

The **texnegar** package
Kashida justification in LuaTeX and XeTeX
Source code documentation

Hossein Movahhedian*

Released 2021-02-09 v0.1e

Negar: *Negar, in Persian, is the present stem of negaashtan meaning to design; to paint; to write; and as a noun it means “sweetheart, idol, beloved, figuratively refering to a beautiful woman, pattern, painting, and artistic design”*

*E-mail: dma8hm1334@gmail.com

Contents

1	TEXNegar Implementation	3
1.1	File: texnegar.sty	3
1.2	File: texnegar-luatex.sty	3
1.3	File: texnegar-xetex.sty	4
1.4	File: texnegar-ini.tex	4
1.5	File: texnegar-common-kashida.tex	12
1.6	File: texnegar-xetex-kashida.tex	14
1.7	File: texnegar-char-table.lua	17
1.8	File: texnegar.lua	23
1.9	File: texnegar-ini.lua	25
1.10	File: texnegar-luatex-kashida.lua	26
2	Acknowledgments	37
3	Change History	38
	[2020-08-29 v0.1a]	38
	[2020-08-30 v0.1b]	38
	[2021-01-27 v0.1c]	38
	Index	39

1 T_EXNegar Implementation

1.1 File: texnegar.sty

```
1 <*texnegar-sty>
2 \RequirePackage{xparse}
3 \RequirePackage{l3keys2e}
4 \RequirePackage{graphicx}[2019-11-30]
5 \RequirePackage{array}[2019-10-01]
6 \RequirePackage[dvipsnames,svgnames,x11names]{xcolor}[2016/05/11]
7 \RequirePackage{fontspec}[2020/02/21]
8 \RequirePackage{newverbs}[2010/09/02]
9 \RequirePackage{environ}[2014/05/04]
10
11 \ProvidesExplPackage {texnegar} {2021-02-09} {0.1e} { Full implementation of kashida feature
12
13 \sys_if_engine_luatex:T
14 {
15   \RequirePackageWithOptions{texnegar-luatex}
16   \endinput
17 }
18 \sys_if_engine_xetex:T
19 {
20   \RequirePackageWithOptions{texnegar-xetex}
21   \endinput
22 }
23 \msg_new:nnn {texnegar} {cannot-use-pdftex}
24 {
25   The~ texnegar~ package~ requires~ either~ XeTeX~ or~ LuaTeX.\\\
26   You~ must~ change~ your~ typesetting~ engine~ to,~ e.g.,~
27   "xelatex"~ or~ "lualatex"~ instead~ of~ "latex"~ or~ "pdflatex".
28 }
29 \msg_fatal:nm {texnegar} {cannot-use-pdftex}
30
31 \endinput
32 </texnegar-sty>
```

1.2 File: texnegar-luatex.sty

```
33 <*texnegar-luatex-sty>
34 \ProvidesExplPackage {texnegar-luatex} {2021-02-09} {0.1e} { Full implementation of kashida
35
36 \tex_input:D { texnegar-ini.tex }
37
38 \bool_if:NT \l_texnegar_kashida_fix_bool
39 {
40   \if_int_compare:w \luatexversion < \c_texnegar_luatexversionmajormin_int\c_texnegar_luat
41     \msg_error:nxxx { texnegar } { luatex-version-is-too-old } { !!!! } { \c_texnegar_l
42   \fi:
43
44   \hbox_set:Nn \l_texnegar_k_box { \resizebox{5000sp}{\height}{-} }
45
46   \hbox_set:Nn \l_texnegar_ksh_box { \char\lua_now:n { tex.sprint(0, font.getfont(font.cur
47
48   \directlua{dofile(kpse.find_file("texnegar.lua"))}
```

```

49 }
50
51 \bool_if:NT \l_texnegar_kashida_fix_bool
52 {
53   \tex_input:D { texnegar-common-kashida.tex }
54
55   \AtBeginDocument
56   {
57     \KashidaOn
58   }
59 }
60
61 \endinput
62 </texnegar-luatex-sty>

```

1.3 File: texnegar-xetex.sty

```

63 <*texnegar-xetex-sty>
64 \RequirePackage{zref-savepos}[2020-03-03]
65 \ProvidesExplPackage {texnegar-xetex} {2021-02-09} {0.1e} { Full implementation of kashida f
66
67 \tex_input:D { texnegar-ini.tex }
68
69 \bool_if:NT \l_texnegar_kashida_fix_bool
70 {
71   \tex_input:D { texnegar-xetex-kashida.tex }
72 }
73
74 \endinput
75 </texnegar-xetex-sty>

```

1.4 File: texnegar-ini.tex

```

76 <*texnegar-ini-tex>
77 \ProvidesExplFile {texnegar-ini.tex} {2021-02-09} {0.1e} { Full implementation of kashida fe
78
79 \def\TeXNegar{\TeX Negar}
80
81 \box_new:N \l_texnegar_k_box
82 \box_new:N \l_texnegar_ksh_box
83
84 \tl_const:Nn \c_texnegar_luatexversionmajormin_int {1}
85 \tl_const:Nn \c_texnegar_luatexversionminormin_int {12}
86
87 \int_const:Nn \c_texnegar_ksh_int {"0640} % kashida
88 \int_const:Nn \c_texnegar_lrm_int {"200E} % left-right-mark
89 \int_const:Nn \c_texnegar_zwj_int {"200D} % zero-width joiner
90
91 \int_const:Nn \c_texnegar_two_int {2}
92 \int_const:Nn \c_texnegar_four_int {4}
93
94 \tl_const:Nn \c_texnegar_skip_a_tl { 0 em plus 0.5 em }
95 \tl_const:Nn \c_texnegar_skip_b_tl { 0.14 em plus 5.5 em }
96
97 \int_new:N \l_texnegar_counter_int
98

```

```

99 \int_new:N \l_texnegar_kashida_slot_int
100
101 \int_new:N \l_texnegar_line_break_penalty_int
102
103 \int_new:N \l_texnegar_min_penalty_int
104 \int_new:N \l_texnegar_low_penalty_int
105 \int_new:N \l_texnegar_med_penalty_int
106 \int_new:N \l_texnegar_high_penalty_int
107 \int_new:N \l_texnegar_max_penalty_int
108
109 \int_new:N \l_fontnumber_int
110
111 \tl_new:N \l_texnegar_line_break_tl
112
113 \tl_new:N \l_texnegar_main_font_full_tl
114 \tl_new:N \l_texnegar_main_font_name_tl
115
116 \tl_new:N \l_texnegar_font_full_tl
117 \tl_new:N \l_texnegar_font_name_tl
118
119 \tl_new:N \l_texnegar_skip_default_tl
120
121 \tl_new:N \l_texnegar_active_ligs_tl
122
123 \tl_new:N \l_texnegar_gap_filler_tl
124
125 \tl_new:N \l_texnegar_use_color_tl
126 \tl_new:N \l_texnegar_color_tl
127 \tl_new:N \l_texnegar_color_rgb_tl
128
129 \dim_new:N \l_texnegar_diff_pos_dim
130
131 \bool_set_false:N \l_texnegar_minimal_bool
132 \tl_set:Nn \l_texnegar_minimal_off_tl { Off }
133 \tl_set:Nn \l_texnegar_minimal_on_tl { On }
134
135 \bool_set_false:N \l_texnegar_kashida_fix_bool
136
137 \bool_set_false:N \l_texnegar_kashida_fontfamily_bool
138 \tl_new:N \l_texnegar_kashida_fontfamily_tl
139 \tl_set:Nn \l_texnegar_kashida_fontfamily_tl { N/A }
140
141 \bool_set_false:N \l_texnegar_kashida_glyph_bool
142 \bool_set_false:N \l_texnegar_kashida_leaders_glyph_bool
143 \bool_set_false:N \l_texnegar_kashida_leaders_hrulerule_bool
144
145 \bool_set_false:N \l_texnegar_ligature_bool
146 \bool_set_false:N \l_texnegar_linebreakpenalty_bool
147 \bool_set_false:N \l_texnegar_hboxrecursion_bool
148 \bool_set_false:N \l_texnegar_vboxrecursion_bool
149 \bool_set_false:N \l_texnegar_color_bool
150
151 \int_set:Nn \l_texnegar_min_penalty_int { 0 }
152 \int_set:Nn \l_texnegar_low_penalty_int { 8 }

```

```

153 \int_set:Nn \l_texnegar_med_penalty_int { 15 }
154 \int_set:Nn \l_texnegar_high_penalty_int { 25 }
155 \int_set:Nn \l_texnegar_max_penalty_int { 10000 }
156
157 \tl_set:Nn \l_texnegar_stretch_glyph_tl { glyph }
158 \tl_set:Nn \l_texnegar_stretch_leaders_glyph_tl { leaders+glyph }
159 \tl_set:Nn \l_texnegar_stretch_leaders_hruler_tl { leaders+hruler }
160 \tl_set:Nn \l_texnegar_stretch_off_tl { Off }
161 \tl_set:Nn \l_texnegar_stretch_on_tl { On }
162
163 \tl_set:Nn \l_texnegar_hboxrecursion_off_tl { Off }
164 \tl_set:Nn \l_texnegar_hboxrecursion_on_tl { On }
165
166 \tl_set:Nn \l_texnegar_vboxrecursion_off_tl { Off }
167 \tl_set:Nn \l_texnegar_vboxrecursion_on_tl { On }
168
169 \tl_set:Nn \l_texnegar_fnt_kayhan_tl { kayhan }
170 \tl_set:Nn \l_texnegar_fnt_kayhannavaar_tl { kayhannavaar }
171 \tl_set:Nn \l_texnegar_fnt_kayhanpook_tl { kayhanpook }
172 \tl_set:Nn \l_texnegar_fnt_kayhansayeh_tl { kayhansayeh }
173 \tl_set:Nn \l_texnegar_fnt_khoramshahr_tl { khoramshahr }
174 \tl_set:Nn \l_texnegar_fnt_khorramshahr_tl { khorrasmshahr }
175 \tl_set:Nn \l_texnegar_fnt_niloofar_tl { niloofar }
176 \tl_set:Nn \l_texnegar_fnt_paatch_tl { paatch }
177 \tl_set:Nn \l_texnegar_fnt_riyaz_tl { riyaz }
178 \tl_set:Nn \l_texnegar_fnt_roya_tl { roya }
179 \tl_set:Nn \l_texnegar_fnt_shafigh_tl { shafigh }
180 \tl_set:Nn \l_texnegar_fnt_shafighKurd_tl { shafighKurd }
181 \tl_set:Nn \l_texnegar_fnt_shafighUzbek_tl { shafighUzbek }
182 \tl_set:Nn \l_texnegar_fnt_shiraz_tl { shiraz }
183 \tl_set:Nn \l_texnegar_fnt_sols_tl { sols }
184 \tl_set:Nn \l_texnegar_fnt_tabriz_tl { tabriz }
185 \tl_set:Nn \l_texnegar_fnt_titr_tl { titr }
186 \tl_set:Nn \l_texnegar_fnt_titre_tl { titre }
187 \tl_set:Nn \l_texnegar_fnt_traffic_tl { traffic }
188 \tl_set:Nn \l_texnegar_fnt_vahid_tl { vahid }
189 \tl_set:Nn \l_texnegar_fnt_vosta_tl { vosta }
190 \tl_set:Nn \l_texnegar_fnt_yaghut_tl { yaghut }
191 \tl_set:Nn \l_texnegar_fnt_yagut_tl { yagut }
192 \tl_set:Nn \l_texnegar_fnt_yas_tl { yas }
193 \tl_set:Nn \l_texnegar_fnt_yekan_tl { yekan }
194 \tl_set:Nn \l_texnegar_fnt_yermook_tl { yermook }
195 \tl_set:Nn \l_texnegar_fnt_zar_tl { zar }
196 \tl_set:Nn \l_texnegar_fnt_ziba_tl { ziba }
197 \tl_set:Nn \l_texnegar_fnt_default_tl { default }
198 \tl_set:Nn \l_texnegar_fnt_noskip_tl { noskip }
199
200 \tl_set:Nn \l_texnegar_lig_aalt_tl { aalt } % Access All Alternatives
201 \tl_set:Nn \l_texnegar_lig_ccmp_tl { ccmp } % Glyph Composition/Decomposition
202 \tl_set:Nn \l_texnegar_lig_dlig_tl { dlig } % Discretionary Ligatures
203 \tl_set:Nn \l_texnegar_lig_fina_tl { fina } % Final (Terminal) Forms
204 \tl_set:Nn \l_texnegar_lig_init_tl { init } % Initial Forms
205 \tl_set:Nn \l_texnegar_lig_locl_tl { locl } % Localized Forms
206 \tl_set:Nn \l_texnegar_lig_medi_tl { medi } % Medial Forms

```

```

207 \tl_set:Nn \l_texnegar_lig_rlig_tl { rlig } % Required Ligatures
208 \tl_set:Nn \l_texnegar_lig_default_tl { default }
209
210 \tl_set:Nn \l_texnegar_col_default_tl { magenta }
211
212 \clist_set:Nn \l_texnegar_lig_aalt_clist { } % Access All Alternatives
213 \clist_set:Nn \l_texnegar_lig_ccmp_clist { } % Glyph Composition/Decomposition
214 \clist_set:Nn \l_texnegar_lig_dlig_clist { FDF2 = , FDF3 = , FDFB = } % Discretionary
215 \clist_set:Nn \l_texnegar_lig_fina_clist { } % Final (Terminal) Forms
216 \clist_set:Nn \l_texnegar_lig_init_clist { } % Initial Forms
217 \clist_set:Nn \l_texnegar_lig_locl_clist { } % Localized Forms
218 \clist_set:Nn \l_texnegar_lig_medi_clist { } % Medial Forms
219 \clist_set:Nn \l_texnegar_lig_rlig_clist { } % Required Ligatures
220 \clist_set:Nn \l_texnegar_lig_default_clist { }
221
222 \clist_set:Nn \l_texnegar_lig_names_clist
223 {
224   \l_texnegar_lig_aalt_tl , { \l_texnegar_lig_aalt_clist } ,
225   \l_texnegar_lig_ccmp_tl , { \l_texnegar_lig_ccmp_clist } ,
226   \l_texnegar_lig_dlig_tl , { \l_texnegar_lig_dlig_clist } ,
227   \l_texnegar_lig_fina_tl , { \l_texnegar_lig_fina_clist } ,
228   \l_texnegar_lig_init_tl , { \l_texnegar_lig_init_clist } ,
229   \l_texnegar_lig_locl_tl , { \l_texnegar_lig_locl_clist } ,
230   \l_texnegar_lig_medi_tl , { \l_texnegar_lig_medi_clist } ,
231   \l_texnegar_lig_rlig_tl , { \l_texnegar_lig_rlig_clist } ,
232 }
233
234 \msg_new:nnn { texnegar } { error-kashida-character-is-not-available-in-the-main-
font }
235 {
236   Sorry,~ kashida~ character~ is~ not~ available~ in~ the~ main~ font~#1!
237 }
238
239 \msg_new:nnn { texnegar } { error-value-not-available-for-kashida-option }
240 {
241   Sorry,~ value~ '#1'~ is~ not~ available~ for~ 'Kashida'~ option~ yet~!
242 }
243
244 \msg_new:nnn { texnegar } { error-specify-value-for-kashida-option }
245 {
246   Sorry,~ you~ must~ specify~ a~ value~ for~ 'Kashida'~ option~ yet~!
247 }
248
249 \msg_new:nnn { texnegar } { warning-experimental-feature }
250 {
251   Please~ note~ that~ the~ feature~ '#1'~ is~ still~ experimental~
252   and~ is~ not~ regarded~ as~ stable.
253 }
254
255 \msg_new:nnn { texnegar } { hm-series-font-not-found }
256 {
257   Either~ the~ font~'#1'~ is~ not~ installed~ on~ your~ system~ or~ does~ not~
258   belong~ to~ HM-Series-fonts.~
259   Please~ note~ that~ the~ option~ 'Kashida=leaders+glyph'~ is~ currently~ only~

