610     ;;show-paren-function ;; won't highlight rendered paren otherwise
2611   )
2612   "A list of function symbols of functions that do not set
2613   inhibit-point-motion-hooks even though they should.
2614
2615   This can lead to unwanted unrendering."
2616   :type '(repeat function))
2617
2618 (dolist (fun talcum-buffer-molestors)
2619   (message "disabling in: %s %s" fun (symbolp fun))
2620   (eval
2621    '(defadvice ,fun (around talcum-point-motion-hook-disabler activate)
2622      "Prevent point-motion-hooks in this function."
2623      (let ((inhibit-point-motion-hooks t))
2624        ad-do-it))))
2625
2626 (provide 'talcum-compat)
2627
2628 ;; talcum-compat.el ends here
```

## 9.2 talcum-tools.el

\$Id: talcum-tools.el,v 2.7 2005/08/12 16:11:31 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

```
2630
2631 (defmacro talcum-namespace_ (nspc &rest what)
2632   ;; "Make NSPC the namespace prefix in WHAT. You can use ~~ in
2633   ;; symbols and it will be replaced by NSPC. For example,
2634   ;; (talcum-namespace \"mess\" (~~age \"foo!\")) becomes (progn
2635   ;; (message \"foo!\"))."
2636   (let (rst name expr (what what))
2637
2638     (while (equal (car-safe (car-safe what)) 'macro)
2639       (setq what (macroexpand what)))
2640
2641     (while what
2642       (setq expr (pop what))
2643       (cond
2644
2645         ;; A symbol. Maybe we need to substitute.
2646         ((symbolp expr)
2647          (push (if (and (< 1 (length (setq name (symbol-name expr))))
2648                    (string= "~~" (substring name 0 2)))
2649                (intern (concat nspc (substring name 2)))
2650                expr)
2651                rst))
2652
2653         ;; A non-empty list. We have excluded the case of a macro by
2654         ;; the loop at the top. Do the car and cdr recursively and put
2655         ;; them together again. Be careful to strip the added progn and
2656         ;; list nesting by cadr. (This took a while to get right.)
2657         ((and (consp expr) (not (listp (cdr expr))))
2658          (push
2659           (let* ((x (cadr
2660                     (macroexpand '(talcum-namespace_ ,nspc ,(car expr)))))
2661                 (y (cadr
2662                     (macroexpand '(talcum-namespace_ ,nspc ,(cdr expr)))))
2663                (cons x y))
2664           rst))
2665
2666         ((consp expr)
2667          (push (cdr (macroexpand '(talcum-namespace_ ,nspc ,@expr))) rst))
2668
2669         ;; ((consp expr)
2670         ;; (push
2671         ;; (cons
2672         ;; (cadr (macroexpand '(talcum-namespace_ ,nspc ,(car expr))))
2673         ;; (cadr (macroexpand '(talcum-namespace_ ,nspc ,(cdr expr))))
2674         ;; rst))
2675
2676         ;; It's a literal or nil or something like that.
2677         ;; Just push it over.
```



```

2678      ((push expr rst))))
2679
2680      (cons 'progn (nreverse rst))))
2681
2682      ;; The default namespace for internal Talcum use is "\ talcum\ \ ".
2683      ;; Identifier names with spaces in them should be pretty rare, so
2684      ;; we will probably not collide.
2685      (defmacro talcum-namespace (&rest what)
2686        '(talcum-namespace_ " talcum " ,@what))
2687
2688      ;; We might want for some reason to disable the namespacing.
2689      (talcum-namespace
2690        (defmacro ~~no-namespace (&rest what)
2691          '(talcum-namespace_ "~~" ,@what)))
2692
2693
2694      (defcustom talcum-verbosity 5
2695        "How much going-on Talcum should send to *Messages*.
2696        0 means everything (good for debugging), 10 is silence of the lambdas."
2697        :group 'talcum
2698        :type '(integer))
2699
2700
2701      (talcum-namespace
write one save, get save-excursion, -match-data, etc. We do not actually check any semantics, or the
presence of the corresponding save-foos.
      (save (foo bar baz) bang) expands to (save-foo (save-bar (save-baz bang)))
2703      (defmacro ~~save (things &rest cmds)
2704        (let (result
2705              (things (nreverse things)))
2706          (setq result (cons (intern (concat "save-"
2707                                     (symbol-name (pop things))))
2708                            cmds))
2709          (dolist (thing things result)
2710            (setq result (list (intern (concat "save-"
2711                                               (symbol-name thing)))
2712                              result)))))
2713
2714
2715
2716      (defmacro ~~append! (l &rest ls)
2717        "Append all args and setq the first to the result."
2718        '(setq ,l (append ,l ,@ls)))
2719
2720      (defmacro ~~point-after (fu)
2721        "Execute FU and return the (possibly new) value of point."
2722        '(progn ,fu (point)))
2723      );namespace
2724      (talcum-namespace
2725        (defmacro ~~without-touching-buffer (&rest cmds)
2726          "Do something without touching the buffer, i.e. it is read-only.
2727          Just because we're extra paranoid and this is probably not very expensive."

```

```

2728   '(let ((buffer-read-only t)
2729         (inhibit-read-only nil)
2730         (inhibit-redisplay t)
2731         (inhibit-point-motion-hooks t)
2732         (inhibit-modification-hooks t))
2733     ,@cmds))
2734
2735 (defmacro ~with-secretly-touching-buffer (&rest cmds)
2736   "Modify the current buffer, but do not change read-only or
2737   buffer-modified flags. This is for the rare situation where you want
2738   to override without-touching-buffer."
2739   '(let ((inhibit-read-only t)
2740         (deactivate-mark nil)
2741         (was-virgin (buffer-modified-p)))
2742     (progn (progn ,@cmds) (set-buffer-modified-p was-virgin))))
2743   );namespace
2744 (talcum-namespace
2745
2746 (defun ~scantoken (&optional kind)
2747   "Move point forward by a token, and return the token. Whitespace
2748   after the token is skipped, just as (La)TeX does. Optional parameter
2749   KIND can be 'cmd or 'env; if 'env, we actually skip a whole \begin{foo}."
2750   (let ((bot (point))
2751         (eot nil)
2752         (forward-sexp-function nil) ;;latex-mode thinks it's clever. It's not.
2753         )
2754
2755     (skip-chars-forward "\t")
2756     (if (eq kind 'env)
2757         (~scantoken 'cmd) ; get the "\begin"
2758     (setq eot
2759         (cond
2760          ((looking-at "\\\\") ; a control sequence proper
2761           (forward-char 1)
2762           (if (not (looking-at "[a-zA-Z]")) ;; self-terminating: \$ etc.
2763               (~point-after (forward-char 1))
2764               (skip-chars-forward "a-zA-Z") ;; proper command. skip trailing blanks.
2765           (progn
2766              (point)
2767              (skip-chars-forward "\t")))))
2768          ((looking-at "{")
2769           (condition-case nil
2770             (~point-after
2771              (forward-sexp))
2772             (scan-error (setq eot (1+ bot))))
2773           )) ;FIXME
2774     (t
2775      (~point-after (forward-char 1))))))
2776
2777   (buffer-substring bot eot)))
2778
2779 (defun ~fwd-arg ()
2780   "Go forward until a semi-properly nested exp has been moved over.

```

```

2781 This means there may be spurious non-matching closing parens."
2782 (catch 'fail
2783   (let (stack)
2784     (skip-syntax-forward "^(")
2785     (push (char-after) stack)
2786     (forward-char 1)
2787     (while stack
2788       (skip-syntax-forward "^()")
2789       (if (looking-at "\\s(")
2790         (push (char-after) stack)
2791         (if (equal (char-after)
2792                   (cdr (aref (syntax-table) (car stack))))
2793           (pop stack)
2794           (if (eobp) (throw 'fail nil))))
2795       (forward-char 1))
2796     t)))
2797 );namespace

```

## Multimaps

```

2798
2799 (talcum-namespace
2800 ;; create and return a new multimap.
2801 (defun ~mmmap-new ()
2802   (make-hash-table :test 'equal))
2803
2804 ;; put a key-value entry into a multimap.
2805 ;; This does not remove current entries in the multimap,
2806 ;; even if they have the same key.
2807 (defun ~mmmap-put (key val mmap)
2808   (puthash key (cons val (gethash key mmap nil)) mmap))
2809
2810 ;; look up all entries for a key in a multimap.
2811 ;; returns nil if none present.
2812 (defun ~mmmap-get (key mmap)
2813   (gethash key mmap nil))
2814
2815 (defun ~mmmap-keys (mmap)
2816   (let (result)
2817     (maphash (lambda (key val) (push key result)) mmap)
2818     result))
2819 ); namespace

```

## Scoped Variables.

