

new/usr/src/cmd/mandoc/Makefile.common

1

\*\*\*\*\*

1094 Fri Jul 25 15:00:10 2014

new/usr/src/cmd/mandoc/Makefile.common

local changes

\*\*\*\*\*

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2012 Nexenta Systems, Inc. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #
```

```
17 PROGS=      mandoc mandoc_preconv
18 mandoc_OBJS = arch.o att.o chars.o eqn.o eqn_html.o eqn_term.o \
19              html.o lib.o main.o man.o man_hash.o man_html.o \
20              man_macro.o man_term.o man_validate.o mandoc.o mdoc.o \
21              mdoc_argv.o mdoc_hash.o mdoc_html.o mdoc_macro.o \
22              mdoc_man.o mdoc_term.o mdoc_validate.o msec.o out.o \
23              read.o roff.o st.o tbl.o tbl_data.o tbl_html.o \
24              tbl_layout.o tbl_opts.o tbl_term.o term.o term_ascii.o \
25              term_ps.o tree.o vol.o
```

```
27 preconv_OBJS = preconv.o
```

```
29 CFLAGS +=    $(CC_VERBOSE)
```

```
31 CPPFLAGS +=  -DHAVE_CONFIG_H -DUSE_WCHAR \
32              -DOSNAME="\"illumos\"" \
33              -DOSNAME="\"illumos\"" \
34              -DVERSION="\"1.12.1\""
```

\*\*\*\*\*

3308 Fri Jul 25 15:00:10 2014

new/usr/src/cmd/mandoc/arch.in

Update to 1.12.3.

\*\*\*\*\*

```

1 /* $Id: arch.in,v 1.14 2013/09/16 22:12:57 schwarze Exp $ */
1 /* $Id: arch.in,v 1.12 2012/01/28 14:02:17 joerg Exp $ */
2 /*
3  * Copyright (c) 2009 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17
18 /*
19  * This file defines the architecture token of the .Dt prologue macro.
20  * All architectures that your system supports (or the manuals of your
21  * system) should be included here. The right-hand-side is the
22  * formatted output.
23  *
24  * Be sure to escape strings.
25  *
26  * REMEMBER TO ADD NEW ARCHITECTURES TO MDOC.7!
27  */
28
29 LINE("acorn26",      "Acorn26")
30 LINE("acorn32",      "Acorn32")
31 LINE("algor",        "Algor")
32 LINE("alpha",        "Alpha")
33 LINE("amd64",        "AMD64")
34 LINE("amiga",        "Amiga")
35 LINE("amigappc",     "AmigaPPC")
36 LINE("arc",          "ARC")
37 LINE("arm",          "ARM")
38 LINE("arm26",        "ARM26")
39 LINE("arm32",        "ARM32")
40 LINE("armish",       "ARMISH")
41 LINE("armv7",        "ARMv7")
42 LINE("aviion",       "AViion")
43 LINE("atari",        "ATARI")
44 LINE("beagle",       "Beagle")
45 LINE("bebox",        "BeBox")
46 LINE("cats",         "cats")
47 LINE("cesfic",       "CESFIC")
48 LINE("cobalt",       "Cobalt")
49 LINE("dreamcast",   "Dreamcast")
50 LINE("emips",        "EMIPS")
51 LINE("evbarm",       "evbARM")
52 LINE("evbmips",     "evbMIPS")
53 LINE("evbppc",       "evbPPC")
54 LINE("evbsh3",      "evbSH3")
55 LINE("ews4800mips", "EWS4800MIPS")
56 LINE("hp300",        "HP300")
57 LINE("hp700",        "HP700")
58 LINE("hpcarm",       "HPCARM")
59 LINE("hpcmips",     "HPCMIPS")
60 LINE("hpcsh",       "HPCSH")

```