```

```

260 supported~ by~ HM~Series~fonts.~
261 If~ you~ know~ of~ any~ other~ font~ that~ supports~ this~ option,~ please~
262 let~ me~ know~ to~ add~ it~ to~ the~ list~ of~ corresponding~ fonts.~
263 }
264
265 \msg_new:nnn { texnegar } { luatex-version-is-too-old }
266 {
267 #1:~Your~luatex~is~too~old,~you~need~at~least~version~#2.#3~!
268 }
269
270 \keys_define:nn { texnegar }
271 {
272 Kashidafontfamily .code:n =
273 {
274 \tl_set:Nn \l_tmpa_tl { #1 }
275 \tl_case:Nn \l_tmpa_tl
276 {
277 \tl_if_empty:NTF \l_tmpa_tl
278 {
279 \bool_set_false:N \l_texnegar_kashida_fontfamily_bool
280 }
281 {
282 \bool_set_true:N \l_texnegar_kashida_fontfamily_bool
283 \tl_set:Nx \l_texnegar_kashida_fontfamily_tl { \l_tmpa_tl }
284 }
285 }
286 } ,
287
288 Minimal .code:n =
289 {
290 \tl_set:Nn \l_tmpa_tl { #1 }
291 \tl_case:Nn \l_tmpa_tl
292 {
293 \l_texnegar_minimal_off_tl
294 {
295 \bool_set_false:N \l_texnegar_minimal_bool
296 }
297 \l_texnegar_minimal_on_tl
298 {
299 \bool_set_true:N \l_texnegar_minimal_bool
300 }
301 }
302 } ,
303
304 Kashida .code:n =
305 {
306 \tl_set:Nn \l_tmpa_tl { #1 }
307 \tl_case:NnTF \l_tmpa_tl
308 {
309 \l_texnegar_stretch_glyph_tl
310 {
311 \msg_warning:nnn { texnegar } { warning-experimental-feature } { Kashida=gly
312 \tl_set:Nx \l_texnegar_gap_filler_tl { \l_texnegar_stretch_glyph_tl }
313 \AtBeginDocument

```



```

314         {
315             \tl_set:Nx \l_texnegar_main_font_full_tl { \tex_fontname:D \tex_the:D \t
316             \tl_set:Nx \l_texnegar_main_font_name_tl { \l_texnegar_main_font_full_tl
317             \regex_replace_once:nnN { ^"([~/+)]/. * } { \1 } \l_texnegar_main_font_na
318         }
319         \bool_set_true:N \l_texnegar_kashida_fix_bool
320         \bool_set_true:N \l_texnegar_kashida_glyph_bool
321     }
322     \l_texnegar_stretch_leaders_glyph_tl
323     {
324         \tl_set:Nx \l_texnegar_gap_filler_tl { \l_texnegar_stretch_leaders_glyph_tl
325         \bool_set_true:N \l_texnegar_kashida_fix_bool
326         \bool_set_true:N \l_texnegar_kashida_leaders_glyph_bool
327     }
328     \l_texnegar_stretch_leaders_hruler_tl
329     {
330         \tl_set:Nx \l_texnegar_gap_filler_tl { \l_texnegar_stretch_leaders_hruler_tl
331         \bool_set_true:N \l_texnegar_kashida_fix_bool
332         \bool_set_true:N \l_texnegar_kashida_leaders_hruler_bool
333     }
334     \l_texnegar_stretch_off_tl
335     {
336         \tl_set:Nx \l_texnegar_gap_filler_tl { \l_texnegar_stretch_off_tl }
337         \bool_set_false:N \l_texnegar_kashida_fix_bool
338     }
339     \l_texnegar_stretch_on_tl
340     {
341         \tl_set:Nx \l_texnegar_gap_filler_tl { \l_texnegar_stretch_leaders_glyph_tl
342         \bool_set_true:N \l_texnegar_kashida_fix_bool
343         \bool_set_true:N \l_texnegar_kashida_leaders_glyph_bool
344     }
345     } { } { \tl_set:Nx \l_texnegar_gap_filler_tl { #1 } }
346     \tl_if_empty:NT \l_texnegar_gap_filler_tl { \msg_error:nn { texnegar } { error-
specify-value-for-kashida-option } }
347     } ,
348
349     linebreakpenalty .code:n =
350     {
351         \int_set:Nn \l_tmpa_int { #1 }
352         \int_case:nnTF \l_tmpa_int
353         {
354             \l_texnegar_min_penalty_int { \int_set:Nn \l_texnegar_line_break_penalty_int {
355             \l_texnegar_low_penalty_int { \int_set:Nn \l_texnegar_line_break_penalty_int {
356             \l_texnegar_med_penalty_int { \int_set:Nn \l_texnegar_line_break_penalty_int {
357             \l_texnegar_high_penalty_int { \int_set:Nn \l_texnegar_line_break_penalty_int {
358             \l_texnegar_max_penalty_int { \int_set:Nn \l_texnegar_line_break_penalty_int {
359         } { } { \int_set:Nn \l_texnegar_line_break_penalty_int { #1 } }
360         \bool_set_true:N \l_texnegar_linebreakpenalty_bool
361     } ,
362
363     kashidastretch .code:n =
364     {
365         \tl_set:Nn \l_tmpa_tl { #1 }
366         \tl_case:NnTF \l_tmpa_tl

```

```

367     {
368         \l_texnegar_fnt_kayhan_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.14
369         \l_texnegar_fnt_kayhannavaar_tl   { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
370         \l_texnegar_fnt_kayhanpook_tl     { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
371         \l_texnegar_fnt_kayhansayeh_tl    { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
372         \l_texnegar_fnt_khoramshahr_tl    { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
373         \l_texnegar_fnt_khorramshahr_tl   { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
374         \l_texnegar_fnt_nilooфар_tl       { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
375         \l_texnegar_fnt_paatch_tl         { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
376         \l_texnegar_fnt_riyaz_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
377         \l_texnegar_fnt_roya_tl            { \tl_set:Nn \l_texnegar_skip_default_tl { 0.14
378         \l_texnegar_fnt_shafigh_tl        { \tl_set:Nn \l_texnegar_skip_default_tl { 0.14
379         \l_texnegar_fnt_shafighKurd_tl    { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
380         \l_texnegar_fnt_shafighUzbek_tl   { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
381         \l_texnegar_fnt_shiraz_tl         { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
382         \l_texnegar_fnt_sols_tl           { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
383         \l_texnegar_fnt_tabriz_tl         { \tl_set:Nn \l_texnegar_skip_default_tl { 0.11
384         \l_texnegar_fnt_titr_tl           { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
385         \l_texnegar_fnt_titre_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
386         \l_texnegar_fnt_traffic_tl        { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
387         \l_texnegar_fnt_vahid_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
388         \l_texnegar_fnt_vosta_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
389         \l_texnegar_fnt_yaghut_tl         { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
390         \l_texnegar_fnt_yagut_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
391         \l_texnegar_fnt_yas_tl            { \tl_set:Nn \l_texnegar_skip_default_tl { 0.12
392         \l_texnegar_fnt_yekan_tl          { \tl_set:Nn \l_texnegar_skip_default_tl { 0.14
393         \l_texnegar_fnt_yermook_tl        { \tl_set:Nn \l_texnegar_skip_default_tl { 0.13
394         \l_texnegar_fnt_zar_tl            { \tl_set:Nn \l_texnegar_skip_default_tl { 0.11
395         \l_texnegar_fnt_ziba_tl           { \tl_set:Nn \l_texnegar_skip_default_tl { 0.11
396         \l_texnegar_fnt_default_tl        { \tl_set:Nn \l_texnegar_skip_default_tl { 0.14
397         \l_texnegar_fnt_noskip_tl         { \tl_set:Nn \l_texnegar_skip_default_tl { 0
398     } { } { \tl_set:Nn \l_texnegar_skip_default_tl { #1 } }
399 } ,
400 kashidastretch .default:n = \tl_set:Nn \l_texnegar_skip_default_tl { 0 em plus 0.5 em }
401
402 ligatures .code:n =
403 {
404     \tl_set:Nn \l_tmpa_tl { #1 }
405     \tl_case:NnTF \l_tmpa_tl
406     {
407         \l_texnegar_lig_aalt_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
408         \l_texnegar_lig_ccmp_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
409         \l_texnegar_lig_dlig_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
410         \l_texnegar_lig_fina_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
411         \l_texnegar_lig_init_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
412         \l_texnegar_lig_locl_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
413         \l_texnegar_lig_medi_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
414         \l_texnegar_lig_rlig_tl          { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
415         \l_texnegar_lig_default_tl       { \tl_set:Nx \l_texnegar_active_ligs_tl { \l_texnegar
416     } { } { \tl_set:Nn \l_texnegar_active_ligs_tl { #1 } }
417     \bool_set_true:N \l_texnegar_ligature_bool
418 } ,
419 ligatures .default:n = \tl_set:Nn \l_texnegar_active_ligs_tl { \l_texnegar_lig_default_t
420

```

```

421 color .code:n =
422 {
423   \tl_set:Nn \l_tmpa_tl { #1 }
424   \tl_if_empty:NTF \l_tmpa_tl
425   {
426     \tl_set:Nx \l_texnegar_color_tl { \l_texnegar_col_default_tl }
427   }
428   {
429     \tl_set:Nx \l_texnegar_color_tl { \l_tmpa_tl }
430   }
431   \bool_set_true:N \l_texnegar_color_bool
432   \sys_if_engine_luatex:T
433   {
434     \convertcolorspec{named}{\l_texnegar_color_tl}{rgb}\l_texnegar_color_rgb_tl
435     \sys_if_engine_luatex:T
436     {
437       \directlua{\l_texnegar_color_rgb_tl = "\l_texnegar_color_rgb_tl"}
438     }
439   }
440 } ,
441
442 hboxrecursion .code:n =
443 {
444   \tl_set:Nn \l_tmpa_tl { #1 }
445   \tl_case:NnTF \l_tmpa_tl
446   {
447     \l_texnegar_hboxrecursion_off_tl
448     {
449       \bool_set_false:N \l_texnegar_hboxrecursion_bool
450     }
451     \l_texnegar_hboxrecursion_on_tl
452     {
453       \bool_set_true:N \l_texnegar_hboxrecursion_bool
454     }
455   } { } { \bool_set_false:N \l_texnegar_hboxrecursion_bool }
456 } ,
457 hboxrecursion .default:n = \bool_set_true:N \l_texnegar_hboxrecursion_bool ,
458
459 vboxrecursion .code:n =
460 {
461   \tl_set:Nn \l_tmpa_tl { #1 }
462   \tl_case:NnTF \l_tmpa_tl
463   {
464     \l_texnegar_vboxrecursion_off_tl
465     {
466       \bool_set_false:N \l_texnegar_vboxrecursion_bool
467     }
468     \l_texnegar_vboxrecursion_on_tl
469     {
470       \bool_set_true:N \l_texnegar_vboxrecursion_bool
471     }
472   } { } { \bool_set_false:N \l_texnegar_vboxrecursion_bool }
473 } ,
474 vboxrecursion .default:n = \bool_set_true:N \l_texnegar_vboxrecursion_bool ,

```

```

475 }
476
477 \ProcessKeysOptions { texnegar }
478
479 \sys_if_engine luatex:T
480 {
481   \NewDocumentCommand \KashidaHMFixOff {} { \directlua{StopStretching()} }
482   \NewDocumentCommand \KashidaHMFixOn  {} { \directlua{StartStretching()} }
483 }
484
485 \sys_if_engine xetex:T
486 {
487   \NewDocumentCommand \KashidaHMFixOn {} { \bool_set_true:N \l_texnegar_kashida_fix_bool }
488   \NewDocumentCommand \KashidaHMFixOff {} { \bool_set_false:N \l_texnegar_kashida_fix_bool }
489 }
490
491 \tex_let:D \KashidaOn \KashidaHMFixOn
492 \tex_let:D \KashidaOff \KashidaHMFixOff
493
494 \bool_if:NTF \l_texnegar_kashida_fix_bool
495 {
496   \tl_if_empty:NT \l_texnegar_skip_default_tl { \tl_set:Nn \l_texnegar_skip_default_tl {
497   }
498   {
499     \tl_set:NV \l_texnegar_skip_default_tl \c_texnegar_skip_a_tl
500   }
501 }
502 %% % \makeatletter
503 %% % \newif\if@Kashida@on
504 %% Becuase Vafa Khalighi has copied the above code (injecting the character uni+200E) in xepersian
505 %% (https://tug.org/svn/texlive/trunk/Master/texmf-dist/tex/xelatex/xepersian/kashida-xepersian.def?revision=55165&view=co),
506 %% the following line of code is not needed in xepersian anymore.
507 %% % \newif\if@Kashida@XB@fix
508 %% % \makeatother
509
510 \bool_if:NF \l_texnegar_minimal_bool
511 {
512   \directlua{dofile(kpse.find_file("luatex-tools.lua"))}
513   \input texnegar-luabidi.tex
514 }
515
516 \endinput
517 </texnegar-ini-tex>

```

1.5 File: texnegar-common-kashida.tex

```

518 <*texnegar-common-kashida-tex>
519 \ProvidesExplFile {texnegar-common-kashida.tex} {2021-02-09} {0.1e} { Full implementation of
520
521 \bool_if:NT \l_texnegar_ligature_bool
522 {
523   \clist_new:N \l_texnegar_ligatures_clist
524   \int_new:N \l_texnegar_lig_names_len_int