```

2821
2822 (talcum-namespace
2823 (defun ~var-scope-start (name)
2824   (let ((stack (get name 'talcum-varstack)))
2825     (if (boundp name)
2826         (push (symbol-value name) stack))
2827         ;(make-local-variable name)) nope, render-string opens tempbuffer
2828     (put name 'talcum-varstack stack)
2829     (set name nil)))
2830

```

```

2831 (defun ~~var-scope-end (name)
2832   (let ((stack (get name 'talcum-varstack)))
2833     (put name 'talcum-varstack (cdr-safe stack))
2834     (if (null stack)
2835         (makunbound name)
2836         (set name (car-safe stack)))))
2837
2838 (defun talcum-var-history (name)
2839   (let ((stack (get name 'talcum-varstack)))
2840     (and (boundp name)
2841          (cons (symbol-value name) stack))))
2842
2843 );namespace

```

Generic tools.

```

2844
2845 (talcum-namespace
2846 (defun ~~filter (p xs)
2847   (let ((r nil))
2848     (if (not (listp xs))
2849         (error "Trying to filter %S which is not a list" xs)
2850         (if (not (functionp p))
2851             (error "Trying to filter by %S which is not a function" p)
2852             (while xs
2853                 (let ((x (car xs))
2854                     (xss (cdr xs)))
2855                   (setq r (if (funcall p x) (cons x r)
2856                               r))
2857                   (setq xs xss)))
2858             (nreverse r)))))
2859
2860 (defun ~~debug (severity &rest args)
2861   "Do debugging output, depending on talcum-verbosity."
2862   (if (>= severity talcum-verbosity)
2863       (apply 'message args)))
2864
2865
2866 (defun ~~sourcefile-versionstamp (id rev)
2867   (setq talcum--versions
2868         (cons (list id rev)
2869               (if (boundp 'talcum--versions)
2870                   talcum--versions
2871                   nil))))
2872
2873
2874 (defmacro ~~switch (expr &rest choices)
2875   "Cond template. Each element of CHOICES is compared against the value of EXPR in a cond."
2876   (let ((temp (make-symbol "--switchvar--")))
2877     `(let* ((,temp ,expr))
2878       ,(cons 'cond
2879              (mapcar
2880               (lambda (choice)
2881                 (if (consp choice)

```

```

2882         '(((equal ,temp ,(car choice))
2883           ,(cadr choice))
2884         '(t ,choice)))
2885     choices))))))
2886
2887 (defmacro ~value-safe (symbol)
2888   (if (boundp symbol) symbol
2889       nil))
2890
2891
2892 (~sourcefile-versionstamp "$RCSfile: talcum-tools.el,v $" "$Revision: 2.7 $")
2893
2894 ); namespace
2895
2896 (provide 'talcum-tools)
2897
talcum-tools.el end here

```

### 9.3 talcum-custom.el

\$Id: talcum-custom.el,v 1.4 2005/02/19 18:13:59 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, [ulmiusers.sarovar.org](mailto:ulmiusers.sarovar.org).

```

2898
2899 (require 'talcum-tools)
2900
2901 (defconst talcum-render-rule-custom-type
2902   '(repeat                                     ; many rules
2903     (list :tag "rule"                         ; one rule
2904       (list :tag "pattern part"               ; the pattern part
2905         (choice :tag "type"                   ; the type part
2906           :value :macro
2907           (const :value :macro)
2908           (const :value :environment)
2909           (const :value :active))
2910         (string :tag "name"
2911           :value "textbf") ; the name
2912         (choice :tag "argspec"                 ; the argspec
2913           :value "*{"
2914           (string :tag "short form"
2915             :value "*{"") ; short argspec
2916           (repeat :tag "long form"
2917             (symbol :value :mandatory) ; long argspec
2918             ))
2919         (choice :tag "featspec" ; featspec

```

```

2925 ;; FIXME: featspec needs extra attention to disambiguate
2926 ;; feature keywords and function results; likewise for
2927 ;; "never" rule and nested lists ? Or drop nil rule altogether?
2928 ;; OTOH, empty lists aren't really meaningful here
2929
2930 (const :tag "always"
2931       :value t) ;-- success
2932 (const :tag "never"
2933       :value nil) ;-- failure
2934 (integer :tag "render-level"
2935         :value 3) ;-- render-thres
2936 (list :tag "feature"
2937      (const :value --feature)
2938      (symbol :tag "feature"
2939             :value ":trex-my-feature")
2940      ) ; -- feature
2941 (list :tag "ask predicate"
2942      (const :value --function)
2943      (function :tag "ask predicate"
2944               :value ignore)) ;-- function
2945 (repeat :tag "all of the following" ;-- recursive featspec
2946        (choice :tag "all of the following"
2947                (const :tag "always"
2948                      :value t) ;-- success
2949                (const :tag "never" ;-- failure
2950                      :value nil)
2951                (integer :tag "render-level"
2952                      :value 3) ;-- render-thres
2953                (list :tag "feature"
2954                      (const :value --feature)
2955                      (symbol :tag "feature"
2956                             :value ":trex-my-feature")
2957                      ) ;-- feature
2958                (list :tag "ask predicate"
2959                      (const :value --function)
2960                      (function :tag "ask predicate"
2961                              :value ignore)) ;-- function
2962                (repeat :tag "any of the following" ;-- recursive featspec ii
2963                       (choice
2964                        (const :tag "always" t) ;--- success
2965                        (const :tag "never" nil) ;--- failure
2966                        (integer :tag "render-level"
2967                                :value 3) ;--- render-thres
2968                        (list :tag "feature"
2969                              (const :value --feature)
2970                              (symbol :tag "feature"
2971                                     :value ":trex-my-feature")
2972                              ) ;--- feature
2973                        (list :tag "ask predicate"
2974                              (const :value --function)
2975                              (function :tag "ask predicate"
2976                                      :value ignore)) ;--- function
2976                        (repeat :tag "deeper nesting (freeform)"
2977

```

```

2978                                     (sexp))))
2979                                 ))
2980                             ))
2981
2982 (repeat                               ;the action part
2983   :tag "action part"
2984   :value (:display "*"))
2985
2986 (choice
2987   :value (:display "*"))
2988
2989 (list :tag "display"
2990       :value (:display "*"))
2991
2992 (const :value :display)
2993 (choice                               ; proper display flavour
2994   :value "*")
2995
2996 (string :tag "fixed text"
2997         :value "*")
2998 (integer :tag "(n+1)th argument"
2999         :value 0)
3000 (const :tag "body of environment"
3001       :value body)
3002 (list :tag "(rendering recursively)"
3003       (const :tag ":rendered"
3004             :value :rendered)
3005       (choice                               ; recursive display flavour
3006         :value "*")
3007       (string :tag "fixed text"
3008             :value "*")
3009       (integer :tag "(n+1)th argument"
3010             :value 0)
3011       (const :tag "body of environment"
3012             :value body body)
3013       (function :tag "result of function"
3014                 :value ignore)))
3015 (function :tag "result of function"
3016         :value ignore)))
3017
3018 (list :tag "evaluate anything"
3019       :value nil
3020       (const :value :eval)
3021       (sexp))
3022
3023 (list :tag "force face"
3024       (const :value :face) ; face action
3025       (face))
3026
3027 (list :tag "fly-over help"
3028       (const :value :help)
3029       (choice                               ; display flavour
3030         (string :tag "fixed text"

```

```

3031         :value "*")
3032     (integer :tag "(n+1)th argument"
3033       :value 0)
3034     (const :tag "body of environment"
3035       :value body body)
3036     (function)))
3037
3038   (list :tag "render-opener"
3039     (const
3040       :value :opener)
3041     (function
3042       :value ignore))
3043
3044   (list :tag "property"
3045     (const
3046       :value :property)
3047     (symbol :tag "name"
3048       :value prop)
3049     (sexp :tag "value"
3050       :value nil))
3051
3052   (list :tag "local variable"
3053     :value my-local-var
3054     (const :value :var)
3055     (choice
3056       (variable :tag "name"
3057         :value my-local-var)
3058       (cons :tag "with default value"
3059         (variable :tag "name"
3060           :value my-local-var)
3061         (sexp :tag "value"
3062           :value ""))))))
3063   ))
3064 ))
3065 "A custom-compatible semi-representation of rules. Only works in
3066 conjunction with the special getter talcum-render-rule-getter and
3067 setter talcum-render-rule-setter, which see."
3068 )
3069
3070
3071 ;; Custom vars for rules should look like this:
3072 ;; (defcustom foorules nil "" :get 'talcum-render-rule-getter :set 'talcum-render-rule-setter :type talcum-render-rule-custom
3073
3074 (defconst talcum-render-actions-arities
3075   '((:display . 1)
3076     (:eval . 1)
3077     (:face . 1)
3078     (:help . 1)
3079     (:opener . 1)
3080     (:property . 2)
3081     (:var . 1))
3082 "Alist that gives the number of arguments for each display flavour.
3083

```



```

3084 Not keeping this list up-to-date will prevent customize from working
3085 for unlisted flavours." )

    Circumvent limitations of customization interface.
    Here be dragons! At least, -getter isn't one sexp with nested mapcar lambdas anymore.

3087
3088 (talcum-namespace
3089
3090 ;; FIXME: this should set dirty flag?
3091 (defun talcum-render-rule-setter (nam val)
3092   "Customize-setter for render rule lists ."
3093   (set nam
3094     (mapcar (lambda (l) (cons (~render-rule-pattern-setter (car l))
3095                               (apply 'append
3096                                     (apply 'append (cdr l))))))
3097     val)))
3098
3099 (defun talcum-render-rule-getter (nam)
3100   "Customize-getter for render rule lists ."
3101   (mapcar (quote ~render-rule-getter1)
3102     (symbol-value nam)))
3103
3104 (defun ~render-rule-getter1 (rule)
3105   (let ((rule rule)
3106         pattern
3107         newactions)
3108     (setq pattern (~render-rule-pattern-getter (pop rule))) ; pattern part
3109     ;; now treat the action part
3110     (while rule
3111       (let ((action nil)
3112             i)
3113         (push (pop rule) action) ;; action name
3114         (dotimes (i (cdr (assoc (car action)
3115                                talcum-render-actions-arities)))
3116           (push (pop rule) action))
3117         (push (nreverse action) newactions)))
3118     (list pattern (nreverse newactions))))
3119
3120 (defun ~render-rule-pattern-getter (pattern)
3121   (list (nth 0 pattern)
3122         (nth 1 pattern)
3123         (nth 2 pattern)
3124         (~render-rule-pattern-getter1 (nth 3 pattern))))
3125
3126 (defun ~render-rule-pattern-getter1 (pattern)
3127   (cond
3128     ((functionp pattern) (list '--function pattern))
3129     ((listp pattern) (mapcar (quote ~render-rule-pattern-getter1)
3130                             pattern))
3131     ((keywordp pattern) (list '--feature pattern))
3132     (t pattern)))
3133
3134 (defun ~render-rule-pattern-setter (pattern)

```

```

3135 (list (nth 0 pattern)
3136       (nth 1 pattern)
3137       (nth 2 pattern)
3138       (~render-rule-pattern-setter1 (nth 3 pattern))))
3139
3140 (defun ~render-rule-pattern-setter1 (pattern)
3141   (cond
3142     ((listp pattern)
3143      (if (member (car pattern) '--feature --function))
3144          (cadr pattern)
3145          (mapcar (quote ~render-rule-pattern-setter1)
3146                  pattern)))
3147     (t pattern)))
3148 ) ; namespace
3149
3150 (talcum-namespace
3151  (~sourcefile-versionstamp "$RCSfile: talcum-custom.el,v $" "$Revision: 1.4 $"))
3152
3153 (provide 'talcum-custom)

```

## 9.4 talcum-menu.el

\$Id: talcum-menu.el,v 1.5 2005/08/26 18:37:41 ulmi Exp \$

This file is part of Talcum.

Talcum is released under the terms and conditions of the LaTeX Project Public License 1.3a (see file LPPL-13a for details)

Talcum has the maintenance status: author-maintained. Maintainer: Ulrich M. Schwarz, ulmiusers.sarovar.org.

Finally: a clickable menu.

The entries are generally runtime-generated. This would not be necessary for all kinds of menu, but for some (think chapter selection), and I think it should not pose a big run-time overhead.

```

3155
3156 (require 'talcum-tools)
3157 (require 'talcum-custom)
3158
3159 (talcum-namespace
3160  (~sourcefile-versionstamp "$RCSfile: talcum-menu.el,v $" "$Revision: 1.5 $"))
3161
3162 (talcum-namespace
3163
3164  (defmacro ~negate (var)
3165    '(setq ,var (not ,var)))

```

Some common menu helpers

A checkbox for a variable.

```

3168 (defmacro ~menu-flag-toggle (var str &optional more)
3169   '[,str
3170    (talcum-namespace (~negate ,var) ,more)
3171    :style toggle
3172    :selected ,var

```