```

```

525 \int_set:Nn \l_texnegar_lig_names_len_int { \clist_count:N \l_texnegar_lig_names_clist }
526 \int_step_inline:nmmm { 1 } { 2 } { \l_texnegar_lig_names_len_int }
527 {
528   \int_set:Nn \l_tmpa_int { #1 }
529   \int_set:Nn \l_tmpb_int { \int_eval:n { \l_tmpa_int + 1 } }
530   \tl_set:Nf \l_tmpa_tl { \clist_item:Nn \l_texnegar_lig_names_clist { \l_tmpa_int } }
531   \clist_set:Nx \l_tmpa_clist { { \clist_item:Nn \l_texnegar_lig_names_clist { \l_tmpb_int } } }
532   \bool_if:nT { \tl_if_eq_p:NN \l_texnegar_active_ligs_tl \l_tmpa_tl || \tl_if_eq_p:NN
533     {
534       \clist_put_left:Nx \l_texnegar_ligatures_clist { \l_tmpa_clist }
535     }
536   }
537 \clist_map_inline:Nn \l_texnegar_ligatures_clist
538 {
539   \seq_set_split:Nnn \l_tmpa_seq { = } { #1 }
540   \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl { } { }
541   \seq_pop_left:NN \l_tmpa_seq \l_tmpb_tl { } { }
542   \tl_const:cx { \tl_use:N \l_tmpb_tl } { \char" \l_tmpa_tl \ }
543 }
544 }
545
546 \bool_if:NT \l_texnegar_linebreakpenalty_bool
547 {
548   %% Partly adapted from LaTeX2e source
549   \cs_new:Nn \texnegar_line_break: {
550     \if_mode_vertical:
551       \GenericError{
552         \space\space\space\space\space\space\space\space\space\space\space\space\space\space\space\space
553       }{
554         LaTeX Error: There's no line here to end
555       }{
556         See the LaTeX manual or LaTeX Companion for explanation.
557       }{
558         Your command was ignored.\MessageBreak
559         Type \space I <command> <return> \space to replace it~
560         with another command,\MessageBreak
561         or \space <return> \space to continue without it.}
562     \else:
563       \l_tmpa_skip \tex_lastskip:D
564       \tex_unskip:D
565       \tex_penalty:D -\l_texnegar_line_break_penalty_int
566       \dim_compare:nT { \l_tmpa_skip > \c_zero_skip }
567         { \skip_horizontal:N \l_tmpa_skip \tex_ignorespaces:D }
568     \fi:
569   }
570
571 \NewDocumentCommand { \discouragebadlinebreaks } { 0{ \l_texnegar_line_break_penalty_int } }
572 {
573   \IfNoValueF {#1}
574     { \int_set:Nn \l_texnegar_line_break_penalty_int {#1} }
575   \IfNoValueF {#2}
576     { \tl_set:Nn \l_texnegar_skip_default_tl {#2} }
577   \texnegar_put_line_breaks:n { #3 }
578 }

```

```

579
580 \cs_new_protected:Nn \texnegar_put_line_breaks:n
581 {
582   \tl_set:Nn \l_texnegar_line_break_tl { #1 }
583   \regex_replace_all:nnN { ([\])+ } { \ \0 \ \c{texnegar_line_break:}\ } \l_texnegar_line_break_tl
584   \tl_use:N \l_texnegar_line_break_tl
585 }
586 }
587
588 \endinput
589 </texnegar-common-kashida.tex>

```

1.6 File: texnegar-xetex-kashida.tex

```

590 <*texnegar-xetex-kashida.tex>
591 \ProvidesExplFile {texnegar-xetex-kashida.tex} {2021-02-09} {0.1e} { Full implementation of
592
593 \newXeTeXintercharclass \c_texnegar_d_charclass % dual-joiner class
594 \newXeTeXintercharclass \c_texnegar_l_charclass % lam
595 \newXeTeXintercharclass \c_texnegar_r_charclass % right-joiner
596 \newXeTeXintercharclass \c_texnegar_a_charclass % alef
597 \newXeTeXintercharclass \c_texnegar_y_charclass % yeh
598
599 \tex_input:D { texnegar-common-kashida.tex }
600
601 \tl_set:Nn \l_texnegar_use_color_tl
602 {
603   \bool_if:NTF \l_texnegar_color_bool
604   {
605     \colorlet{default}{\l_texnegar_color_tl}
606   }
607   {
608     \colorlet{default}{.}
609   }
610   \color{default}
611 }
612
613 %% Partly adapted from the code provided by David Carlisle in:
614 %% https://tex.stackexchange.com/questions/356709/how-to-know-the-width-and-fill-the-glue-space-between-two-characters-when-using/356721#356721
615 \cs_new:Npn \texnegar_kashida_glyph #1
616 {
617   \bool_if:NT \l_texnegar_kashida_fix_bool
618   {
619     \c_texnegar_lrm_int\tex_penalty:D 10000
620     \mode_leave_vertical:
621     \tex_global:D \tex_advance:D \l_texnegar_counter_int \c_one_int
622
623     \tl_set:Nx \l_texnegar_pos_tl { p\tex_romannumeral:D \l_texnegar_counter_int }
624     \tl_set:Nx \l_texnegar_zref_tl { z\tex_romannumeral:D \l_texnegar_counter_int }
625
626     \zsaveposx{x_i\l_texnegar_zref_tl}
627     \tl_set:Nx \l_tmpa_tl
628     {
629       \iow_now:cx { @auxout }

```

```

630     {
631       \token_to_str:N \gdef \exp_after:wN \token_to_str:N \cs:w xi\l_texnegar_pos_tl \cs
632     }
633   }
634   \l_tmpa_tl
635   \skip_horizontal:n { #1 }
636   \zsaveposx{x_f_\l_texnegar_zref_tl}
637   \tl_set:Nx \l_tmpa_tl
638     {
639       \iow_now:cx { @auxout }
640       {
641         \token_to_str:N \gdef \exp_after:wN \token_to_str:N \cs:w xf\l_texnegar_pos_tl \cs
642       }
643     }
644   \l_tmpa_tl
645   \exp_after:wN
646   \if_meaning:w
647     \cs:w xi\l_texnegar_pos_tl \cs_end: \tex_relax:D
648   \else:
649     \dim_set:Nn \l_texnegar_diff_pos_dim
650       {
651         \dim_eval:n { \cs:w xi\l_texnegar_pos_tl \cs_end: sp - \cs:w xf\l_texnegar_pos_tl
652       }
653     \dim_compare:nTF { \l_texnegar_diff_pos_dim == 0sp }
654       { }
655     { \llap { \resizebox { \l_texnegar_diff_pos_dim \tex_relax:D } { \height } { \l_texn
656   \fi:
657 }
658 }
659
660 \cs_new:Npn \texnegar_kashida_leaders #1
661 {
662   \bool_if:NT \l_texnegar_kashida_fix_bool
663   {
664     \tl_if_eq:NNTF \l_texnegar_gap_filler_tl \l_texnegar_stretch_leaders_glyph_tl
665     {
666       \tl_set:Nx \l_texnegar_font_full_tl { \tex_fontname:D \tex_the:D \tex_font:D }
667       \tl_set:Nx \l_texnegar_font_name_tl { \l_texnegar_font_full_tl }
668       \tl_set:Nx \l_texnegar_font_init_tl { \l_texnegar_font_name_tl }
669       \regex_replace_once:nnN { ~" \[?(HM) [\_ \ ] (X|F) .* } { \1\2 } \l_texnegar_font_init_
670       \tl_set:Nn \l_tmpa_tl { HMF }
671       \tl_set:Nn \l_tmpb_tl { HMX }
672       \bool_if:nTF { \str_if_eq_p:NN { \l_texnegar_font_init_tl } { \l_tmpa_tl } || \str
673       {
674         \hbox_set:Nn \l_texnegar_ksh_box { \l_texnegar_use_color_tl \XeTeXglyph\XeTeXg
675         \c_texnegar_zwj_int \tex_penalty:D 10000
676         \tex_leaders:D \copy\l_texnegar_ksh_box \skip_horizontal:n { #1 }
677         \c_texnegar_zwj_int
678       }
679       {
680         \msg_error:nxx { texnegar } { hm-series-font-not-found } { \l_texnegar_font_na
681       }
682     }
683   }

```

```

684         %% Partly adapted from the code provided by Jonathan Kew in:
685         %% https://tug.org/pipermail/xetex/2009-February/012307.html.
686         %% Somebody notified me that the code in 'kashida-xepersian.def' from xepersian
687         %% package is an exact copy of Jonathan Kew's code. Being unaware of this, in
688         %% the earlier versions of this package I made a mistake and acknowledged
689         %% Vafa Khalighi instead of Jonathan Kew. A sincere thank you to Jonathan Kew
690         %% for his excellent code.
691         \c_texnegar_lrm_int\c_texnegar_zwj_int
692         {\l_texnegar_use_color_tl\tex_penalty:D 10000
693         \tex_leaders:D \tex_hrulerule:D height \XeTeXglyphbounds \c_texnegar_two_int
694         \int_use:N \XeTeXcharglyph \c_texnegar_ksh_int depth \XeTeXglyphbounds \c_texnegar
695         \int_use:N \XeTeXcharglyph \c_texnegar_ksh_int \skip_horizontal:n { #1 }
696         }
697         \c_texnegar_zwj_int
698     }
699 }
700 }
701
702 \XeTeXinterchartokenstate = 1
703
704 \clist_set:Nn \l_texnegar_a_clist { 0622,0623,0625,0627 } %
705 \clist_map_inline:Nn \l_texnegar_a_clist
706 {
707     \XeTeXcharclass "#1 \c_texnegar_a_charclass
708 }
709
710 \clist_set:Nn \l_texnegar_d_clist { 0626,0628,062A,062B,062C,062D,062E,0633,0634,0635,0636,0
711 \clist_map_inline:Nn \l_texnegar_d_clist
712 {
713     \XeTeXcharclass "#1 \c_texnegar_d_charclass
714 }
715
716 \clist_set:Nn \l_texnegar_l_clist { 0644 } %
717 \clist_map_inline:Nn \l_texnegar_l_clist
718 {
719     \XeTeXcharclass "#1 \c_texnegar_l_charclass
720 }
721
722 \clist_set:Nn \l_texnegar_r_clist { 0624,0629,062F,0630,0631,0632,0648,0698 } % , , , , , ,
723 \clist_map_inline:Nn \l_texnegar_r_clist
724 {
725     \XeTeXcharclass "#1 \c_texnegar_r_charclass
726 }
727
728 \clist_set:Nn \l_texnegar_y_clist { 0649,064A,06CC } % , ,
729 \clist_map_inline:Nn \l_texnegar_y_clist
730 {
731     \XeTeXcharclass "#1 \c_texnegar_y_charclass
732 }
733
734 \tl_if_eq:NNTF \l_texnegar_gap_filler_tl \l_texnegar_stretch_glyph_tl {
735     \XeTeXinterchartoks \c_texnegar_y_charclass \c_texnegar_y_charclass = {
736         \bool_if:NTF \l_texnegar_kashida_fix_bool
737         { \c_texnegar_zwj_int \texnegar_kashida_glyph \l_texnegar_skip_default_tl \c_texnegar_zw

```



```

738   { \c_texnegar_zwj_int \texnegar_kashida_glyph \c_texnegar_skip_a_tl \c_texnegar_zwj_int
739 }
740 \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_y_charclass = {
741   \bool_if:NTF \l_texnegar_kashida_fix_bool
742   { \c_texnegar_zwj_int \texnegar_kashida_glyph \l_texnegar_skip_default_tl \c_texnegar_zwj_int
743   { \c_texnegar_zwj_int \texnegar_kashida_glyph \c_texnegar_skip_a_tl \c_texnegar_zwj_int
744   }
745 \XeTeXinterchartoks \c_texnegar_y_charclass \c_texnegar_d_charclass = { \c_texnegar_zwj_int
746 \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_d_charclass = { \c_texnegar_zwj_int
747 \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_d_charclass = { \c_texnegar_zwj_int
748 \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_l_charclass = { \c_texnegar_zwj_int
749 \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_l_charclass = { \c_texnegar_zwj_int
750 \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_r_charclass = { \c_texnegar_zwj_int
751 \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_a_charclass = { \c_texnegar_zwj_int
752 \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_r_charclass = { \c_texnegar_zwj_int
753 \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_a_charclass = { }
754 }
755 {
756   \bool_if:nTF {
757     \tl_if_eq_p:NN \l_texnegar_gap_filler_tl \l_texnegar_stretch_leaders_glyph_tl ||
758     \tl_if_eq_p:NN \l_texnegar_gap_filler_tl \l_texnegar_stretch_leaders_hrule_tl
759   }
760   {
761     \XeTeXinterchartoks \c_texnegar_y_charclass \c_texnegar_y_charclass = {
762       \bool_if:NTF \l_texnegar_kashida_fix_bool
763       { \texnegar_kashida_leaders \l_texnegar_skip_default_tl }
764       { \texnegar_kashida_leaders \c_texnegar_skip_a_tl }
765     }
766     \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_y_charclass = {
767       \bool_if:NTF \l_texnegar_kashida_fix_bool
768       { \texnegar_kashida_leaders \l_texnegar_skip_default_tl }
769       { \texnegar_kashida_leaders \c_texnegar_skip_a_tl }
770     }
771     \XeTeXinterchartoks \c_texnegar_y_charclass \c_texnegar_d_charclass = { \texnegar_kashid
772     \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_d_charclass = { \texnegar_kashid
773     \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_d_charclass = { \texnegar_kashid
774     \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_l_charclass = { \texnegar_kashid
775     \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_l_charclass = { \texnegar_kashid
776     \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_r_charclass = { \texnegar_kashid
777     \XeTeXinterchartoks \c_texnegar_d_charclass \c_texnegar_a_charclass = { \texnegar_kashid
778     \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_r_charclass = { \texnegar_kashid
779     \XeTeXinterchartoks \c_texnegar_l_charclass \c_texnegar_a_charclass = { }
780   }
781   {
782     \msg_error:nxx { texnegar } { error-value-not-available-for-kashida-option } { \l_texneg
783   }
784 }
785
786 \endinput
787 </texnegar-xetex-kashida-tex>

```

1.7 File: texnegar-char-table.lua

```

788 < *texnegar-char-table.lua >
789 --

```

```

790 -- This is file 'texnegar-char-table.lua',
791 -- generated with the docstrip utility.
792 --
793 -- The original source files were:
794 --
795 -- texnegar.dtx (with options: 'texnegar-char-table-lua')
796 --
797 -- Copyright (C) 2020-2021 Hossein Movahhedian
798 --
799 -- It may be distributed and/or modified under the LaTeX Project Public License,
800 -- version 1.3c or higher (your choice). The latest version of
801 -- this license is at: http://www.latex-project.org/lppl.txt
802 --
803 -- texnegar_char_table = texnegar_char_table or {}
804 -- local texnegar_char_table = texnegar_char_table
805 -- texnegar_char_table.module = {
806 --     name = "texnegar_char_table",
807 --     version = "0.1e",
808 --     date = "2021-02-09",
809 --     description = "Full implementation of kashida feature in XeLaTeX and LuaLaTeX",
810 --     author = "Hossein Movahhedian",
811 --     copyright = "Hossein Movahhedian",
812 --     license = "LPPL v1.3c"
813 -- }
814 --
815 -- -- ^A% texnegar-lua.dtx -- part of TEXNEGAR <bitbucket.org/dma8hm1334/texnegar>
816 -- local err, warn, info, log = luatexbase.provides_module(texnegar_char_table.module)
817 -- texnegar_char_table.log = log or (function (s) luatexbase.module_info("texnegar_char_table", s))
818 -- texnegar_char_table.warning = warn or (function (s) luatexbase.module_warning("texnegar_char_table", s))
819 -- texnegar_char_table.error = err or (function (s) luatexbase.module_error("texnegar_char_table", s))
820 --
821 local peCharTableDiacritic = {
822 [1611] = utf8.char(1611), -- "", utf8.codepoint("") == 1611, "\u{064B}", ARABIC-
FATHATAN
823 [1612] = utf8.char(1612), -- "", utf8.codepoint("") == 1612, "\u{064C}", ARABIC-
DAMMATAN
824 [1613] = utf8.char(1613), -- "", utf8.codepoint("") == 1613, "\u{064D}", ARABIC-
KASRATAN
825 [1614] = utf8.char(1614), -- "", utf8.codepoint("") == 1614, "\u{064E}", ARABIC-
FATHA
826 [1615] = utf8.char(1615), -- "", utf8.codepoint("") == 1615, "\u{064F}", ARABIC-
DAMMA
827 [1616] = utf8.char(1616), -- "", utf8.codepoint("") == 1616, "\u{0650}", ARABIC-
KASRA
828 [1617] = utf8.char(1617), -- "", utf8.codepoint("") == 1617, "\u{0651}", ARABIC-
SHADDA
829 [1618] = utf8.char(1618), -- "", utf8.codepoint("") == 1618, "\u{0652}", ARABIC-
SUKUN
830 [1619] = utf8.char(1619), -- "", utf8.codepoint("") == 1619, "\u{0653}", ARABIC-
MADDA ABOVE
831 [1620] = utf8.char(1620), -- "", utf8.codepoint("") == 1620, "\u{0654}", ARABIC-
HAMZA ABOVE
832 [1621] = utf8.char(1621), -- "", utf8.codepoint("") == 1621, "\u{0655}", ARABIC-
HAMZA BELOW