```

3173   ])
3174
3175   A radio field. Names come from the cars, values come from the cadrs.
3176
3177   (defmacro ~menu-choose-one (var names-and-vals)
3178     `(mapcar
3179       (lambda (nav)
3180         (vector (car nav)
3181                 (list 'setq ',var (cadr nav))
3182                 :style 'radio
3183                 :selected (list 'equal ',var (cadr nav)))))
3184       ,names-and-vals))
3185
3186   (defvar talcum-menu-functions nil
3187     "A list of functions. Each function is called with no arguments.
3188     The result should be a menu item definition. All results are
3189     concatenated and used as the Talcum menu.")
3190
3191   ;; Not sure if this will be needed. Menus are strange, unclear if
3192   ;; it's safe to change talcum-menu's value directly.
3193   (defun talcum-menu-regenerate ()
3194     (interactive)
3195     (easy-menu-define talcum-menu talcum-mode-map
3196       "Talcum's menu."
3197       (cons "Talcum"
3198             (apply 'append
3199                    (mapcar 'funcall talcum-menu-functions)))))
3200   t)
3201
3202   (talcum-menu-regenerate)
3203
3204   ) ; namespace
3205   (provide 'talcum-menu)

```

## Index

~~action-part, 1349, **1949**  
~~append, 1328, 1526, **2716**  
~~at-suitable-place, **279**, 288, 320, 351, 373  
~~backslash, **1940**, 1944, 1970, 1972  
~~debug, 108–110, 116, 1083, 1094, 1098, 1115, 1117, 1124, 1976, **2860**  
~~eval-display-spec, 1402, 1408, 1420, **1439**  
~~filter, 624, 627, 729, **2846**  
~~find-end-of-environment, 1322, **1587**  
~~fwd-arg, 269, 270, 274, **2779**  
~~match-any-re, **1196**, 1258, 1754  
~~maybe-apply-rule-at-point, 1272, **1289**  
~~maybe-strip-parens, 1449, **1578**, 2107, 2503  
~~menu-choose-one, 888, 928, 964, **3175**  
~~menu-flag-toggle, 502, 1074, 1889, **3168**  
~~mmap-get, 1262, **2812**  
~~mmap-keys, **2815**  
~~mmap-new, 1198, 1743, **2801**  
~~mmap-put, 1752, **2807**  
~~mode-syntax-table, 1206, **1637**, 1781, 1782  
~~negate, 947, 995, 1014, **3164**, 3170  
~~newfoo-find-suitable-place, **220**, 282  
~~newfoo-find-suitable-position, 246, **248**, 374  
~~no-namespaces, **2690**  
~~not-after-backslash, **1943**  
~~nth-filtered, 595, 604, 613, **623**  
~~parse-arguments, 1308, **1510**  
~~pattern-args, 1293, **1956**  
~~pattern-feat, **1958**, 1995  
~~pattern-matcher, 1748, **1961**  
~~pattern-name, 1323, **1954**, 1966  
~~pattern-part, 1292, 1748, 1749, **1947**, 1995  
~~pattern-type, 1294, 1319, 1749, **1952**, 1967  
~~pikeman-all-chapters, **791**, 796, 803, 810, 818, 865  
~~pikeman-compiler-options, **723**, 746, 779  
~~pikeman-draftmode, **961**, 964, 969, 978  
~~pikeman-draftmode-menu, **962**, 984  
~~pikeman-find-includes, 797, **837**  
~~pikeman-first-line, **735**, 747, 782  
~~pikeman-format, **923**, 928, 932, 933  
~~pikeman-format-menu, **925**, 938  
~~pikeman-format-option, **931**, 936  
~~pikeman-generate-menu, 711, **715**  
~~pikeman-include-menu, **794**, 877  
~~pikeman-interaction-menu, **886**, 902  
~~pikeman-interaction-mode, **884**, 888, 896, 897  
~~pikeman-interaction-option, **895**, 900  
~~pikeman-listfiles, **943**, 947, 949, 951  
~~pikeman-listfiles-menu, **944**, 954  
~~pikeman-make-draftmode, **968**, 986  
~~pikeman-make-includeonly, **863**, 875  
~~pikeman-make-listfiles, **950**, 956  
~~pikeman-make-menu, 822  
~~pikeman-make-nonembed, **998**, 1005  
~~pikeman-menu, 822  
~~pikeman-menu-item, 810, **825**  
~~pikeman-nonembed-fonts, **991**, 995, 997, 1000  
~~pikeman-nonembed-menu, **992**, 1003  
~~pikeman-prepare-compiler-string, **743**, 772  
~~pikeman-selected-chapters, **790**, 814, 818, 828–831, 834, 866, 869  
~~pikeman-srcspc, **1010**, 1014, 1016, 1018  
~~pikeman-srcspc-menu, **1011**, 1024  
~~pikeman-srcspc-option, **1017**, 1022  
~~point-after, 1562, **2720**, 2763, 2770, 2775  
~~pseudolength, 518, **626**  
~~really-apply-rule-at-point, 1334, **1348**  
~~render-open/close, **1480**  
~~render-rule-pattern-getter, 3108, **3120**  
~~render-rule-pattern-setter, 3094, **3134**  
~~rendermap, **1198**, 1230, 1264, 1743, 1752, 1764, 1768  
~~rule-desired-p, 1304, 1747, **1989**  
~~rule-desired-p\_, 1995, **2004**, 2016  
~~sanitized-mode-syntax-table, 1535, 1543, **1638**, 1782, 1788  
~~save, 1235, 1270, **2703**  
~~scantoken, 1536, **2746**, 2757  
~~sourcefile-versionstamp, 42, 135, 207, 456, 686, 1164, 1181, 1626, 1937, 2590, **2866**, 2892, 3151, 3160  
~~sparsed, **542**, 571  
~~switch, 2503, **2874**  
~~tabdance-hijack, 556, **559**  
~~user-point, 1238, 1239, 1277, 1313–1315, **1714**, 1715, 1726, 1727, 1834  
~~value-safe, 573, 574, 751, 752, 764, 1238, 1313, 1726, **2887**  
~~var-scope-end, 1435, **2831**  
~~var-scope-start, 1374, 1375, **2823**  
~~with-secretly-touching-buffer, 1387, 1400, 1464, 1482, 1802, **2735**  
~~without-touching-buffer, 1236, 1481, 1801, 1816, 1824, **2725**  
~~work-prod-queue, **1079**, 1129, 1138, 1147  
-, 1405, 2076  
-feature, 2937, 2954, 2969, 3143  
-function, 2942, 2959, 2974, 3143  
:active, 1906, 2057, 2249, 2251, 2910

:any-repeated, 2530  
 :at-end, 197  
 :desc, 2092, 2128  
 :display, 2058, 2095, 2198, 2228, 2231, 2234, 2237,  
     2241, 2244, 2247, 2250, 2252, 2267, 2284,  
     2488, 2494, 2532, 2535, 2556, 2559, 2562,  
     2565, 2568, 2571, 2984, 2987, 2990, 2992,  
     3075  
 :enum, 2086, 2121  
 :environment, 2085, 2088, 2091, 2533, 2909  
 :eval, 3020, 3076  
 :face, 2267, 2487, 2531, 2534, 3024, 3077  
 :filter, 717  
 :foreground, 2208  
 :german, 2057  
 :get, 1647, 2256, 2576  
 :group, 50, 68, 88, 395, 408, 413, 419, 435, 461,  
     465, 473, 483, 489, 495, 688, 916, 1036,  
     1040, 1056, 1065, 1630, 1642, 1646, 1696,  
     1705, 2138, 2149, 2167, 2184, 2192, 2206,  
     2210, 2222, 2255, 2295, 2302, 2446, 2552,  
     2574, 2697  
 :height, 2227, 2230, 2233, 2236, 2240, 2243, 2246  
 :help, 2557, 2560, 2563, 2566, 2569, 2572, 3028,  
     3078  
 :included, 1897, 1899, 1901, 1903  
 :initialize, 1649, 2258, 2578  
 :item, 2089, 2125  
 :key-sequence, 832  
 :lists, 2085, 2088, 2091, 2094  
 :macro, 2094, 2197, 2226, 2229, 2232, 2235, 2239,  
     2242, 2245, 2266, 2283, 2486, 2493, 2530,  
     2555, 2558, 2561, 2564, 2567, 2570, 2907,  
     2908  
 :mandatory, 2921  
 :markup, 1701, 2197  
 :math, 1699, 2226, 2229, 2232, 2235, 2239, 2242,  
     2245, 2249, 2251, 2266  
 :opener, 3040, 3079  
 :package, 261, 374  
 :property, 1184, 1186, 1187, 2227, 2230, 2233, 2236,  
     2240, 2243, 2246, 3046, 3080  
 :recurse, 2087, 2090, 2093  
 :rendered, 2198, 3004  
 :sectioning, 1703, 2283  
 :selected, 834, 949, 997, 1016, 1915, 3172, 3181  
 :set, 1648, 2257, 2577  
 :slant, 2297  
 :star, 2530  
 :style, 833, 948, 996, 1015, 1914, 3171, 3180  
 :tag, 409, 420, 422–426, 436, 438, 439, 918–921,  
     2903, 2904, 2906, 2912, 2915, 2917, 2920,  
     2923, 2930, 2932, 2934, 2936, 2938, 2941,  
     2943, 2945–2947, 2949, 2951, 2953, 2955,  
     2958, 2960, 2962, 2964–2966, 2968, 2970,  
     2973, 2975, 2977, 2983, 2989, 2996, 2998,  
     3000, 3002, 3003, 3007, 3009, 3011, 3013,  
     3015, 3018, 3023, 3027, 3030, 3032, 3034,  
     3038, 3044, 3047, 3049, 3052, 3056, 3058,  
     3059, 3061  
 :test, 2802  
 :type, 69, 89, 409, 414, 421, 437, 466, 474, 484,  
     490, 496, 917, 1041, 1057, 1066, 1631,  
     1650, 1706, 2150, 2168, 2193, 2223, 2259,  
     2447, 2575, 2616, 2698  
 :unicode, 1700, 2486, 2488, 2493  
 :value, 425, 426, 438, 439, 1059, 1060, 1066, 2907–  
     2910, 2913, 2916, 2918, 2921, 2931, 2933,  
     2935, 2937, 2939, 2942, 2944, 2948, 2950,  
     2952, 2954, 2956, 2959, 2961, 2967, 2969,  
     2971, 2974, 2976, 2984, 2987, 2990, 2992,  
     2994, 2997, 2999, 3001, 3004, 3006, 3008,  
     3010, 3012, 3014, 3016, 3019, 3020, 3024,  
     3028, 3031, 3033, 3035, 3040, 3042, 3046,  
     3048, 3050, 3053, 3054, 3057, 3060, 3062  
 :var, 2086, 2089, 2092, 3054, 3081  
 :verbatim, 2530, 2533  
 :weight, 2208  
 :xref, 1702, 2555, 2558, 2561, 2564, 2567, 2570  
     \_, 588, 717, 1469  
     \_\_, 1469  
  
 a, 1470, 1472, 1474, 1601, 1610, 1993, 1994  
 action, 1353, 1365, 1369, 1371, 1380, 1383, 1386,  
     1395, 1398, 1411, 1416, 1422, 3111, 3113,  
     3114, 3116, 3117  
 actions, 1349, 1363, 1365, 1372, 1381, 1384, 1391,  
     1396, 1402, 1408, 1418, 1420  
 activate, 632, 771, 1723, 2621  
 ad-activate, 1140, 1149  
 ad-disable-advice, 1156, 1160  
 ad-do-it, 674, 2624  
 ad-enable-advice, 1139, 1148  
 ad-get-arg, 1147  
 add-hook, 442, 443, 676, 1169, 1170, 1922, 1923,  
     2471, 2481, 2526  
 add-text-properties, 1388, 1467  
 add-to-list, 23, 33, 34, 195, 215, 510, 545, 713, 777,  
     780, 874, 876, 899, 901, 935, 937, 953,  
     955, 983, 985, 1002, 1004, 1021, 1023,  
     1077, 1919, 2034, 2196  
 after, 1136, 1145  
 all-completions, 602, 607, 620  
 allp, 594, 600, 609, 618  
 append, 62, 572, 807, 1349, 1518, 1745, 1906, 2083,  
     2262, 2540, 2580, 2718

apply, 317, 718, 2863, 3095, 3096, 3196  
 aref, 2792  
 argnumber, 300, 315, 318, 322, 326, 327, 333, 344,  
 353, 356, 357, 362  
 args, 1290, 1307, 1328, 1334, 1348, 1402, 1409,  
 1413, 1421, 2473, 2500, 2503, 2511, 2532,  
 2860, 2863  
 argspec, 1293, 1308, 1510, 1518  
 around, 630, 2621  
 assoc, 372, 3114  
 AucTeX-version, 181  
 autoload, 44  
  
 b, 1993, 1994, 2520, 2523, 2524  
 before, 771, 1723  
 beg, 306, 308, 311, 1465, 1468, 1483, 1489, 1491,  
 1504, 1506  
 blockname, 568, 582, 583  
 body, 308, 328, 1413, 1414, 2535, 3001, 3012, 3035  
 bodypos, 1451–1453  
 bold, 2208  
 bor, 1226, 1234, 1241, 1250, 1255, 1797, 1803,  
 1810  
 bos, 1835, 1837, 1839  
 bot, 2750, 2772, 2777  
 boundp, 16, 28, 54, 173, 233, 517, 527, 776, 798,  
 2102, 2599, 2603, 2825, 2840, 2869, 2888  
 bu, 841, 843, 845, 971, 972  
 buffer, 839, 851, 860  
 buffer-modified-p, 2741  
 buffer-modified-tick, 1111  
 buffer-or-file, 837, 839, 840, 844, 859  
 buffer-read-only, 243, 2728  
 buffer-string, 1212  
 buffer-substring, 1452, 1553, 1561, 2523, 2777  
 bufferp, 839  
  
 c, 2455–2457, 2460, 2461, 2484, 2486, 2488  
 cadr, 1414, 1442, 1453, 2118, 2123, 2477, 2532,  
 2659, 2661, 2883, 3144, 3179, 3181  
 call-interactively, 537  
 car, 304, 348, 650, 666, 1085, 1086, 1351, 1359,  
 1375–1377, 1401, 1405, 1413, 1414, 1451,  
 1452, 1773, 1948, 2057, 2117, 2456, 2486,  
 2660, 2792, 2853, 2882, 3094, 3114, 3143,  
 3178  
 car-safe, 1441, 2110, 2115, 2474, 2503, 2511, 2638,  
 2836  
 catch, 1516, 1542, 1598, 1871, 2005, 2782  
 category, 1184  
 cdr, 651, 1087, 1351, 1359, 1376, 1404, 1774, 1950,  
 2058, 2457, 2488, 2657, 2662, 2667, 2792,  
 2854, 3096, 3114  
  
 cdr-safe, 372, 2833  
 ch, 1785, 1788, 2055, 2057, 2058  
 chapter, 825  
 char-after, 1524, 2519, 2785, 2790, 2791  
 char-to-string, 2067, 2476, 2504–2510, 2522  
 choice, 421, 919, 1059, 1060, 1066, 2880–2883,  
 2906, 2915, 2923, 2946, 2963, 2986, 2993,  
 3005, 3029, 3055  
 choices, 2874, 2885  
 chord, 476, 478  
 cmds, 279, 2703, 2708, 2725, 2735  
 compilation-in-progress, 1113  
 completing-read, 569, 629  
 compose-mail, 139  
 concat, 23, 34, 110, 140, 172, 377, 749, 867, 897,  
 933, 1262, 1600, 1944, 1970, 1972, 2075,  
 2077, 2142–2145, 2649, 2706, 2710  
 condition-case, 2769  
 cons, 96, 438, 831, 887, 926, 963, 1335, 1391, 2057,  
 2283, 2448, 2486, 2492, 2663, 2680, 2706,  
 2808, 2841, 2855, 2868, 2878, 3058, 3094,  
 3195  
 consp, 649, 665, 1373, 2007, 2657, 2666, 2881  
 const, 422–424, 921, 1059, 2151, 2169, 2908–2910,  
 2930, 2932, 2937, 2942, 2947, 2949, 2954,  
 2959, 2964, 2965, 2969, 2974, 2992, 3000,  
 3003, 3011, 3020, 3024, 3028, 3034, 3039,  
 3045, 3054  
 copy-marker, 1250, 1298  
 copy-syntax-table, 1781, 1782  
 current-buffer, 239, 846, 1498  
  
 deactivate-mark, 312, 2740  
 decode-char, 2457, 2476, 2504–2510  
 def, 631  
 defadvice, 629, 771, 1136, 1145, 1722, 2621  
 defalias, 189, 591  
 defconst, 1, 84, 1843, 1940, 2901, 3074  
 defcustom, 60, 86, 397, 411, 417, 430, 463, 468,  
 480, 486, 492, 908, 1038, 1045, 1063,  
 1628, 1644, 1698, 2140, 2158, 2186, 2214,  
 2225, 2306, 2554, 2605, 2694  
 defface, 2208, 2297  
 defgroup, 47, 395, 458, 688, 1036, 1640, 1693, 2136,  
 2181, 2204, 2293, 2550  
 define-key, 389–391, 551, 1883, 1884  
 define-minor-mode, 91  
 define-prefix-command, 79  
 define-skeleton, 565  
 defmacro, 279, 2631, 2685, 2690, 2703, 2716, 2720,  
 2725, 2735, 2874, 2887, 3164, 3168, 3175  
 delete-and-extract-region, 308  
 delete-overlay, 1361, 1431, 1502, 1809

delete-region, 1403  
 delim-char, 2519, 2522  
 delq, 2039  
 depth, 1599, 1603, 1608, 1609  
 direction, 533, 538  
 disj, 1989, 1995, 2004, 2006, 2010, 2016, 2020  
 display, 2227, 2230, 2233, 2236, 2240, 2243, 2246  
 docu, 1079, 1084, 1090, 1113, 1123–1125, 1129  
 dolist, 115, 259, 544, 1268, 1359, 1434, 1518, 1745,  
     1772, 1774, 1785, 1907, 2008, **2055**, 2114,  
     2195, 2281, 2455, 2460, 2484, 2618, 2709  
 dothis-p, 1085, 1092, 1093  