```

```

833 [1622] = utf8.char(1622), -- "", utf8.codepoint("") == 1622, "\u{0656}", ARABIC-
SUBSCRIPT ALEF
834 [1623] = utf8.char(1623), -- "", utf8.codepoint("") == 1623, "\u{0657}", ARABIC-
INVERTED DAMMA
835 [1624] = utf8.char(1624), -- "", utf8.codepoint("") == 1624, "\u{0658}", ARABIC-
MARK NOON GHUNNA
836 [1625] = utf8.char(1625), -- "", utf8.codepoint("") == 1625, "\u{0659}", ARABIC-
ZWARAKAY
837 [1648] = utf8.char(1648), -- "", utf8.codepoint("") == 1648, "\u{0670}", ARABIC-
SUPERSCRIPT ALEF
838 [64606] = utf8.char(64606), -- "", utf8.codepoint("") == 64606, "\u{FC5E}", ARABIC-
LIGATURE SHADDA WITH DAMMATAN ISOLATED FORM
839 [64607] = utf8.char(64607), -- "", utf8.codepoint("") == 64607, "\u{FC5F}", ARABIC-
LIGATURE SHADDA WITH KASRATAN ISOLATED FORM
840 [64608] = utf8.char(64608), -- "", utf8.codepoint("") == 64608, "\u{FC60}", ARABIC-
LIGATURE SHADDA WITH FATHA ISOLATED FORM
841 [64609] = utf8.char(64609), -- "", utf8.codepoint("") == 64609, "\u{FC61}", ARABIC-
LIGATURE SHADDA WITH DAMMA ISOLATED FORM
842 [64610] = utf8.char(64610), -- "", utf8.codepoint("") == 64610, "\u{FC62}", ARABIC-
LIGATURE SHADDA WITH KASRA ISOLATED FORM
843 [64611] = utf8.char(64611), -- "", utf8.codepoint("") == 64611, "\u{FC63}", ARABIC-
LIGATURE SHADDA WITH SUPERSCRIPT ALEF ISOLATED FORM
844 }
845
846 local peCharTableDigit = {
847 [1632] = utf8.char(1632), -- "", utf8.codepoint("") == 1632, "\u{0660}", ARABIC-
INDIC DIGIT ZERO
848 [1633] = utf8.char(1633), -- "", utf8.codepoint("") == 1633, "\u{0661}", ARABIC-
INDIC DIGIT ONE
849 [1634] = utf8.char(1634), -- "", utf8.codepoint("") == 1634, "\u{0662}", ARABIC-
INDIC DIGIT TWO
850 [1635] = utf8.char(1635), -- "", utf8.codepoint("") == 1635, "\u{0663}", ARABIC-
INDIC DIGIT THREE
851 [1636] = utf8.char(1636), -- "", utf8.codepoint("") == 1636, "\u{0664}", ARABIC-
INDIC DIGIT FOUR
852 [1637] = utf8.char(1637), -- "", utf8.codepoint("") == 1637, "\u{0665}", ARABIC-
INDIC DIGIT FIVE
853 [1638] = utf8.char(1638), -- "", utf8.codepoint("") == 1638, "\u{0666}", ARABIC-
INDIC DIGIT SIX
854 [1639] = utf8.char(1639), -- "", utf8.codepoint("") == 1639, "\u{0667}", ARABIC-
INDIC DIGIT SEVEN
855 [1640] = utf8.char(1640), -- "", utf8.codepoint("") == 1640, "\u{0668}", ARABIC-
INDIC DIGIT EIGHT
856 [1641] = utf8.char(1641), -- "", utf8.codepoint("") == 1641, "\u{0669}", ARABIC-
INDIC DIGIT NINE
857 [1780] = utf8.char(1780), -- "", utf8.codepoint("") == 1780, "\u{06F4}", EXTENDED ARABI
INDIC DIGIT FOUR
858 [1781] = utf8.char(1781), -- "", utf8.codepoint("") == 1781, "\u{06F5}", EXTENDED ARABI
INDIC DIGIT FIVE
859 [1782] = utf8.char(1782), -- "", utf8.codepoint("") == 1782, "\u{06F6}", EXTENDED ARABI
INDIC DIGIT SIX
860 }
861
862 local peCharTablePunctuation = {

```

```

863 [1548] = utf8.char(1548), -- "", utf8.codepoint("") == 1548, "\u{060C}", ARABIC COMMA
864 [1549] = utf8.char(1549), -- "", utf8.codepoint("") == 1549, "\u{060D}", ARABIC DATE SE
865 [1563] = utf8.char(1563), -- "", utf8.codepoint("") == 1563, "\u{061B}", ARABIC SEMICOL
866 [1567] = utf8.char(1567), -- "", utf8.codepoint("") == 1567, "\u{061F}", ARABIC QUESTIO
867 [1642] = utf8.char(1642), -- "", utf8.codepoint("") == 1642, "\u{066A}", ARABIC PERCENT
868 [1643] = utf8.char(1643), -- "", utf8.codepoint("") == 1643, "\u{066B}", ARABIC DECIMAL
869 [1644] = utf8.char(1644), -- "", utf8.codepoint("") == 1644, "\u{066C}", ARABIC THOUSAN
870 [1645] = utf8.char(1645), -- "", utf8.codepoint("") == 1645, "\u{066D}", ARABIC FIVE PO
871 }
872
873 local peCharTable = {
874 [1569] = utf8.char(1569), -- "", utf8.codepoint("") == 1569, "\u{0621}", ARABIC LETTE
875 [1570] = utf8.char(1570), -- "", utf8.codepoint("") == 1570, "\u{0622}", ARABIC LETTE
876 [1571] = utf8.char(1571), -- "", utf8.codepoint("") == 1571, "\u{0623}", ARABIC LETTE
877 [1572] = utf8.char(1572), -- "", utf8.codepoint("") == 1572, "\u{0624}", ARABIC LETTE
878 [1573] = utf8.char(1573), -- "", utf8.codepoint("") == 1573, "\u{0625}", ARABIC LETTE
879 [1574] = utf8.char(1574), -- "", utf8.codepoint("") == 1574, "\u{0626}", ARABIC LETTE
880 [1575] = utf8.char(1575), -- "", utf8.codepoint("") == 1575, "\u{0627}", ARABIC LETTE
881 [1576] = utf8.char(1576), -- "", utf8.codepoint("") == 1576, "\u{0628}", ARABIC LETTE
882 [1577] = utf8.char(1577), -- "", utf8.codepoint("") == 1577, "\u{0629}", ARABIC LETTE
883 [1578] = utf8.char(1578), -- "", utf8.codepoint("") == 1578, "\u{062A}", ARABIC LETTE
884 [1579] = utf8.char(1579), -- "", utf8.codepoint("") == 1579, "\u{062B}", ARABIC LETTE
885 [1580] = utf8.char(1580), -- "", utf8.codepoint("") == 1580, "\u{062C}", ARABIC LETTE
886 [1581] = utf8.char(1581), -- "", utf8.codepoint("") == 1581, "\u{062D}", ARABIC LETTE
887 [1582] = utf8.char(1582), -- "", utf8.codepoint("") == 1582, "\u{062E}", ARABIC LETTE
888 [1583] = utf8.char(1583), -- "", utf8.codepoint("") == 1583, "\u{062F}", ARABIC LETTE
889 [1584] = utf8.char(1584), -- "", utf8.codepoint("") == 1584, "\u{0630}", ARABIC LETTE
890 [1585] = utf8.char(1585), -- "", utf8.codepoint("") == 1585, "\u{0631}", ARABIC LETTE
891 [1586] = utf8.char(1586), -- "", utf8.codepoint("") == 1586, "\u{0632}", ARABIC LETTE
892 [1587] = utf8.char(1587), -- "", utf8.codepoint("") == 1587, "\u{0633}", ARABIC LETTE
893 [1588] = utf8.char(1588), -- "", utf8.codepoint("") == 1588, "\u{0634}", ARABIC LETTE
894 [1589] = utf8.char(1589), -- "", utf8.codepoint("") == 1589, "\u{0635}", ARABIC LETTE
895 [1590] = utf8.char(1590), -- "", utf8.codepoint("") == 1590, "\u{0636}", ARABIC LETTE
896 [1591] = utf8.char(1591), -- "", utf8.codepoint("") == 1591, "\u{0637}", ARABIC LETTE
897 [1592] = utf8.char(1592), -- "", utf8.codepoint("") == 1592, "\u{0638}", ARABIC LETTE
898 [1593] = utf8.char(1593), -- "", utf8.codepoint("") == 1593, "\u{0639}", ARABIC LETTE
899 [1594] = utf8.char(1594), -- "", utf8.codepoint("") == 1594, "\u{063A}", ARABIC LETTE
900 [1601] = utf8.char(1601), -- "", utf8.codepoint("") == 1601, "\u{0641}", ARABIC LETTE
901 [1602] = utf8.char(1602), -- "", utf8.codepoint("") == 1602, "\u{0642}", ARABIC LETTE
902 [1603] = utf8.char(1603), -- "", utf8.codepoint("") == 1603, "\u{0643}", ARABIC LETTE
903 [1604] = utf8.char(1604), -- "", utf8.codepoint("") == 1604, "\u{0644}", ARABIC LETTE
904 [1605] = utf8.char(1605), -- "", utf8.codepoint("") == 1605, "\u{0645}", ARABIC LETTE
905 [1606] = utf8.char(1606), -- "", utf8.codepoint("") == 1606, "\u{0646}", ARABIC LETTE
906 [1607] = utf8.char(1607), -- "", utf8.codepoint("") == 1607, "\u{0647}", ARABIC LETTE
907 [1608] = utf8.char(1608), -- "", utf8.codepoint("") == 1608, "\u{0648}", ARABIC LETTE
908 [1609] = utf8.char(1609), -- "", utf8.codepoint("") == 1609, "\u{0649}", ARABIC LETTE
909 [1610] = utf8.char(1610), -- "", utf8.codepoint("") == 1610, "\u{064A}", ARABIC LETTE
910 [1662] = utf8.char(1662), -- "", utf8.codepoint("") == 1662, "\u{067E}", ARABIC LETTE
911 [1670] = utf8.char(1670), -- "", utf8.codepoint("") == 1670, "\u{0686}", ARABIC LETTE
912 [1688] = utf8.char(1688), -- "", utf8.codepoint("") == 1688, "\u{0698}", ARABIC LETTE
913 [1705] = utf8.char(1705), -- "", utf8.codepoint("") == 1705, "\u{06A9}", ARABIC LETTE
914 [1706] = utf8.char(1706), -- "", utf8.codepoint("") == 1706, "\u{06AA}", ARABIC LETTE
915 [1711] = utf8.char(1711), -- "", utf8.codepoint("") == 1711, "\u{06AF}", ARABIC LETTE
916 [1726] = utf8.char(1726), -- "", utf8.codepoint("") == 1726, "\u{06BE}", ARABIC LETTE

```

```

917 [1728] = utf8.char(1728), -- "", utf8.codepoint("") == 1728, "\u{06C0}", ARABIC LETTE
918 [1740] = utf8.char(1740), -- "", utf8.codepoint("") == 1740, "\u{06CC}", ARABIC LETTE
919 [1749] = utf8.char(1749), -- "", utf8.codepoint("") == 1740, "\u{06D5}", ARABIC LETTE
920 [65275] = utf8.char(65275), -- "", utf8.codepoint("") == 65275, "\u{FEFB}", ARABIC LIGA
921 [65276] = utf8.char(65276), -- "", utf8.codepoint("") == 65276, "\u{FEFC}", ARABIC LIGA
922 }
923
924 local peCharTableInitial = {
925 [64344] = utf8.char(64344), -- "", utf8.codepoint("") == 64344, "\u{FB58}", INITIAL FOR
926 [64380] = utf8.char(64380), -- "", utf8.codepoint("") == 64380, "\u{FB7C}", INITIAL FOR
927 [64400] = utf8.char(64400), -- "", utf8.codepoint("") == 64400, "\u{FB90}", INITIAL FOR
928 [64404] = utf8.char(64404), -- "", utf8.codepoint("") == 64404, "\u{FB94}", INITIAL FOR
929 [64510] = utf8.char(64510), -- "", utf8.codepoint("") == 64510, "\u{FBFE}", INITIAL FOR
930 [65169] = utf8.char(65169), -- "", utf8.codepoint("") == 65169, "\u{FE91}", INITIAL FOR
931 [65175] = utf8.char(65175), -- "", utf8.codepoint("") == 65175, "\u{FE97}", INITIAL FOR
932 [65179] = utf8.char(65179), -- "", utf8.codepoint("") == 65179, "\u{FE9B}", INITIAL FOR
933 [65183] = utf8.char(65183), -- "", utf8.codepoint("") == 65183, "\u{FE9F}", INITIAL FOR
934 [65187] = utf8.char(65187), -- "", utf8.codepoint("") == 65187, "\u{FEA3}", INITIAL FOR
935 [65191] = utf8.char(65191), -- "", utf8.codepoint("") == 65191, "\u{FEA7}", INITIAL FOR
936 [65203] = utf8.char(65203), -- "", utf8.codepoint("") == 65203, "\u{FEB3}", INITIAL FOR
937 [65207] = utf8.char(65207), -- "", utf8.codepoint("") == 65207, "\u{FEB7}", INITIAL FOR
938 [65211] = utf8.char(65211), -- "", utf8.codepoint("") == 65211, "\u{FEBB}", INITIAL FOR
939 [65215] = utf8.char(65215), -- "", utf8.codepoint("") == 65215, "\u{FEBF}", INITIAL FOR
940 [65219] = utf8.char(65219), -- "", utf8.codepoint("") == 65219, "\u{FEC3}", INITIAL FOR
941 [65223] = utf8.char(65223), -- "", utf8.codepoint("") == 65223, "\u{FEC7}", INITIAL FOR
942 [65227] = utf8.char(65227), -- "", utf8.codepoint("") == 65227, "\u{FECB}", INITIAL FOR
943 [65231] = utf8.char(65231), -- "", utf8.codepoint("") == 65231, "\u{FECF}", INITIAL FOR
944 [65235] = utf8.char(65235), -- "", utf8.codepoint("") == 65235, "\u{FED3}", INITIAL FOR
945 [65239] = utf8.char(65239), -- "", utf8.codepoint("") == 65239, "\u{FED7}", INITIAL FOR
946 [65243] = utf8.char(65243), -- "", utf8.codepoint("") == 65243, "\u{FEDB}", INITIAL FOR
947 [65247] = utf8.char(65247), -- "", utf8.codepoint("") == 65247, "\u{FEDF}", INITIAL FOR
948 [65251] = utf8.char(65251), -- "", utf8.codepoint("") == 65251, "\u{FEE3}", INITIAL FOR
949 [65255] = utf8.char(65255), -- "", utf8.codepoint("") == 65255, "\u{FEE7}", INITIAL FOR
950 [65259] = utf8.char(65259), -- "", utf8.codepoint("") == 65259, "\u{FEEB}", INITIAL FOR
951 [65267] = utf8.char(65267), -- "", utf8.codepoint("") == 65267, "\u{FEF3}", INITIAL FOR
952 }
953
954 local peCharTableMedial = {
955 [1600] = utf8.char(1600), -- "", utf8.codepoint("") == 1600, "\u{0640}", ARABIC TATW
956 [64345] = utf8.char(64345), -- "", utf8.codepoint("") == 64345, "\u{FB59}", MEDIAL FORM
957 [64381] = utf8.char(64381), -- "", utf8.codepoint("") == 64381, "\u{FB7D}", MEDIAL FORM
958 [64401] = utf8.char(64401), -- "", utf8.codepoint("") == 64401, "\u{FB91}", MEDIAL FORM
959 [64405] = utf8.char(64405), -- "", utf8.codepoint("") == 64405, "\u{FB95}", MEDIAL FORM
960 [64425] = utf8.char(64425), -- "", utf8.codepoint("") == 64425, "\u{FBAB}", MEDIAL FORM
961 [64429] = utf8.char(64429), -- "", utf8.codepoint("") == 64429, "\u{FBAD}", MEDIAL FORM
962 [64511] = utf8.char(64511), -- "", utf8.codepoint("") == 64511, "\u{FBFF}", MEDIAL FORM
963 [65170] = utf8.char(65170), -- "", utf8.codepoint("") == 65170, "\u{FE92}", MEDIAL FORM
964 [65176] = utf8.char(65176), -- "", utf8.codepoint("") == 65176, "\u{FE98}", MEDIAL FORM
965 [65180] = utf8.char(65180), -- "", utf8.codepoint("") == 65180, "\u{FE9C}", MEDIAL FORM
966 [65184] = utf8.char(65184), -- "", utf8.codepoint("") == 65184, "\u{FEA0}", MEDIAL FORM
967 [65188] = utf8.char(65188), -- "", utf8.codepoint("") == 65188, "\u{FEA4}", MEDIAL FORM
968 [65192] = utf8.char(65192), -- "", utf8.codepoint("") == 65192, "\u{FEA8}", MEDIAL FORM
969 [65204] = utf8.char(65204), -- "", utf8.codepoint("") == 65204, "\u{FEB4}", MEDIAL FORM
970 [65208] = utf8.char(65208), -- "", utf8.codepoint("") == 65208, "\u{FEB8}", MEDIAL FORM