 dotimes, 3114  
 dowhat, 1086, 1094, 1097–1100, 1117, 1122, 1124–  
     1126  
  
 easy-menu-define, 3193  
 emacs-version, 178, 2595  
 end, 307, 308, 1466, 1468, 1484, 1489, 1491, 1504,  
     1506, 1510, 1519, 1523, 1547, 1551, 1587,  
     1604, 1870, 1878, 1879  
 endofpattern, 1298, 1300, 1309, 1315, 1326, 1332,  
     1336  
 enter-or-leave, 1480, 1486, 1494  
 envlimit, 1291, 1321, 1324, 1328  
 envname, 338, 353, 355, 362, 1587, 1600  
 eobp, 1247, 2794  
 eor, 1226, 1234, 1244, 1245, 1259, 1272, 1797, 1803,  
     1810  
 eos, 1836, 1837, 1839  
 eot, 2751, 2758, 2772, 2777  
 eq, 261, 305, 349, 1371, 1380, 1383, 1386, 1395,  
     1398, 1411, 1416, 1441, 1450, 1498, 1969,  
     1971, 1973, 2012, 2013, 2020, 2474, 2599,  
     2603, 2756  
 equal, 600, 618, 650, 1092, 1294, 1319, 1486, 1494,  
     1533, 1534, 1539, 1540, 1558, 1559, 1749,  
     1808, 2115, 2121, 2125, 2128, 2518, 2638,  
     2791, 2882  
 error, 1126, 1455, 2018, 2849, 2851  
 eval, 1381, 2620  
 evaporate, 1186  
 excursion, 1235, 1270  
 expr, 2636, 2642, 2646, 2647, 2650, 2657, 2660,  
     2662, 2666, 2678, 2874  
  
 face, 1805, 3025  
 feat, 115–117, 1774, 1775, 2008, 2012–2018, 2029,  
     2030, 2032, 2034, 2036, 2039  
 file, 863, 895, 931, 950, 968, 973, 998, 1017  
 file-name-directory, 19  
 filifun, 739  
 final-point, 1250, 1266, 1274, 1275, 1278  
  
 find-file-noselect, 242  
 firstline, 739, 741, 747, 763  
 flag, 53, 54  
 font-lock-add-keywords, 1779  
 font-lock-constant-face, 2531, 2534  
 font-lock-fontify-region, 1489  
 font-lock-remove-keywords, 1793  
 fontified, 1806  
 format, 174, 313, 323, 327, 354, 357, 570, 977,  
     1408, 1420, 1574, 1755  
 forward-char, 265, 1248, 1562, 1872, 2521, 2761,  
     2763, 2775, 2786, 2795  
 forward-line, 256, 275, 329, 358  
 forward-sexp, 1549, 2771  
 forward-sexp-function, 1545, 2752  
 found, 1517, 1524, 1525, 1541, 1560, 1575  
 fu, 2720  
 fun, 2618, 2619  
 funcall, 63–65, 231, 720, 730, 739, 1093, 1099,  
     1429, 1454, 2010, 2015, 2122, 2126, 2855  
 function, 426, 1059, 1060, 2152–2155, 2170–2173,  
     2616, 2943, 2960, 2975, 3013, 3015, 3036,  
     3041  
 functionp, 230, 1093, 1097, 1454, 2015, 2850, 3128  
  
 generate-new-buffer, 841, 971  
 get, 1230, 2824, 2832, 2839  
 get-text-property, 1392  
 gethash, 2808, 2813  
 goodbuffer, 222, 241, 242  
 goto-char, 254, 852, 974, 1245, 1255, 1275, 1300,  
     1401, 2524  
  
 hist, 631, 655, 665–667  
  
 i, 2066–2068, 2071, 2073, 2074, 2076, 2078, 2080,  
     2142–2145, 3112, 3114  
 id, 2866, 2868  
 identity, 806  
 ignore, 1048, 1059, 1060, 2151, 2159, 2169, 2944,  
     2961, 2976, 3014, 3016, 3042  
 inherit, 631  
 inhibit-modification-hooks, 2732  
 inhibit-point-motion-hooks, 848, 1850, 1860, 2623,  
     2731  
 inhibit-read-only, 2729, 2739  
 inhibit-redisplay, 849, 2730  
 initial, 631  
 insert, 144, 171, 313, 323, 354, 375, 1207, 1402,  
     1863, 2456, 2459, 2461  
 insert-file-contents, 844, 973  
 integer, 1066, 2450, 2698, 2934, 2951, 2966, 2998,  
     3009, 3032

integerp, 303, 347, 1449, 2014  
 interactive, 138, 287, 298, 342, 367, 477, 516, 526,  
 536, 838, 1227, 1738, 1799, 1833, 1847,  
 1859, 2025, 2033, 2037, 2454, 3192  
 interactive-p, 519, 529  
 intern, 2649, 2706, 2710  
  
 jit-lock-fontify-now, 1722  
  
 key, 2807, 2808, 2812, 2813, 2817  
 keywordp, 2017, 3131  
 kill-buffer, 860  
 kind, 2746, 2756  
  
 l, 2716, 3094, 3096  
 lambda, 53, 170, 426, 676, 717, 720, 730, 739, 779,  
 782, 1046, 1469, 1807, 1993, 1994, 2142–  
 2145, 2160–2163, 2265, 2532, 2817, 2880,  
 3094, 3177  
 last, 1413, 1451  
 latex-block-default, 570, 580, 582  
 latex-block-names, 575  
 latex-indent, 2609  
 latex-run-command, 749  
 latex-standard-block-names, 574  
 length, 265, 376, 627, 865, 866, 2647  
 limit, 1865, 1866  
 list, 56, 95, 192, 210, 501, 656, 710, 716, 778, 781,  
 887, 918, 926, 945, 963, 993, 1012, 1058,  
 1073, 1328, 1390, 1492, 1505, 1527, 1610,  
 1888, 2007, 2117, 2141, 2150, 2168, 2215,  
 2460, 2461, 2710, 2868, 2903, 2904, 2936,  
 2941, 2953, 2958, 2968, 2973, 2989, 3002,  
 3018, 3023, 3027, 3038, 3044, 3052, 3118,  
 3121, 3128, 3131, 3135, 3179, 3181  
 list-type, 2110, 2115, 2121, 2125, 2128  
 listp, 304, 348, 1838, 2016, 2657, 2848, 3129, 3142  
 localvars, 1350, 1378, 1434  
 locate-library, 14  
 looking-at, 266, 271, 1546, 2760, 2762, 2768, 2789  
 lower-frame, 1049  
 ls, 2716  
 lv, 1434, 1435  
  
 macname, 293, 314, 322, 325, 333  
 macparams, 2100, 2105, 2107  
 macro-matchers, 1741, 1750, 1756, 1757  
 macroexpand, 2639, 2660, 2662, 2667  
 major-mode, 112, 2599, 2603  
 make-hash-table, 2802  
 make-list, 318  
 make-overlay, 1351  
 make-symbol, 2876  
 make-variable-buffer-local, 1715  
  
 makunbound, 2835  
 mapc, 169, 1807  
 mapcar, 571, 659, 719, 730, 810, 2264, 2879, 3094,  
 3101, 3129, 3145, 3176, 3197  
 mapconcat, 728, 739, 868  
 maphash, 2817  
 mark-active, 306, 307  
 marker-position, 1336  
 MATCH, 1294, 1297, 1298  
 match-beginning, 1297  
 match-data, 1270  
 match-end, 1266, 1298  
 match-string, 979, 980, 1262, 1263, 1584, 1607  
 match-string-no-properties, 856  
 matcher, 1741, 1748, 1750–1752  
 matching-rules, 1253, 1261, 1268  
 max, 1739  
 max-specpdl-size, 1739  
 mcf, 720  
 member, 619, 829, 834, 3143  
 memq, 2030  
 message, 31, 289, 330, 359, 520, 530, 549, 1422,  
 1763, 1777, 1878, 2026, 2619  
 min, 1403  
 minibuffer-local-completion-map, 551  
 mmap, 2807, 2808, 2812, 2813, 2815, 2817  
 modify-syntax-entry, 1788  
 more, 3168  
 move-marker, 1266, 1274  
 my-local-var, 3053, 3057, 3060  
  
 n, 623, 624, 2114, 2115  
 nam, 3091, 3093, 3099, 3102  
 name, 1966, 1970, 1972, 1974, 1976, 2636, 2647–  
 2649, 2823–2826, 2828, 2829, 2831–2833,  
 2835, 2836, 2838–2841  
 names-and-vals, 3175  
 nav, 3177–3179, 3181  
 nest-depth, 2111, 2113, 2114, 2116, 2122, 2126  
 new-end-or-nil, 1252, 1265, 1269, 1271, 1273, 1274  
 newactions, 3107, 3117, 3118  
 no-display-flag, 1354, 1412, 1428, 1430  
 normal, 2297  
 nreverse, 857, 1907, 2680, 2705, 2858, 3117, 3118  
 nspc, 2631, 2649  
 nth, 624, 1085, 1086, 1449, 1953, 1955, 1957, 1959,  
 2078, 2105, 2107, 2122, 2126, 3121–3124,  
 3135–3138  
 number, 1066  
 number-to-string, 2066  
  
 o, 1471, 1472, 1474, 1602  
 op, 2265



optfun, 730  
 options, 366, 368, 369, 376, 377, 727, 733, 746, 754  
 other-matchers, 1741, 1751, 1759, 1760  
 ov, 1351, 1369, 1389, 1392, 1396, 1407, 1419, 1425,  