```

```

971 [65212] = utf8.char(65212), -- "", utf8.codepoint("") == 65212, "\u{FEB3}", MEDIAL FORM
972 [65216] = utf8.char(65216), -- "", utf8.codepoint("") == 65216, "\u{FEC0}", MEDIAL FORM
973 [65220] = utf8.char(65220), -- "", utf8.codepoint("") == 65220, "\u{FEC4}", MEDIAL FORM
974 [65224] = utf8.char(65224), -- "", utf8.codepoint("") == 65224, "\u{FEC8}", MEDIAL FORM
975 [65228] = utf8.char(65228), -- "", utf8.codepoint("") == 65228, "\u{FECC}", MEDIAL FORM
976 [65232] = utf8.char(65232), -- "", utf8.codepoint("") == 65232, "\u{FEDO}", MEDIAL FORM
977 [65236] = utf8.char(65236), -- "", utf8.codepoint("") == 65236, "\u{FED4}", MEDIAL FORM
978 [65240] = utf8.char(65240), -- "", utf8.codepoint("") == 65240, "\u{FED8}", MEDIAL FORM
979 [65244] = utf8.char(65244), -- "", utf8.codepoint("") == 65244, "\u{FEDC}", MEDIAL FORM
980 [65248] = utf8.char(65248), -- "", utf8.codepoint("") == 65248, "\u{FEE0}", MEDIAL FORM
981 [65252] = utf8.char(65252), -- "", utf8.codepoint("") == 65252, "\u{FEE4}", MEDIAL FORM
982 [65256] = utf8.char(65256), -- "", utf8.codepoint("") == 65256, "\u{FEE8}", MEDIAL FORM
983 [65260] = utf8.char(65260), -- "", utf8.codepoint("") == 65260, "\u{FEEC}", MEDIAL FORM
984 [65268] = utf8.char(65268), -- "", utf8.codepoint("") == 65268, "\u{FEF4}", MEDIAL FORM
985 }
986
987 local peCharTableFinal = {
988 [64343] = utf8.char(64343), -- "", utf8.codepoint("") == 64343, "\u{FB57}", FINAL FORM
989 [64379] = utf8.char(64379), -- "", utf8.codepoint("") == 64379, "\u{FB7B}", FINAL FORM
990 [64395] = utf8.char(64395), -- "", utf8.codepoint("") == 64395, "\u{FB8B}", FINAL FORM
991 [64399] = utf8.char(64399), -- "", utf8.codepoint("") == 64399, "\u{FB8F}", FINAL FORM
992 [64403] = utf8.char(64403), -- "", utf8.codepoint("") == 64403, "\u{FB93}", FINAL FORM
993 [64421] = utf8.char(64421), -- "", utf8.codepoint("") == 64421, "\u{FBA5}", FINAL FORM
994 [64509] = utf8.char(64509), -- "", utf8.codepoint("") == 64509, "\u{FBFD}", FINAL FORM
995 [65166] = utf8.char(65166), -- "", utf8.codepoint("") == 65166, "\u{FE8E}", FINAL FORM
996 [65168] = utf8.char(65168), -- "", utf8.codepoint("") == 65168, "\u{FE90}", FINAL FORM
997 [65172] = utf8.char(65172), -- "", utf8.codepoint("") == 65172, "\u{FE94}", FINAL FORM
998 [65174] = utf8.char(65174), -- "", utf8.codepoint("") == 65174, "\u{FE96}", FINAL FORM
999 [65178] = utf8.char(65178), -- "", utf8.codepoint("") == 65178, "\u{FE9A}", FINAL FORM
1000 [65182] = utf8.char(65182), -- "", utf8.codepoint("") == 65182, "\u{FE9E}", FINAL FORM
1001 [65186] = utf8.char(65186), -- "", utf8.codepoint("") == 65186, "\u{FEA2}", FINAL FORM
1002 [65190] = utf8.char(65190), -- "", utf8.codepoint("") == 65190, "\u{FEA6}", FINAL FORM
1003 [65194] = utf8.char(65194), -- "", utf8.codepoint("") == 65194, "\u{FEAA}", FINAL FORM
1004 [65196] = utf8.char(65196), -- "", utf8.codepoint("") == 65196, "\u{FEAC}", FINAL FORM
1005 [65198] = utf8.char(65198), -- "", utf8.codepoint("") == 65198, "\u{FEAE}", FINAL FORM
1006 [65200] = utf8.char(65200), -- "", utf8.codepoint("") == 65200, "\u{FEB0}", FINAL FORM
1007 [65202] = utf8.char(65202), -- "", utf8.codepoint("") == 65202, "\u{FEB2}", FINAL FORM
1008 [65206] = utf8.char(65206), -- "", utf8.codepoint("") == 65206, "\u{FEB6}", FINAL FORM
1009 [65210] = utf8.char(65210), -- "", utf8.codepoint("") == 65210, "\u{FEB8}", FINAL FORM
1010 [65214] = utf8.char(65214), -- "", utf8.codepoint("") == 65214, "\u{FEBE}", FINAL FORM
1011 [65218] = utf8.char(65218), -- "", utf8.codepoint("") == 65218, "\u{FEC2}", FINAL FORM
1012 [65222] = utf8.char(65222), -- "", utf8.codepoint("") == 65222, "\u{FEC6}", FINAL FORM
1013 [65226] = utf8.char(65226), -- "", utf8.codepoint("") == 65226, "\u{FECA}", FINAL FORM
1014 [65230] = utf8.char(65230), -- "", utf8.codepoint("") == 65230, "\u{FECE}", FINAL FORM
1015 [65234] = utf8.char(65234), -- "", utf8.codepoint("") == 65234, "\u{FED2}", FINAL FORM
1016 [65238] = utf8.char(65238), -- "", utf8.codepoint("") == 65238, "\u{FED6}", FINAL FORM
1017 [65242] = utf8.char(65242), -- "", utf8.codepoint("") == 65242, "\u{FEDA}", FINAL FORM
1018 [65246] = utf8.char(65246), -- "", utf8.codepoint("") == 65246, "\u{FEDE}", FINAL FORM
1019 [65250] = utf8.char(65250), -- "", utf8.codepoint("") == 65250, "\u{FEE2}", FINAL FORM
1020 [65254] = utf8.char(65254), -- "", utf8.codepoint("") == 65254, "\u{FEE6}", FINAL FORM
1021 [65258] = utf8.char(65258), -- "", utf8.codepoint("") == 65258, "\u{FEEA}", FINAL FORM
1022 [65262] = utf8.char(65262), -- "", utf8.codepoint("") == 65262, "\u{FEEE}", FINAL FORM
1023 [65264] = utf8.char(65264), -- "", utf8.codepoint("") == 65264, "\u{FEF0}", FINAL FORM
1024 [65266] = utf8.char(65266), -- "", utf8.codepoint("") == 65266, "\u{FEF2}", FINAL FORM

```

```

1025 [65276] = utf8.char(65276), -- "", utf8.codepoint("") == 65276, "\u{FEFC}", FINAL FORM
1026 }
1027
1028 return peCharTableInitial, peCharTableMedial, peCharTableFinal, peCharTableDiacritic
1029 --
1030 --
1031 -- End of file 'texnegar-char-table.lua'.
1032 </texnegar-char-table-lua>

```

1.8 File: texnegar.lua

```

1033 <*texnegar-lua>
1034 --
1035 -- This is file 'texnegar.lua',
1036 -- generated with the docstrip utility.
1037 --
1038 -- The original source files were:
1039 --
1040 -- texnegar.dtx (with options: 'texnegar-lua')
1041 --
1042 -- Copyright (C) 2020-2021 Hossein Movahhedian
1043 --
1044 -- It may be distributed and/or modified under the LaTeX Project Public License,
1045 -- version 1.3c or higher (your choice). The latest version of
1046 -- this license is at: http://www.latex-project.org/lppl.txt
1047 --
1048 -- texnegar          = texnegar or {}
1049 -- local texnegar    = texnegar
1050 -- texnegar.module   = {
1051 --   name             = "texnegar",
1052 --   version          = "0.1e",
1053 --   date             = "2021-02-09",
1054 --   description      = "Full implementation of kashida feature in XeLaTeX and LuaLaTeX",
1055 --   author           = "Hossein Movahhedian",
1056 --   copyright        = "Hossein Movahhedian",
1057 --   license          = "LPPL v1.3c"
1058 -- }
1059 --
1060 -- -- ^A%% texnegar-lua.dtx -- part of TEXNEGAR <bitbucket.org/dma8hm1334/texnegar>
1061 -- local err, warn, info, log = luatexbase.provides_module(texnegar.module)
1062 -- texnegar.log      = log or (function (s) luatexbase.module_info("texnegar", s) end)
1063 -- texnegar.warning = warn or (function (s) luatexbase.module_warning("texnegar", s) end)
1064 -- texnegar.error   = err or (function (s) luatexbase.module_error("texnegar", s) end)
1065
1066 local l_texnegar_kashida_fontfamily_bool = token.create("l_texnegar_kashida_fontfamily_bool")
1067
1068 local debug_getinfo = debug.getinfo
1069 local string_format = string.format
1070
1071 function TableLength(t)
1072   local i = 0
1073   for _ in pairs(t) do
1074     i = i + 1
1075   end
1076   return i

```

```

1077 end
1078
1079 tex.enableprimitives('',tex.extraprimitives ())
1080
1081 local range_tble = {
1082     [1536] = 1791,
1083     [1872] = 1919,
1084     [2208] = 2274,
1085     [8204] = 8297,
1086     [64336] = 65023,
1087     [65136] = 65279,
1088     [126464] = 126719,
1089     [983040] = 1048575
1090 }
1091
1092 local tbl_fonts_used = { }
1093 local tbl_fonts_chars = { }
1094 local tbl_fonts_chars_init = { }
1095 local tbl_fonts_chars_medi = { }
1096 local tbl_fonts_chars_fina = { }
1097
1098 local pattern_list = {
1099     "%.(ini)t?$", "%.(ini)t?%..*",
1100     "%.(med)i?$", "%.(med)i?%..*",
1101     "%.(fin)a?$", "%.(fin)a?%..*",
1102
1103     "%_(ini)t?$", "%_(ini)t?_.*",
1104     "%_(med)i?$", "%_(med)i?_.*",
1105     "%_(fin)a?$", "%_(fin)a?_.*",
1106 }
1107
1108 function GetFontsChars()
1109     local funcName = debug_getinfo(1).name
1110     local funcNparams = debug_getinfo(1).nparams
1111
1112     for f_num = 1, font.max() do
1113         local f_tmp = font.fonts[f_num]
1114         if f_tmp then
1115             local f_tmp_fontname = f_tmp.fontname
1116             if f_tmp_fontname then
1117                 local f_id_tmp = font.getfont(f_num)
1118                 local f_fontname_tmp = f_id_tmp.fontname
1119                 local f_filename_tmp = f_id_tmp.filename
1120                 if not tbl_fonts_used[f_fontname_tmp] then
1121                     tbl_fonts_used[f_fontname_tmp] = {f_filename_tmp, f_id_tmp}
1122                 end
1123             end
1124         end
1125     end
1126
1127     for f_fontname, v in pairs(tbl_fonts_used) do
1128         f_filename = v[1]
1129         f_id = v[2]
1130         if not tbl_fonts_chars[f_fontname] then

```



```

1131         tbl_fonts_chars[f_fontname] = { }
1132         tbl_fonts_chars_init[f_fontname] = { }
1133         tbl_fonts_chars_medi[f_fontname] = { }
1134         tbl_fonts_chars_fina[f_fontname] = { }
1135         local f = fontloader.open(f_filename)
1136         local char_name
1137         local char_unicode
1138         local char_class
1139         for k, v in pairs(range_tble) do
1140             for glyph_idx = k, v do
1141                 if f_id.characters[glyph_idx] then
1142                     char_name = f.glyphs[f_id.characters[glyph_idx].index].name
1143                     char_unicode = f.glyphs[f_id.characters[glyph_idx].index].unicode
1144                     char_class = f.glyphs[f_id.characters[glyph_idx].index].class
1145
1146                     kashida_fontfamily = token.get_macro("l_texnegar_kashida_fontfamily_
1147                     fontfamily_match = string.match(f_fontname, "^(" .. kashida_fontfami
1148                     if fontfamily_match == kashida_fontfamily then
1149                         if not tbl_fonts_chars[f_fontname][glyph_idx] then
1150                             if string.match(f_fontname, "^(Amiri).*") == "Amiri" and ch
1151                                 current_kashida_unicode = glyph_idx
1152                             end
1153                             tbl_fonts_chars[f_fontname][glyph_idx] = {char_name, char_un
1154                             for _, pattern in ipairs( pattern_list ) do
1155                                 local pos_alt = string.match(char_name, pattern)
1156                                 if pos_alt == 'ini' or pos_alt == 'AltIni' then
1157                                     tbl_fonts_chars_init[f_fontname][glyph_idx] = {char
1158                                 elseif pos_alt == 'med' or pos_alt == 'AltMed' then
1159                                     tbl_fonts_chars_medi[f_fontname][glyph_idx] = {char
1160                                 elseif pos_alt == 'fin' or pos_alt == 'AltFin' then
1161                                     tbl_fonts_chars_fina[f_fontname][glyph_idx] = {char
1162                                 end
1163                             end
1164                         end
1165                     end
1166                 end
1167             end
1168         end
1169         fontloader.close(f)
1170     end
1171 end
1172 return tbl_fonts_used, tbl_fonts_chars, tbl_fonts_chars_init, tbl_fonts_chars_medi, tbl_
1173 end
1174
1175 dofile(kpse.find_file("texnegar-ini.lua"))
1176 --
1177 --
1178 -- End of file 'texnegar.lua'.
1179 </texnegar-lua>