     1426, 1429, 1431, 1462, 1465, 1466, 1480,  
     1483, 1484, 1487, 1495–1500, 1502, 1807–  
     1809  
 overlay-buffer, 1498  
 overlay-end, 1389, 1466, 1471, 1484, 1497, 1500  
 overlay-get, 1360, 1425, 1495, 1808  
 overlay-put, 1396, 1407, 1419, 1426, 1487  
 overlay-start, 1389, 1392, 1465, 1470, 1483, 1496,  
     1499  
 overlays-in, 1359, 1810  
  
 p, 1851, 2846, 2850, 2851, 2855  
 package, 366, 372, 379  
 pattern, 1292–1294, 1319, 1323, 1952–1959, 1961,  
     1966, 1967, 3106, 3108, 3118, 3120–3124,  
     **3126**, 3128–3132, 3134–3138, **3140**, 3142–  
     3144, 3146, 3147  
 patternargs, 1439, 1447, 1449, 1451, 1454  
 patternpos, 1348, 1351, 1359, 1401, 1404, 1405,  
     1429  
 pfx, 293, 302–305, 315, 338, 346–349, 1829, 1838  
 point, 1239, 1249, 1309, 1326, 1403, 1499, 1500,  
     1519, 1544, 1547, 1551, 1554, 1562, 1601,  
     1613, 1727, 1819, 1827, 1834, 1851, 1866,  
     2520, 2523, 2722, 2750, 2766  
 point-entered, 1469, 1803  
 point-left, 1804  
 point-max, 855, 1208, 1406, 1794, 1826, 1855, 1856,  
     1875, 1877, 1896, 1900, 2522  
 point-min, 254, 852, 974, 1208, 1794, 1818, 1853,  
     1854, 1896, 1900  
 pop, 1365, 1372, 1381, 1384, 1391, 1396, 1402,  
     1408, 1418, 1420, 2642, 2707, 2793, 3108,  
     3113, 3116  
 pos, 1462  
 pred, 594, 602, 607, 611, 616, 620, 623, 624  
 predicate, 631  
 priority, 1187  
 proc, 1088  
 progn, 1114, 1890, 1909, 2722, 2742  
 prompt, 630  
 prop, 3048  
 propertize, 2475  
 provide, 127, 199, 445, 679, 1028, 1173, 1618, 1925,  
     2043, 2063, 2177, 2201, 2272, 2289, 2514,  
     2545, 2585, 2626, 2896, 3153, 3203  
 push, 856, 1378, 1750, 1751, 1908, 2647, 2658,  
     2667, 2678, 2785, 2790, 2817, 2826, 3113,  
     3116, 3117  
  
 put, 1764, 1768, 2828, 2833  
 puthash, 2808  
  
 quote, 552, 810, 834, 875, 877, 900, 902, 936, 938,  
     954, 956, 984, 986, 1003, 1005, 1022,  
     1024, 1129, 1230, 1715, 1764, 1768, 3101,  
     3129, 3145  
  
 r, 2847, 2855, 2856, 2858  
 re, 1600, 1604, 1943, 1944  
 read-from-minibuffer, 370  
 realchoice, 595, 602, 604, 607, 611, 613, 616, 620  
 recursive-edit, 290, 331, 360  
 reftex-view-cr-cite, 2608  
 reftex-view-cr-ref, 2607  
 reftex-view-crossref-when-idle, 2606  
 regexp-opt, 1757, 1760  
 regexp-quote, 1944  
 region-beginning, 306, 1898, 1902  
 region-end, 307, 1898, 1902  
 remove, 830  
 remove-text-properties, 1490, 1503, 1803  
 repeat, 69, 409, 437, 917, 1057, 2193, 2223, 2447,  
     2616, 2902, 2920, 2945, 2962, 2977, 2982  
 reqmatch, 631  
 require, 37–39, 102, 103, 117, 118, 131, 132, 201–  
     203, 450–453, 682, 683, 1031–1034, 1176–  
     1178, 1620–1623, 1773, 1934, 2044, 2064,  
     2178, 2179, 2202, 2273, 2290, 2291, 2515,  
     2546–2548, 2587, 2899, 3156, 3157  
 restjobs, 1087, 1116, 1127, 1129  
 restriction, 1235  
 result, 1440, 1444, 1458, 1459, 1517, 1521, 1526,  
     2072, 2075, 2077, 2704, 2706, 2709, 2710,  
     2712, 2816–2818  
 rev, 2866, 2868  
 rst, 847, 856, 857, 861, 2636, 2651, 2664, 2667,  
     2678, 2680  
 rule, 1268, 1272, 1289, 1292, 1304, 1334, 1348,  
     1349, 1745, 1747–1749, 1752, 1947–1950,  
     1989, 1995, 2195, **3104**, 3105, 3108, 3110,  
     3113, 3116  
 run-at-time, 1128  
 run-hook-with-args, 321, 332, 352, 361  
 run-hook-with-args-until-success, 1368, 1446, 1530  
 run-hooks, 119, 123, 1778  
  
 s, 1544, 1554  
 save-buffer, 291, 334, 363  
 save-excursion, 280, 842, 850, 970, 1244, 1333, 1611,  
     1800, 1817, 1825, 1853, 1855, 1861, 1870  
 save-window-excursion, 281  
 scan-error, 2772

scopelimit, 1289, 1308, 1323  
 search-backward, 260, 264, 1612, 1818  
 search-backward-regexp, 1853  
 search-forward, 255, 1550, 1826, 2522  
 search-forward-regexp, 853, 975, 1257, 1604, 1855, 1873  
 sec, 2281  
 set, 1376, 2829, 2836, 3093  
 set-buffer, 843, 851, 972  
 set-buffer-modified-p, 2742  
 set-marker, 1278, 1309, 1326  
 severity, 2860, 2862  
 sexp, 2978, 3021, 3049, 3061  
 sit-for, 380  
 skip-chars-forward, 2755, 2764, 2767  
 skip-syntax-forward, 268, 273, 1246, 1523, 2784, 2788  
 spec, 1439, 1441–1443, 1447–1450, 1454, 1455, 2473, 2474, 2477  
 speclist, 2004, 2007, 2008  
 stack, 2783, 2785, 2787, 2790, 2792, 2793, 2824, 2826, 2828, 2832–2834, 2836, 2839, 2841  
 standard-latex-block-names, 573  
 start, 1869, 1878, 1879  
 startofpattern, 1297, 1314, 1335  
 str, 587, 589, 594, 602, 607, 611, 616, 619, 620, 1200, 1202, 1203, 1207, 1578, 1582, 1584, 1585, 3168  
 string, 409, 425, 438, 439, 918, 920, 1060, 2193, 2223, 2449, 2912, 2917, 2996, 3007, 3030  
 string-match, 21, 1046, 1582, 2595  
 stringp, 228, 234, 241, 840, 859, 1100, 1307, 1324, 1448  
 subj, 137, 143  
 subov, 1359–1361  
 substring, 2532, 2648, 2649  
 switch-to-buffer, 242  
 sym, 53, 56, 170, 172–174  
 symbol, 69, 2887, 2888, 2921, 2938, 2955, 2970, 3047  
 symbol-name, 172, 1908, 2647, 2707, 2711  
 symbol-value, 54, 174, 664, 2826, 2841, 3102  
 symbolp, 2619, 2646  
 syntax-table, 2792  
 system-configuration, 180  
 system-type, 179  
  
 tabdance, 630  
 tabdance-choices, 518, 596, 605, 614, 633, 651, 656  
 tabdance-level, 518, 520, 521, 528, 530, 531, 595, 604, 613, 635  
 table, 630, 634, 649–652, 668, 669  
 talcum, 47  
 talcum–versions, 183, 2867, 2870  
 talcum-add-render-feature, 1775, 1912, **2032**  
 talcum-apply-action-functions, **1654**  
 talcum-auctex-p, 109, 237, 800, 1090, 1113, 1123, 1144, 1159, **2597**  
 talcum-buffer-molestors, **2605**, 2618  
 talcum-bug-generate-menu, **191**  
 talcum-default-actions, **1183**, 1349  
 talcum-definitions-file, 221, 228–231, **417**  
 talcum-desired-features, **60**, 115, 185  
 talcum-display-spec-functions, **1663**  
 talcum-enter-mode-hook, **76**  
 talcum-excuse-to-preamble, **286**  
 talcum-has-render-feature, 1910, 1915, 2017, **2029**  
 talcum-infidel-p, **2593**  
 talcum-init-specpdl, **1684**, 1740  
 talcum-insert-newcommand, **293**  
 talcum-insert-newenvironment, **338**  
 talcum-insert-usepackage, **366**  
 talcum-keymap, 389–391, 1883, 1884  
 talcum-latex-p, 108, 554, 1089, 1108, 1121, 1135, 1155, **2601**  
 talcum-leave-mode-hook, **77**  
 talcum-lighter, **84**, 94  
 talcum-make-render-map-dirty, **1767**, 2061, 2097, 2199, 2270, 2287, 2497, 2542, 2583  
 talcum-make-render-opener, 1352, 1384, 1429, **1462**  
 talcum-menu, 3193  
 talcum-menu-functions, **3184**, 3197  
 talcum-menu-regenerate, 120, **3191**, 3200  
 talcum-mode, 91, 105, 122, 1083, 1084  
 talcum-mode-chord, **86**, 96  