```

1.9 File: texnegar-ini.lua

```

1180 < *texnegar-ini-lua >
1181 --
1182 -- This is file 'texnegar-ini.lua',

```

```

1183 -- generated with the docstrip utility.
1184 --
1185 -- The original source files were:
1186 --
1187 -- texnegar.dtx (with options: 'texnegar-ini-lua')
1188 --
1189 -- Copyright (C) 2020-2021 Hossein Movahhedian
1190 --
1191 -- It may be distributed and/or modified under the LaTeX Project Public License,
1192 -- version 1.3c or higher (your choice). The latest version of
1193 -- this license is at: http://www.latex-project.org/lppl.txt
1194 --
1195 -- texnegar_ini = texnegar_ini or {}
1196 -- local texnegar_ini = texnegar_ini
1197 -- texnegar_ini.module = {
1198 --   name = "texnegar_ini",
1199 --   version = "0.1e",
1200 --   date = "2021-02-09",
1201 --   description = "Full implementation of kashida feature in XeLaTeX and LuaLaTeX",
1202 --   author = "Hossein Movahhedian",
1203 --   copyright = "Hossein Movahhedian",
1204 --   license = "LPPL v1.3c"
1205 -- }
1206 --
1207 -- ^^A%% texnegar-lua.dtx -- part of TEXNEGAR <bitbucket.org/dma8hm1334/texnegar>
1208 -- local err, warn, info, log = luatexbase.provides_module(texnegar_ini.module)
1209 -- texnegar_ini.log = log or (function (s) luatexbase.module_info("texnegar_ini", s)
1210 -- texnegar_ini.warning = warn or (function (s) luatexbase.module_warning("texnegar_ini", s)
1211 -- texnegar_ini.error = err or (function (s) luatexbase.module_error("texnegar_ini", s)
1212
1213 c_true_bool = token.create("c_true_bool")
1214
1215 l_texnegar_color_bool = token.create("l_texnegar_color_bool")
1216
1217 if l_texnegar_color_bool.mode == c_true_bool.mode then
1218   color_tbl = color_tbl or {}
1219   for item in l_texnegar_color_rgb_tl:gmatch("[^,%s]+") do
1220     table.insert(color_tbl, item)
1221   end
1222 end
1223
1224 dofile(kpse.find_file("texnegar-luatex-kashida.lua"))
1225 --
1226 --
1227 -- End of file 'texnegar-ini.lua'.
1228 </texnegar-ini-lua>

```

1.10 File: texnegar-luatex-kashida.lua

```

1229 <*texnegar-luatex-kashida-lua>
1230 --
1231 -- This is file 'texnegar-luatex-kashida.lua',
1232 -- generated with the docstrip utility.
1233 --
1234 -- The original source files were:

```

```

1235 --
1236 -- texnegar.dtx (with options: 'texnegar-luatex-kashida-lua')
1237 --
1238 -- Copyright (C) 2020-2021 Hossein Movahhedian
1239 --
1240 -- It may be distributed and/or modified under the LaTeX Project Public License,
1241 -- version 1.3c or higher (your choice). The latest version of
1242 -- this license is at: http://www.latex-project.org/lppl.txt
1243 --
1244 -- texnegar_luatex_kashida = texnegar_luatex_kashida or {}
1245 -- local texnegar_luatex_kashida = texnegar_luatex_kashida
1246 -- texnegar_luatex_kashida.module = {
1247 --     name = "texnegar_luatex_kashida",
1248 --     version = "0.1e",
1249 --     date = "2021-02-09",
1250 --     description = "Full implementation of kashida feature in XeLaTeX and L
1251 --     author = "Hossein Movahhedian",
1252 --     copyright = "Hossein Movahhedian",
1253 --     license = "LPPL v1.3c"
1254 -- }
1255 --
1256 -- -- ^A%% texnegar-lua.dtx -- part of TEXNEGAR <bitbucket.org/dma8hm1334/texnegar>
1257 -- local err, warn, info, log = luatexbase.provides_module(texnegar_luatex_kashida.module)
1258 -- texnegar_luatex_kashida.log = log or (function (s) luatexbase.module_info("texnegar_
1259 -- texnegar_luatex_kashida.warning = warn or (function (s) luatexbase.module_warning("texnegar
1260 -- texnegar_luatex_kashida.error = err or (function (s) luatexbase.module_error("texnegar
1261
1262 local peCharTableInitial, peCharTableMedial, peCharTableFinal, peCharTableDiacritic = dofile(
char-table.lua"))
1263
1264 local kashida_unicode = 1600
1265 local kashida_subtype = 256
1266
1267 local COLORSTACK = node.subtype("pdf_colorstack")
1268 local node_id = node.id
1269 local GLUE = node_id("glue")
1270 local GLYPH = node_id("glyph")
1271 local HLIST = node_id("hlist")
1272 local RULE = node_id("rule")
1273 local VLIST = node_id("vlist")
1274 local WHATSIT = node_id("whatsit")
1275
1276 local l_texnegar_kashida_glyph_bool = token.create("l_texnegar_kashida_glyph_bool")
1277 local l_texnegar_kashida_leaders_glyph_bool = token.create("l_texnegar_kashida_leaders_glyph
1278 local l_texnegar_kashida_leaders_hrrule_bool = token.create("l_texnegar_kashida_leaders_hrrule
1279
1280 local l_texnegar_hboxrecursion_bool = token.create("l_texnegar_hboxrecursion_bool")
1281 local l_texnegar_vboxrecursion_bool = token.create("l_texnegar_vboxrecursion_bool")
1282
1283 local selected_font = font.current()
1284 local selected_font_old = selected_font
1285
1286 local string_format = string.format
1287 local debug_getinfo = debug.getinfo

```

```

1288
1289 function GetGlyphDimensions(font_file, glyph_index)
1290     local funcName      = debug_getinfo(1).name
1291     local funcNparams   = debug_getinfo(1).nparams
1292
1293     local fnt = fontloader.open(font_file)
1294     local idx = 0
1295     local fnt_glyphcnt = fnt.glyphcnt
1296     local fnt_glyphmin = fnt.glyphmin
1297     local fnt_glyphmax = fnt.glyphmax
1298     if fnt_glyphcnt > 0 then
1299         for idx = fnt_glyphmin, fnt_glyphmax do
1300             local gl = fnt.glyphs[idx]
1301             if gl then
1302                 local gl_unicode = gl.unicode
1303                 if gl_unicode == glyph_index then
1304                     local gl_name      = gl.name
1305                     local gl_width     = gl.width
1306                     local gl_bbox     = gl.boundingBox
1307                     local gl_llx      = gl_bbox[1]
1308                     local gl_depth    = gl_bbox[2]
1309                     local gl_urx      = gl_bbox[3]
1310                     local gl_height   = gl_bbox[4]
1311                     break
1312                 end
1313             end
1314             idx = idx + 1
1315         end
1316     end
1317     fontloader.close(fnt)
1318     return {width = gl_width, height = gl_height, depth = gl_depth, llx = gl_llx, urx = gl_urx}
1319 end
1320
1321 function GetGlue(t_plb_line_glue_node, t_plb_node)
1322     local funcName      = debug_getinfo(1).name
1323     local funcNparams   = debug_getinfo(1).nparams
1324
1325     local glue_id       = t_plb_line_glue_node.id
1326     local glue_subtype  = t_plb_line_glue_node.subtype
1327     local glue_width    = t_plb_line_glue_node.width
1328     local glue_stretch  = t_plb_line_glue_node.stretch
1329     local glue_shrink   = t_plb_line_glue_node.shrink
1330     local eff_glue_width = node.effective_glue(t_plb_line_glue_node, t_plb_node)
1331     local glue_stretch_order = t_plb_line_glue_node.stretch_order
1332     local glue_shrink_order = t_plb_line_glue_node.shrink_order
1333     local glue_delta    = 0
1334     glue_delta = eff_glue_width - glue_width
1335     return { id = glue_id, subtype = glue_subtype, width = glue_width, stretch = glue_stretch,
1336             shrink = glue_shrink, stretch_order = glue_stretch_order, shrink_order = glue_shrink_order,
1337             effective_glue = eff_glue_width, delta = glue_delta }
1338 end
1339
1340 function GetGlyph(t_plb_line_glyph_node, t_tbl_line_fields, t_CharTableInitial, t_CharTableM
1341     local funcName      = debug_getinfo(1).name

```

```

1342     local funcNparams = debug_getinfo(1).nparams
1343
1344     local glyph_id      = t_plb_line_glyph_node.id
1345     local glyph_subtype = t_plb_line_glyph_node.subtype
1346     local glyph_char    = t_plb_line_glyph_node.char
1347     local glyph_font    = t_plb_line_glyph_node.font
1348     local glyph_lang    = t_plb_line_glyph_node.lang
1349     local glyph_width   = t_plb_line_glyph_node.width
1350     local glyph_data    = t_plb_line_glyph_node.data
1351
1352     if not (t_CharTableInitial[glyph_char] == nil) then
1353         t_tbl_line_fields.joinerCharInitial = t_tbl_line_fields.joinerCharInitial + 1
1354         t_plb_line_glyph_node.data = 1
1355     elseif not (t_CharTableMedial[glyph_char] == nil) then
1356         t_tbl_line_fields.joinerCharMedial = t_tbl_line_fields.joinerCharMedial + 1
1357         t_plb_line_glyph_node.data = 2
1358     elseif not (t_CharTableFinal[glyph_char] == nil) then
1359         t_tbl_line_fields.joinerCharFinal = t_tbl_line_fields.joinerCharFinal + 1
1360         t_plb_line_glyph_node.data = 3
1361     end
1362     return { id = glyph_id, subtype = glyph_subtype, char = glyph_char, font = glyph_font, }
1363 end
1364
1365 function ProcessTableKashidaHlist(ksh_hlistNode, hbox_num, in_font)
1366     local funcName      = debug_getinfo(1).name
1367     local funcNparams = debug_getinfo(1).nparams
1368
1369     local ksh_hlistNode_id      = ksh_hlistNode.id
1370     local ksh_hlistNode_subtype = ksh_hlistNode.subtype
1371
1372     for tn in node.traverse(ksh_hlistNode.head) do
1373         local tn_id = tn.id
1374         local tn_subtype = tn.subtype
1375
1376         if tn_id == HLIST then
1377             for tp in node.traverse(tn.head) do
1378                 local tp_id = tp.id
1379                 local tp_subtype = tp.subtype
1380                 if tp_id == GLYPH then
1381                     if l_texnegar_color_bool.mode == c_true_bool.mode then
1382                         local col_str      = color_tbl[1] .. " " .. color_tbl[2] .. " " .. c
1383                         local col_str_rg   = col_str .. " rg "
1384                         local col_str_RG   = col_str .. " RG"
1385
1386                         local color_push   = node.new(WHATSIT, COLORSTACK)
1387                         local color_pop    = node.new(WHATSIT, COLORSTACK)
1388                         color_push.stack   = 0
1389                         color_pop.stack    = 0
1390                         color_push.command = 1
1391                         color_pop.command  = 2
1392                         glue_ratio        = .2
1393                         color_push.data    = col_str_rg .. col_str_RG
1394                         color_pop.data     = col_str_rg .. col_str_RG
1395                         tn.head = node.insert_before(tn.list, tn.head, node.copy(color_push))

```

```

1396         tn.head = node.insert_after(tn.list, node.tail(tn.head), node.copy(c
1397     end
1398
1399     local tp_font = tp.font
1400     local tp_char = tp.char
1401     tp.font = in_font
1402
1403     local ksh_unicode
1404     ksh_unicode = font.getfont(in_font).resources.unicodes['kashida']
1405     if hbox_num == 'l_texnegar_k_box' then
1406         tp.char = current_kashida_unicode or kashida_unicode
1407     elseif hbox_num == 'l_texnegar_ksh_box' then
1408         tp.char = ksh_unicode
1409         tn_width = tn.width
1410         ksh_hlistNode.width = tn_width
1411     end
1412     elseif tp_id == HLIST then
1413         if tp.subtype ~= 3 then
1414             tbl_kashida_hlist_nodes[ #tbl_kashida_hlist_nodes + 1 ] = tp
1415         end
1416     end
1417     end
1418 elseif tn_id == VLIST then
1419     do end
1420 elseif tn_id == WHATSIT then
1421     do end
1422 elseif tn_id == GLYPH then
1423     if l_texnegar_color_bool.mode == c_true_bool.mode then
1424         local col_str      = color_tbl[1] .. " " .. color_tbl[2] .. " " .. color_tbl
1425         local col_str_rg   = col_str .. " rg "
1426         local col_str_RG   = col_str .. " RG"
1427
1428         local color_push   = node.new(WHATSIT, COLORSTACK)
1429         local color_pop    = node.new(WHATSIT, COLORSTACK)
1430         color_push.stack   = 0
1431         color_pop.stack    = 0
1432         color_push.command = 1
1433         color_pop.command  = 2
1434         glue_ratio        = .2
1435         color_push.data    = col_str_rg .. col_str_RG
1436         color_pop.data     = col_str_rg .. col_str_RG
1437         ksh_hlistNode.head = node.insert_before(ksh_hlistNode.list, ksh_hlistNode.he
1438         ksh_hlistNode.head = node.insert_after(ksh_hlistNode.list, node.tail(ksh_hli
1439     end
1440
1441     local tn_font = tn.font
1442     local tn_char = tn.char
1443     tn.font = in_font
1444
1445     local ksh_unicode
1446     ksh_unicode = font.getfont(in_font).resources.unicodes['kashida']
1447     if hbox_num == 'l_texnegar_k_box' then
1448         tn.char = kashida_unicode
1449     elseif hbox_num == 'l_texnegar_ksh_box' then

```

```

1450         tn.char = ksh_unicode
1451         tn.width = tn.width
1452         ksh_hlistNode.width = tn_width
1453     end
1454 else
1455     print(string_format("\n tn. Not processed node id is: %d", tn_id))
1456 end
1457 end
1458 end
1459
1460 function SetFontInHbox(hbox_num, font_num)
1461     local funcName = debug_getinfo(1).name
1462     local funcNparams = debug_getinfo(1).nparams
1463
1464     tbl_kashida_hlist_nodes = {}
1465
1466     local tmp_node
1467     tmp_node = node.new("hlist")
1468     tmp_node = tex.getbox(hbox_num)
1469
1470     ProcessTableKashidaHlist(tmp_node, hbox_num, font_num)
1471
1472     ::kashida_hlist_BEGIN::
1473     if #tbl_kashida_hlist_nodes > 0 then
1474         local kashida_hlistNodeAdded = table.remove(tbl_kashida_hlist_nodes,1)
1475         ProcessTableKashidaHlist(kashida_hlistNodeAdded, hbox_num, font_num)
1476         goto kashida_hlist_BEGIN
1477     end
1478 end
1479
1480 function StretchGlyph(t_plb_node, t_plb_glyph_node, t_gluePerJoiner, t_dir, t_filler)
1481     local funcName = debug_getinfo(1).name
1482     local funcNparams = debug_getinfo(1).nparams
1483
1484     if t_filler == "resized_kashida" then
1485         SetFontInHbox('l_texnegar_k_box', selected_font)
1486     elseif t_filler == "leaders+kashida" then
1487         SetFontInHbox('l_texnegar_ksh_box', selected_font)
1488     end
1489
1490     kashida_node = node.new(GLYPH)
1491     node_glue = node.new(GLUE)
1492     node_rule = node.new(RULE)
1493     node_hlist = node.new(HLIST)
1494
1495     font_current = selected_font
1496     font_name = font.fonts[font_current].fullname
1497     font_file = font.fonts[font_current].filename
1498     kashida_char = font.fonts[font_current].characters[1600]
1499
1500     kashida_node.subtype = kashida_subtype
1501     kashida_node.font = font_current
1502     if string.match(font_name, "(Amiri).*") == "Amiri" then
1503         kashida_node.char = current_kashida_unicode

```

```

1504     else
1505         kashida_node.char = kashida_unicode
1506     end
1507     kashida_node.lang      = tex.language
1508
1509     kashida_width  = kashida_node.width
1510     kashida_height = kashida_node.height
1511     kashida_depth  = kashida_node.depth
1512
1513     tbl_gl_dimen = GetGlyphDimensions(font_file, kashida_unicode)
1514     ksh_width, ksh_height, ksh_depth, ksh_llx, ksh_urx =
1515         tbl_gl_dimen.width, tbl_gl_dimen.height, tbl_gl_dimen.depth, tbl_gl_dimen.llx, tbl_g
1516
1517     ratio_width = kashida_width / ksh_width
1518     leaders_height = ratio_width * ksh_height
1519     leaders_depth = - ratio_width * ksh_depth
1520
1521     node_glue.subtype = 100
1522     node.setglue(node_glue, t_gluePerJoiner, 0, 0, 0, 0)
1523
1524     if t_filler == "resized_kashida" then
1525         node_glue.leader = node.copy_list(tex.box['l_texnegar_k_box'])
1526     elseif t_filler == "leaders+kashida" then
1527         node_glue.leader = node.copy_list(tex.box['l_texnegar_ksh_box'])
1528     elseif t_filler == "leaders+hrule" then
1529         node_glue.leader = node_rule
1530     end
1531
1532     node_glue.leader.subtype = 0
1533     node_glue.leader.height  = leaders_height
1534     node_glue.leader.depth   = leaders_depth
1535
1536     node_glue.leader.dir     = t_dir
1537
1538     local t_plb_glyph_node_next = t_plb_glyph_node.next
1539     local t_plb_glyph_node_next_id = t_plb_glyph_node_next.id
1540     if not t_plb_glyph_node_next then
1541         node.insert_after(t_plb_node.list, t_plb_glyph_node, node_glue)
1542     else
1543         if t_plb_glyph_node_next_id == GLYPH then
1544             local t_plb_glyph_node_next_char = t_plb_glyph_node_next.char
1545             if peCharTableDiacritic[t_plb_glyph_node_next_char] then
1546                 node.insert_after(t_plb_node.list, t_plb_glyph_node_next, node_glue)
1547             else
1548                 node.insert_after(t_plb_node.list, t_plb_glyph_node, node_glue)
1549             end
1550         else
1551             node.insert_after(t_plb_node.list, t_plb_glyph_node, node_glue)
1552         end
1553     end
1554     if t_filler == "leaders+hrule" then
1555         for tn in node.traverse(t_plb_node.head) do
1556             local tn_id = tn.id
1557             local tn_subtype = tn.subtype

```



```

1558
1559         if tn_id == GLUE and tn_subtype == 100 then
1560             local t_hbox = node.new(HLIST)
1561             local t_hrulerule = node.copy(tn)
1562
1563             if string.match(font_name, "^(Amiri).*") == "Amiri" then
1564                 t_hrulerule.leader.height = kashida_height
1565                 t_hrulerule.leader.depth = kashida_depth
1566             end
1567
1568             t_hbox.head = node.insert_after(t_hbox.list, t_hbox.head, t_hrulerule)
1569             t_plb_node.head = node.insert_after(t_plb_node.list, tn, t_hbox)
1570
1571             if l_texnegar_color_bool.mode == c_true_bool.mode then
1572                 local col_str = color_tbl[1] .. " " .. color_tbl[2] .. " " .. color
1573                 local col_str_rg = col_str .. " rg "
1574                 local col_str_RG = col_str .. " RG"
1575
1576                 local color_push = node.new(WHATSIT, COLORSTACK)
1577                 local color_pop = node.new(WHATSIT, COLORSTACK)
1578                 color_push.stack = 0
1579                 color_pop.stack = 0
1580                 color_push.command = 1
1581                 color_pop.command = 2
1582                 glue_ratio = .2
1583                 color_push.data = col_str_rg .. col_str_RG
1584                 color_pop.data = col_str_rg .. col_str_RG
1585                 t_hbox.head = node.insert_before(t_hbox.list, t_hbox.head, node.copy(col
1586                 t_hbox.head = node.insert_after(t_hbox.list, node.tail(t_hbox.head), nod
1587             end
1588         end
1589     end
1590 end
1591 end
1592
1593 function GetFillerSpec(t_plb_node, t_plb_head_node, t_tbl_line_fields, t_CharTableInitial, t
1594     local funcName = debug_getinfo(1).name
1595     local funcNparams = debug_getinfo(1).nparams
1596
1597     t_plb_node_id = t_plb_node.id
1598     t_plb_node_subtype = t_plb_node.subtype
1599
1600     for p in node.traverse(t_plb_head_node) do
1601         local p_id = p.id
1602         local p_subtype = p.subtype
1603         if p_id == HLIST then
1604             t_tbl_line_fields.lineWidthRemainder = t_tbl_line_fields.lineWidthRemainder - p
1605             if p_subtype ~= 3 then
1606                 tbl_hlist_nodes[ #tbl_hlist_nodes + 1 ] = p
1607             end
1608         elseif p_id == VLIST then
1609             t_tbl_line_fields.lineWidthRemainder = t_tbl_line_fields.lineWidthRemainder - p
1610             tbl_vlist_nodes[ #tbl_vlist_nodes + 1 ] = p
1611         elseif p_id == GLUE then

```

```

1612         tbl_p_glue = GetGlue(p, t_plb_node)
1613         t_tbl_line_fields.lineWidthRemainder = t_tbl_line_fields.lineWidthRemainder - tbl_p_glue
1614         t_tbl_line_fields.total_glues = t_tbl_line_fields.total_glues + 1
1615         t_tbl_line_fields.stretchedGlue = t_tbl_line_fields.stretchedGlue + tbl_p_glue["width"]
1616     elseif p_id == GLYPH then
1617         tbl_p_glyph, t_tbl_line_fields = GetGlyph(p, t_tbl_line_fields, t_CharTableInit)
1618         selected_font_old = selected_font
1619         selected_font = tbl_p_glyph["font"]
1620         t_tbl_line_fields.lineWidthRemainder = t_tbl_line_fields.lineWidthRemainder - tbl_p_glyph["width"]
1621         t_tbl_line_fields.total_glyphs = t_tbl_line_fields.total_glyphs + 1
1622     end
1623 end
1624
1625 t_tbl_line_fields.total_joiners = t_tbl_line_fields.joinerCharInitial + t_tbl_line_fields.joinerCharMedial + t_tbl_line_fields.joinerCharFinal
1626 t_tbl_line_fields.gluePerJoiner = 0
1627 if t_tbl_line_fields.total_glues == 0 then
1628     t_tbl_line_fields.stretchedGlue = t_tbl_line_fields.lineWidthRemainder
1629 end
1630 if t_tbl_line_fields.total_joiners > 0 then
1631     t_tbl_line_fields.gluePerJoiner = t_tbl_line_fields.stretchedGlue // t_tbl_line_fields.total_joiners
1632     t_tbl_line_fields.stretchedGlueRemainder = t_tbl_line_fields.stretchedGlue % t_tbl_line_fields.total_joiners
1633 elseif t_tbl_line_fields.total_joiners == 1 then
1634     t_tbl_line_fields.gluePerJoiner = t_tbl_line_fields.stretchedGlue
1635 end
1636
1637 return t_tbl_line_fields
1638 end
1639
1640 function ProcessTableHlist(tmphl_n)
1641     local funcName = debug_getinfo(1).name
1642     local funcNparams = debug_getinfo(1).nparams
1643
1644     local tmphl_n_id = tmphl_n.id
1645     local tmphl_n_subtype = tmphl_n.subtype
1646
1647     local tbl_line_fields = { line_dir = "", line_width = 0, lineWidthRemainder = 0,
1648                             joinerCharInitial = 0, joinerCharMedial = 0, joinerCharFinal = 0,
1649                             stretchedGlue = 0, total_glues = 0, gluePerJoiner = 0 }
1650
1651     local tbl_p_glue, tbl_p_glyph
1652
1653     if (tmphl_n_id == HLIST) and (tmphl_n_subtype == 1 or tmphl_n_subtype == 2) then
1654         tbl_line_fields.line_width = tmphl_n.width
1655         tbl_line_fields.line_dir = tmphl_n.dir
1656         tbl_line_fields.lineWidthRemainder = tbl_line_fields.line_width
1657
1658         if tbl_line_fields.line_dir == "TLT" then
1659             tbl_line_fields = GetFillerSpec(tmphl_n, tmphl_n.head, tbl_line_fields, peCharTable)
1660
1661             if tbl_line_fields.total_joiners == 0 or tbl_line_fields.gluePerJoiner == 0 or
1662                tbl_line_fields.stretchedGlue == 0 then
1663                 goto continue
1664             end
1665
1666             for q in node.traverse_id(GLUE, tmphl_n.head) do

```

```

1666         local eff_glue_width      = node.effective_glue(q, tmp_hl_n)
1667         node.setglue(q, q.width, 0, 0, q.stretch_order, q.glue_shrink_order)
1668     end
1669
1670     for r in node.traverse_id(GLYPH, tmp_hl_n.head) do
1671         local r_data = r.data
1672         if r_data == 1 or r.data == 2 then
1673             StretchGlyph(tmp_hl_n, r, tbl_line_fields.gluePerJoiner, tbl_line_fields.
1674         elseif r.data == 3 then
1675             goto for_loop_01
1676         end
1677         ::for_loop_01::
1678     end
1679     tbl_line_fields.line_width = tmp_hl_n.width
1680     tbl_line_fields.lineWidthRemainder = line_width
1681 elseif tbl_line_fields.line_dir == "TRT" then
1682     tbl_line_fields = GetFillerSpec(tmp_hl_n, tmp_hl_n.head, tbl_line_fields, peCharTa
1683     if tbl_line_fields.total_joiners == 0 or tbl_line_fields.gluePerJoiner == 0 or
1684         goto continue
1685     end
1686
1687     for q in node.traverse_id(GLUE, tmp_hl_n.head) do
1688         local eff_glue_width      = node.effective_glue(q, tmp_hl_n)
1689         node.setglue(q, q.width, 0, 0, q.stretch_order, q.glue_shrink_order)
1690     end
1691
1692     for r in node.traverse_id(GLYPH, tmp_hl_n.head) do
1693         local r_data = r.data
1694         if r_data == 1 or r.data == 2 then
1695             StretchGlyph(tmp_hl_n, r, tbl_line_fields.gluePerJoiner, tbl_line_fields.
1696         elseif r.data == 3 then
1697             goto for_loop_02
1698         end
1699         ::for_loop_02::
1700     end
1701     tbl_line_fields.line_width = tmp_hl_n.width
1702     tbl_line_fields.lineWidthRemainder = line_width
1703 else
1704     print(string_format("\n Line direction '%s' is not supported yet!", tbl_line_fie
1705 end
1706 end
1707 ::continue::
1708 end
1709
1710 function ProcessTableVlist(tmp_vl_n)
1711     local funcName      = debug_getinfo(1).name
1712     local funcNparams   = debug_getinfo(1).nparams
1713
1714     local tmp_vl_n_id   = tmp_vl_n.id
1715     local tmp_vl_n_subtype = tmp_vl_n.subtype
1716
1717     for vbNode in node.traverse(tmp_vl_n) do
1718         if vbNode.id == VLIST and vbNode.subtype == 0 then
1719             for tr_vbNode in node.traverse(vbNode.head) do

```

```

1720             if (tr_vbNode.id == HLIST) and (tr_vbNode.subtype == 1 or tr_vbNode.subtype
1721                 ProcessTableHlist(tr_vbNode)
1722             end
1723         end
1724     end
1725 end
1726 end
1727
1728 function PostLineBreakFilter(hboxes_stack, groupcode)
1729     local funcName    = debug_getinfo(1).name
1730     local funcNparams = debug_getinfo(1).nparams
1731
1732     funcName = "PostLineBreakFilter"
1733
1734     local tbl_fonts_used = { }
1735     local tbl_fonts_chars = { }
1736     local tbl_fonts_chars_init = { }
1737     local tbl_fonts_chars_medi = { }
1738     local tbl_fonts_chars_fina = { }
1739
1740     tbl_fonts_used, tbl_fonts_chars, tbl_fonts_chars_init, tbl_fonts_chars_medi, tbl_fonts_c
1741
1742     local f_fontname
1743
1744     for f_fontname, v in pairs(tbl_fonts_used) do
1745         for k1, v1 in pairs(tbl_fonts_chars_init[f_fontname]) do
1746             if k1 and not peCharTableInitial[k1] then
1747                 peCharTableInitial[k1] = utf8.char(k1)
1748             end
1749         end
1750
1751         for k1, v1 in pairs(tbl_fonts_chars_medi[f_fontname]) do
1752             if k1 and not peCharTableMedial[k1] then
1753                 peCharTableMedial[k1] = utf8.char(k1)
1754             end
1755         end
1756
1757         for k1, v1 in pairs(tbl_fonts_chars_fina[f_fontname]) do
1758             if k1 and not peCharTableFinal[k1] then
1759                 peCharTableFinal[k1] = utf8.char(k1)
1760             end
1761         end
1762     end
1763
1764     tbl_hlist_nodes = {}
1765     tbl_vlist_nodes = {}
1766     for hlistNode in node.traverse(hboxes_stack) do
1767         if node.next(hlistNode) == nil then
1768             goto END
1769         end
1770
1771         ProcessTableHlist(hlistNode)
1772
1773         if l_texnegar_hboxrecursion_bool.mode == c_true_bool.mode then

```

```

1774         ::hboxBEGIN::
1775         if #tbl_hlist_nodes > 0 then
1776             local hlistNodeAdded = table.remove(tbl_hlist_nodes,1)
1777             ProcessTableHlist(hlistNodeAdded)
1778             goto hboxBEGIN
1779         end
1780     end
1781
1782     if l_texnegar_vboxrecursion_bool.mode == c_true_bool.mode then
1783         ::vboxBEGIN::
1784         if #tbl_vlist_nodes > 0 then
1785             local vlistNodeAdded = table.remove(tbl_vlist_nodes,1)
1786             ProcessTableVlist(vlistNodeAdded)
1787             goto vboxBEGIN
1788         end
1789     end
1790
1791     ::END::
1792 end
1793 return hboxes_stack
1794 end
1795
1796 if l_texnegar_kashida_glyph_bool.mode == c_true_bool.mode then
1797     filler_pe = "resized_kashida"
1798 elseif l_texnegar_kashida_leaders_glyph_bool.mode == c_true_bool.mode then
1799     filler_pe = "leaders+kashida"
1800 elseif l_texnegar_kashida_leaders_hrulerule_bool.mode == c_true_bool.mode then
1801     filler_pe = "leaders+hrulerule"
1802 else
1803     print(string_format" Unknown kashida value.")
1804 end
1805
1806 function StartStretching()
1807     if not luatexbase.in_callback('post_linebreak_filter', 'insertKashida') then
1808         luatexbase.add_to_callback('post_linebreak_filter', PostLineBreakFilter, 'insertKashida')
1809     end
1810 end
1811
1812 function StopStretching()
1813     if luatexbase.in_callback('post_linebreak_filter', 'insertKashida') then
1814         luatexbase.remove_from_callback('post_linebreak_filter', 'insertKashida')
1815     end
1816 end
1817 --
1818 --
1819 -- End of file 'texnegar-luatex-kashida.lua'.
1820 </texnegar-luatex-kashida-lua>

```

2 Acknowledgments

In the first place I have to thank Donald Knuth for inventing TeX. During the development of this package I referred to Stack Exchange network of question-and-answer (Q&A) websites to solve problems for which I am grateful. I also would like to thank the developer teams of TeX's friends especially LaTeX, LuaTeX and XeTeX teams.