 talcum-mode-map, 3193  
 talcum-namespace, 41, 106, 134, 206, 455, 498, 685, 690, 787, 813, 817, 821, 827, 834, 882, 906, 942, 947, 949, 960, 990, 995, 997, 1009, 1014, 1016, 1070, 1180, 1195, 1463, 1625, 1636, 1708, 1721, 1730, 1886, 1936, 1939, 2107, 2501, 2589, **2685**, 2689, 2701, 2724, 2744, 2799, 2822, 2845, 3088, 3150, 3159, 3162, 3170  
 talcum-namespace\_, **2631**, 2660, 2662, 2667, 2686, 2691  
 talcum-newcmd, 395  
 talcum-newcmd-exit, **386**  
 talcum-newcmd-generate-menu, **209**  
 talcum-newcmd-init, **384**  
 talcum-newfoo-after-insertion-hooks, **218**  
 talcum-newfoo-before-insertion-hooks, **217**  
 talcum-newpar-and-render, **1858**  
 talcum-package-option-hints, 372, **430**  
 talcum-parse-argument-functions, **1674**  
 talcum-path, 19, 23, 30, 31, 33, 34, 184

talcum-pikeman, 66, 688, 771  
 talcum-pikeman-compile-filestart-functions, **698**, 740  
 talcum-pikeman-compile-options-functions, **692**, 731  
 talcum-pikeman-formats, **908**, 929  
 talcum-pikeman-generate-menu, **709**  
 talcum-pikeman-menu-creator-functions, **704**, 721  
 talcum-point-motion-hook-disabler, 2621  
 talcum-prod, 1036  
 talcum-prod-commands, **1045**, 1138, 1147  
 talcum-prod-cycle, **1063**, 1128  
 talcum-prod-exit, **1153**  
 talcum-prod-generate-menu, **1072**  
 talcum-prod-init, **1132**  
 talcum-prod-tick, **1068**, 1109, 1110  
 talcum-prod-viewer, 1136  
 talcum-prod-viewer-AucTeX, 1145  
 talcum-remove-render-feature, 1911, **2036**  
 talcum-render, 1640  
 talcum-render-actions-arities, **3074**, 3115  
 talcum-render-dwim, **1829**, 1894  
 talcum-render-exit, **1791**, 1893  
 talcum-render-features, **1652**, 1907, **2024**, 2026, 2027, 2030, 2038, 2039  
 talcum-render-find-suitable-end, **1821**, 1836  
 talcum-render-find-suitable-start, **1813**, 1835  
 talcum-render-fontf-function, **1869**  
 talcum-render-fontlock-function, 1779, 1793, **1865**  
 talcum-render-init, **1770**, 1892  
 talcum-render-list, **1633**, 1745, 2056, 2059, 2082, 2084, 2261, 2262, 2282, 2285, 2485, 2489, 2491, 2495, 2539, 2541, 2580  
 talcum-render-make-menu, **1887**  
 talcum-render-paragraph, **1846**, 1862  
 talcum-render-parsep-re, **1843**, 1853, 1855, 1874  
 talcum-render-region, 1208, **1226**, 1414, 1506, 1839, 1852, 1866, 1879, 1896, 1898  
 talcum-render-reinit, 1231, **1737**, 1780, 1913  
 talcum-render-rule-custom-type, 1650, 2259, 2575, **2901**  
 talcum-render-rule-getter, **3099**  
 talcum-render-rule-setter, **3091**  
 talcum-render-string, **1200**, 1458  
 talcum-render-threshold, 2014  
 talcum-render-trexes, **1698**, 1772  
 talcum-rendered, 1185  
 talcum-report-bug, **137**  
 talcum-rex-init-hook, **1191**  
 talcum-semantic-braces, **1688**, 1785  
 talcum-tabdance, 66, 458, **594**  
 talcum-tabdance-auto-escalate, **486**, 601, 610  
 talcum-tabdance-auto-prefer-history, **480**, 655  
 talcum-tabdance-descalate, **523**  
 talcum-tabdance-escalate, **513**, 603, 612  
 talcum-tabdance-generate-menu, **500**  
 talcum-tabdance-init, **547**, 677  
 talcum-tabdance-key, **468**, 478, 548, 552, 641  
 talcum-tabdance-no-hijacking, **492**, 555  
 talcum-tabdance-query-key, **476**  
 talcum-tabdance-scalate, **533**, 552  
 talcum-textinfo-commands, 259, **397**  
 talcum-unrender-region, 1474, 1794, **1797**, 1837, 1900, 1902  
 talcum-use-newcmd-flag, **411**  
 talcum-use-prod-flag, **1038**, 1074, 1080  
 talcum-use-render-flag, 1233, **1628**, 1889, 1891, 1901, 1903, 1906  
 talcum-use-tabdance-flag, **463**, 502, 640, 676  
 talcum-user-point-setter, 1723  
 talcum-user-render-list, **1644**, 1746  
 talcum-var-history, 2114, **2838**  
 talcum-verbosity, **2694**, 2862  
 talcum-version, **1**, 141, 182  
 temp, 2876  
 TeX-active-master, 238, 779, 782, 801  
 tex-command, 748  
 tex-dvi-view-command, 1046  
 tex-file, 771, 1136  
 tex-latex-block, 565  
 tex-main-file, 234, 235, 745, 799  
 TeX-process, 1090  
 TeX-run-command, 1125  
 TeX-run-format, 1145  
 tex-send-command, 1122  
 tex-shell-buf, 1111  
 tex-shell-proc, 1089  
 tex-start-commands, 764  
 tex-start-options, 752  
 tex-start-options-string, 751  
 texfile, 723, 730, 735, 739, 745–747, 766  
 textinfo, 259, 260  
 thing, 2709, 2711  
 things, 2703, 2705, 2707, 2709  
 thisfile, 14, 16, 19, 21  
 throw, 1520, 1525, 1548, 1552, 1573, 1605, 2021, 2794  
 toggle, 833, 948, 996, 1015, 1914, 3171  
 transient-mark-mode, 306, 307  
 trex, 1772–1774, 1907, 1908  
 trex-lists, 2136  
 trex-lists-Alph, **2068**, 2145  
 trex-lists-alph, **2067**, 2068, 2143  
 trex-lists-arabic, **2066**, 2142  
 trex-lists-enum-style, 2122, **2140**  
 trex-lists-item-style, 2126, **2158**  
 trex-lists-nesting, 2086, 2089, 2092, 2110, 2117, 2118, 2123

trex-lists-Roman, **2080**  
 trex-lists-roman, **2071**, 2080, 2144  
 trex-lists-spitzmarke, 2095, **2100**  
 trex-markup, 1701, 2181  
 trex-markup-macros, **2186**, 2195  
 trex-math, 1699, 2204  
 trex-math-operator-face, 2208, 2212, 2267  
 trex-math-operators, **2214**, 2268  
 trex-math-render-rules, **2225**, 2263  
 trex-sectioning, 1703  
 trex-sectioning-section-commands, **2275**, 2281  
 trex-unicode, 1700, 2293  
 trex-unicode-display-function, **2473**  
 trex-unicode-face, 2297, 2304, 2487  
 trex-unicode-init, **2465**  
 trex-unicode-make-sample, **2453**  
 trex-unicode-mappings, **2306**, 2455, 2484  
 trex-unicode-mathbb, 2461, 2494, **2500**  
 trex-verbatim-parse-arg-any-repeated-delimiter, **2517**  
 trex-verbatim-render-rules, **2529**, 2540  
 trex-xref, 1702, 2550  
 trex-xref-rules, **2554**, 2581  
 trexes, 1693  
 trexmenu, 1904, 1907, 1917  
 try-completion, 611, 616  
 type, 248, 261, 1967, 1969, 1971, 1973, 1976  
  
 unless-forbidden, 52, 63–65  
 upcase, 2068, 2080  
  
 val, 2807, 2808, 2817, 3091, 3097  
 var, 3164, 3168, 3175  
 variable, 3056, 3059  
 vc-toggle-read-only, 244  
 vector, 86, 3178  
  
 wanted, 1518, 1531, 1533, 1534, 1539, 1540, 1558,  
     1559, 2517, 2518  
 was-virgin, 2741, 2742  
 what, 2631, 2636, 2638, 2639, 2641, 2642, 2685,  
     2690  
 whatnext, 1079, 1085–1087, 1116  
 while, 601, 610, 853, 1257, 1363, 1546, 1603, 2074,  
     2638, 2641, 2787, 2852, 3110  
 with-syntax-table, 1206, 1535, 1543  
 with-temp-buffer, 1205  
 with-temp-message, 1744  
  
 x, 544, 545, 2659, 2663, 2853, 2855  
 xs, 542, 544, 623, 624, 626, 627, 2846, 2848, 2849,  
     2852–2854, 2857  
 xss, 2854, 2857  
  
 y, 2661, 2663  
  
 ys, 543, 544  
 zerop, 1603