3 Change History

2020-08-29 v0.1a

- First standalone version.

2020-08-30 v0.1b

- Changed some file names.

2021-01-27 v0.1c

- Added the option `Minimal` which is needed if `texnegar` is used for kashida implementation only.
- Fixed the problem with `Scheherazade` and `Amiri` fonts.

To Do's

To do

References:

(Actually, this is not a “References” nor a “Literature”, but the most important although not a complete list of “Resources Used” to develop this package.)

- [1] Donald E. Knuth, *The T_EX book*, Addison-Wesley, 1986.
- [2] Victor Eijkhout, *T_EX BY TOPIC*, Addison-Wesley, 2013.
- [3] Paul W. Abrahams, Kathryn A. Hargreaves, and Karl Berry, *T_EX for the Impatient*, Addison-Wesley, 2013.
- [4] Leslie Lamport, *L^AT_EX, A document preparation System*, Addison-Wesley, 1986.
- [5] Frank Mittelbach and Michel Goossens with Johannes Braams, David Carlisle, and Chris Rowley, *The L^AT_EX Companion*, Addison-Wesley, second edition, 2004.
- [6] Roberto Ierusalimsky, *Programming in Lua*, Lua.org, fourth edition, 2016
- [7] Lua.org, *Lua 5.3 Reference Manual*, Lua.org, 2016
- [8] Package `latex`: The LaTeX Team, *The L^AT_EX 2_ε Sources*, [CTAN:macros/latex/base/source2e.pdf](#), 2020-02-02
- [9] Package `l3kernel`: The LaTeX3 Team, *The L^AT_EX 3 Sources*, [CTAN:macros/latex/contrib/l3kernel/source3.pdf](#), 2020-07-17
- [10] Package `l3kernel`: The LaTeX3 Team, *The L^AT_EX 3 Interfaces*, [CTAN:macros/latex/contrib/l3kernel/interface3.pdf](#), 2020-07-17
- [11] Package `luatex`: The LuaTeX Team, *LuaT_EX Reference Manual*, [CTAN:systems/doc/luatex/luatex.pdf](#), 2020

- [12] Package `xetexref`: Will Robertson, Khaled Hosny, and Karl Berry, X_ƎTeX reference guide, [CTAN:info/xetexref/xetex-reference.pdf](http://ctan.org/info/xetexref/xetex-reference.pdf), 2019-12-09
- [13] Package `xetex`: Jonathan Kew, About X_ƎTeX, [CTAN:systems/doc/xetex/XeTeX-notes.pdf](http://ctan.org/systems/doc/xetex/XeTeX-notes.pdf), 2005-10-17
- [14] Package `xetex`: Michel Goossens, The X_ƎTeX Companion, <http://xml.web.cern.ch/XML/lgc2/xetexmain.pdf>, 2009-08-19
- [15] Website: Stack Exchange: Hot Questions, T_EX-L^AT_EX Q&A for users of TeX, LaTeX, ConTeXt, and related typesetting systems, tex.stackexchange.com
- [16] Website: LuaTeX Wiki, LuaT_EX Wiki, wiki.luatex.org

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols			
<code>\[</code>	669		
<code>\]</code>	25		
<code>_</code>	669		
Numbers			
<code>\0</code>	583		
<code>\1</code>	317, 669		
<code>\2</code>	669		
<code>_</code>	542, 583, 669		
A			
<code>\AtBeginDocument</code>	55, 313		
B			
bool commands:			
<code>\bool_if:NTF</code>	38, 51, 69, 494, 510, 521, 546, 603, 617, 662, 736, 741, 762, 767		
<code>\bool_if:nTF</code>	532, 672, 756		
<code>\bool_set_false:N</code>	131, 135, 137, 141, 142, 143, 145, 146, 147, 148, 149, 279, 295, 337, 449, 455, 466, 472, 488		
<code>\bool_set_true:N</code> ...	282, 299, 319, 320, 325, 326, 331, 332, 342, 343, 360, 417, 431, 453, 457, 470, 474, 487		
box commands:			
<code>\box_new:N</code>	81, 82		
C			
<code>\c</code>	583		
<code>\char</code>	46, 542		
		clist commands:	
		<code>\clist_count:N</code>	525
		<code>\clist_item:Nn</code>	530, 531
		<code>\clist_map_inline:Nn</code>	537, 705, 711, 717, 723, 729
		<code>\clist_new:N</code>	523
		<code>\clist_put_left:Nn</code>	534
		<code>\clist_set:Nn</code>	212, 213, 214, 215, 216, 217, 218, 219, 220, 222, 531, 704, 710, 716, 722, 728
		<code>\l_tmpa_clist</code>	531, 534
		<code>\color</code>	610
		<code>\colorlet</code>	605, 608
		<code>\convertcolorspec</code>	434
		<code>\copy</code>	676
		cs commands:	
		<code>\cs:w</code>	631, 641, 647, 651
		<code>\cs_end:</code>	631, 641, 647, 651
		<code>\cs_new:Nn</code>	549
		<code>\cs_new:Npn</code>	615, 660
		<code>\cs_new_protected:Nn</code>	580
D			
<code>\def</code>	79		
		dim commands:	
		<code>\dim_compare:nTF</code>	566, 653
		<code>\dim_eval:n</code>	651
		<code>\dim_new:N</code>	129
		<code>\dim_set:Nn</code>	649
		<code>\directlua</code>	48, 437, 481, 482, 512
		<code>\discouragebadlinebreaks</code>	571

	E	lua commands:
else commands:		\lua_now:n 46
\else: 562, 648		\luatexversion 40
\endinput . 16, 21, 31, 61, 74, 516, 588, 786		
exp commands:		M
\exp_after:wN 631, 641, 645		\makeatletter 502
	F	\makeatother 508
fi commands:		\MessageBreak 558, 560
\fi: 42, 568, 656		mode commands:
	G	\mode_leave_vertical: 620
\gdef 631, 641		msg commands:
\GenericError 551		\msg_error:nn 346
	H	\msg_error:nnn 680, 782
hbox commands:		\msg_error:nnnn 41
\hbox_set:Nn 44, 46, 674		\msg_fatal:nn 29
\height 44, 655		\msg_new:nnn 23, 234, 239, 244, 249, 255, 265
	I	\msg_warning:nnn 311
if commands:		N
\if_int_compare:w 40		Negar commands:
\if_meaning:w 646		Negar: 1
\if_mode_vertical: 550		\NewDocumentCommand 481, 482, 487, 488, 571
\IfNoValueF 573, 575		\newif 503, 507
\input 513		\newXeTeXintercharclass 593, 594, 595, 596, 597
int commands:		P
\l_fontnumber_int 109		\ProcessKeysOptions 477
\int_case:nnTF 352		\ProvidesExplFile 77, 519, 591
\int_const:Nn 87, 88, 89, 91, 92		\ProvidesExplPackage 11, 34, 65
\int_eval:n 529		R
\int_new:N 97, 99, 101, 103, 104, 105, 106, 107, 109, 524		regex commands:
\int_set:Nn 151, 152, 153, 154, 155, 351, 354, 355, 356, 357, 358, 359, 525, 528, 529, 574		\regex_replace_all:nnN 583
\int_step_inline:nnnn 526		\regex_replace_once:nnN 317, 669
\int_use:N 694, 695		\relax 669
\c_one_int 621		\RequirePackage .. 2, 3, 4, 5, 6, 7, 8, 9, 64
\l_tmpa_int ... 351, 352, 528, 529, 530		\RequirePackageWithOptions 15, 20
\l_tmpb_int 529, 531		\resizebox 44, 655
iow commands:		S
\iow_now:Nn 629, 639		seq commands:
	K	\seq_pop_left:NN 540, 541
\KashidaHMFixOff 481, 488, 492		\seq_set_split:Nnn 539
\KashidaHMFixOn 482, 487, 491		\l_tmpa_seq 539, 540, 541
\KashidaOff 492		skip commands:
\KashidaOn 57, 491		\skip_horizontal:N 567
keys commands:		\skip_horizontal:n 635, 676, 695
\keys_define:nn 270		\l_tmpa_skip 563, 566, 567
	L	\c_zero_skip 566
\llap 655		\space 552, 559, 561
		str commands:
		\str_if_eq_p:NN 672

<code>\l_texnegar_kashida_fontfamily_-</code>	<code>\l_texnegar_linebreakpenalty_-</code>
<code>tl</code> 138, 139, 283	<code>bool</code> 146, 360, 546
<code>\texnegar_kashida_glyph</code>	<code>\l_texnegar_low_penalty_int</code>
. 615, 737, 738, 742, 743, 104, 152, 355
745, 746, 747, 748, 749, 750, 751, 752	<code>\c_texnegar_lrm_int</code> 88, 619, 691
<code>\l_texnegar_kashida_glyph_bool</code> . .	<code>\c_texnegar luatexversionmajormin_-</code>
. 141, 320	<code>int</code> 40, 41, 84
<code>\texnegar_kashida_leaders</code>	<code>\c_texnegar luatexversionminormin_-</code>
. 660, 763, 764, 768, 769,	<code>int</code> 40, 41, 85
771, 772, 773, 774, 775, 776, 777, 778	<code>\l_texnegar_main_font_full_tl</code> . . .
<code>\l_texnegar_kashida_leaders_-</code> 113, 315, 316
<code>glyph_bool</code> 142, 326, 343	<code>\l_texnegar_main_font_name_tl</code> . . .
<code>\l_texnegar_kashida_leaders_-</code> 114, 316, 317
<code>hrule_bool</code> 143, 332	<code>\l_texnegar_max_penalty_int</code>
<code>\l_texnegar_kashida_slot_int</code> 99 107, 155, 358
<code>\l_texnegar_ksh_box</code> 46, 82, 674, 676	<code>\l_texnegar_med_penalty_int</code>
<code>\c_texnegar_ksh_int</code> 87, 655, 694, 695 105, 153, 356
<code>\c_texnegar_l_charclass</code>	<code>\l_texnegar_min_penalty_int</code>
. 594, 719, 747, 748, 103, 151, 354
749, 752, 753, 773, 774, 775, 778, 779	<code>\l_texnegar_minimal_bool</code>
<code>\l_texnegar_l_clist</code> 716, 717 131, 295, 299, 510
<code>\l_texnegar_lig_aalt_clist</code> 212, 224	<code>\l_texnegar_minimal_off_tl</code> 132, 293
<code>\l_texnegar_lig_aalt_tl</code> 200, 224, 407	<code>\l_texnegar_minimal_on_tl</code> 133, 297
<code>\l_texnegar_lig_ccmp_clist</code> 213, 225	<code>\l_texnegar_pos_tl</code>
<code>\l_texnegar_lig_ccmp_tl</code> 201, 225, 408 623, 631, 641, 647, 651
<code>\l_texnegar_lig_default_clist</code> 220	<code>\texnegar_put_line_breaks:n</code> 577, 580
<code>\l_texnegar_lig_default_tl</code>	<code>\c_texnegar_r_charclass</code>
. 208, 415, 419, 532 595, 725, 750, 752, 776, 778
<code>\l_texnegar_lig_dlig_clist</code> 214, 226	<code>\l_texnegar_r_clist</code> 722, 723
<code>\l_texnegar_lig_dlig_tl</code> 202, 226, 409	<code>\c_texnegar_skip_a_tl</code>
<code>\l_texnegar_lig_fina_clist</code> 215, 227 94, 499, 738, 743, 745, 746, 747,
<code>\l_texnegar_lig_fina_tl</code> 203, 227, 410	748, 749, 750, 751, 752, 764, 769,
<code>\l_texnegar_lig_init_clist</code> 216, 228	771, 772, 773, 774, 775, 776, 777, 778
<code>\l_texnegar_lig_init_tl</code> 204, 228, 411	<code>\c_texnegar_skip_b_tl</code> 95, 571
<code>\l_texnegar_lig_locl_clist</code> 217, 229	<code>\l_texnegar_skip_default_tl</code>
<code>\l_texnegar_lig_locl_tl</code> 205, 229, 412 119, 368, 369, 370,
<code>\l_texnegar_lig_medi_clist</code> 218, 230	371, 372, 373, 374, 375, 376, 377,
<code>\l_texnegar_lig_medi_tl</code> 206, 230, 413	378, 379, 380, 381, 382, 383, 384,
<code>\l_texnegar_lig_names_clist</code>	385, 386, 387, 388, 389, 390, 391,
. 222, 525, 530, 531	392, 393, 394, 395, 396, 397, 398,
<code>\l_texnegar_lig_names_len_int</code>	400, 496, 499, 576, 737, 742, 763, 768
. 524, 525, 526	<code>\l_texnegar_stretch_glyph_tl</code>
<code>\l_texnegar_lig_rlig_clist</code> 219, 231 157, 309, 312, 734
<code>\l_texnegar_lig_rlig_tl</code> 207, 231, 414	<code>\l_texnegar_stretch_leaders_-</code>
<code>\l_texnegar_ligature_bool</code>	<code>glyph_tl</code> 158, 322, 324, 341, 664, 757
. 145, 417, 521	<code>\l_texnegar_stretch_leaders_-</code>
<code>\l_texnegar_ligatures_clist</code>	<code>hrule_tl</code> 159, 328, 330, 758
. 523, 534, 537	<code>\l_texnegar_stretch_off_tl</code>
<code>\texnegar_line_break:</code> 549 160, 334, 336
<code>\l_texnegar_line_break_penalty_-</code>	<code>\l_texnegar_stretch_on_tl</code> 161, 339
<code>int</code> 101, 354,	<code>\c_texnegar_two_int</code> 91, 693
355, 356, 357, 358, 359, 565, 571, 574	<code>\l_texnegar_use_color_tl</code>
<code>\l_texnegar_line_break_tl</code> 125, 601, 655, 674, 692
. 111, 582, 583, 584	