```

60 LINE("hppa",        "HPPA")
61 LINE("hppa64",      "HPPA64")
62 LINE("ia64",        "ia64")
63 LINE("i386",        "i386")
64 LINE("ibmnws",      "IBMNWS")
65 LINE("iyonix",      "Iyonix")
66 LINE("landisk",     "LANDISK")
67 LINE("loongson",    "Loongson")
68 LINE("luna68k",     "Luna68k")
69 LINE("luna88k",     "Luna88k")
70 LINE("m68k",        "m68k")
71 LINE("mac68k",      "Mac68k")
72 LINE("macppc",      "MacPPC")
73 LINE("mips",        "MIPS")
74 LINE("mips64",      "MIPS64")
75 LINE("mipsco",     "MIPSCO")
76 LINE("mmeye",       "mmEye")
77 LINE("mvme68k",    "MVME68k")
78 LINE("mvme88k",    "MVME88k")
79 LINE("mvmeppc",    "MVMEPPC")
80 LINE("netwinder",   "NetWinder")
81 LINE("news68k",     "NEWS68k")
82 LINE("newsmps",    "NeWSMIPS")
83 LINE("next68k",    "NeXT68k")
84 LINE("octeon",      "OCTEON")
85 LINE("ofppc",       "OFPPC")
86 LINE("palm",        "Palm")
87 LINE("pc532",       "PC532")
88 LINE("playstation2", "PlayStation2")
89 LINE("pmax",        "PMAX")
90 LINE("pmppc",       "pmPPC")
91 LINE("powerpc",     "PowerPC")
92 LINE("prep",        "PREP")
93 LINE("rs6000",      "RS6000")
94 LINE("sandpoint",  "Sandpoint")
95 LINE("sbmips",      "SBMIPS")
96 LINE("sgi",         "SGI")
97 LINE("sgimips",    "SGIMIPS")
98 LINE("sh3",         "SH3")
99 LINE("shark",       "Shark")
100 LINE("socppc",     "SOCPCC")
101 LINE("solbourne",  "Solbourne")
102 LINE("sparc",       "SPARC")
103 LINE("sparc64",    "SPARC64")
104 LINE("sun2",        "Sun2")
105 LINE("sun3",        "Sun3")
106 LINE("tahoe",      "Tahoe")
107 LINE("vax",         "VAX")
108 LINE("x68k",        "X68k")
109 LINE("x86",         "x86")
110 LINE("x86_64",     "x86_64")
111 LINE("xen",         "Xen")
112 LINE("zaurus",     "Zaurus")

```

```

*****
3565 Fri Jul 25 15:00:10 2014
new/usr/src/cmd/mandoc/chars.c
Update to 1.12.3.
*****
1 /* $Id: chars.c,v 1.54 2013/06/20 22:39:30 schwarze Exp $ */
1 /* $Id: chars.c,v 1.52 2011/11/08 00:15:23 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdlib.h>
25 #include <string.h>
26
27 #include "mandoc.h"
28 #include "libmandoc.h"
29
30 #define PRINT_HI      126
31 #define PRINT_LO      32
32
33 struct ln {
34     struct ln      *next;
35     const char     *code;
36     const char     *ascii;
37     int            unicode;
38 };
39
40 #define LINES_MAX      329
40 #define LINES_MAX      328
41
42 #define CHAR(in, ch, code) \
43     { NULL, (in), (ch), (code) },
44
45 #define CHAR_TBL_START static struct ln lines[LINES_MAX] = {
46 #define CHAR_TBL_END   };
47
48 #include "chars.in"
49
50 struct mchars {
51     struct ln      **htab;
52 };
53
54 unchanged_portion_omitted
55
56 struct mchars *
57 mchars_alloc(void)
58 {
59     struct mchars *tab;
60     struct ln      **htab;

```

```

70     struct ln      *pp;
71     int            i, hash;
72
73     /*
74     * Constructs a very basic chaining hashtable. The hash routine
75     * is simply the integral value of the first character.
76     * Subsequent entries are chained in the order they're processed.
77     */
78
79     tab = mandoc_malloc(sizeof(struct mchars));
80     htab = mandoc_calloc(PRINT_HI - PRINT_LO + 1, sizeof(struct ln *));
80     htab = mandoc_calloc(PRINT_HI - PRINT_LO + 1, sizeof(struct ln **));
81
82     for (i = 0; i < LINES_MAX; i++) {
83         hash = (int)lines[i].code[0] - PRINT_LO;
84
85         if (NULL == (pp = htab[hash])) {
86             htab[hash] = &lines[i];
87             continue;
88         }
89
90         for ( ; pp->next; pp = pp->next)
91             /* Scan ahead. */ ;
92         pp->next = &lines[i];
93     }
94
95     tab->htab = htab;
96     return(tab);
97 }
98
99 unchanged_portion_omitted

```

```

*****
10052 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/chars.in
Update to 1.12.3.
*****
1 /* $Id: chars.in,v 1.43 2013/06/20 22:39:30 schwarze Exp $ */
1 /* $Id: chars.in,v 1.42 2011/10/02 10:02:26 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */

18 /*
19  * The ASCII translation tables.
20 *
21 * The left-hand side corresponds to the input sequence (\x, \(\xx, \*(xx
22 * and so on) whose length is listed second element. The right-hand
23 * side is what's produced by the front-end, with the fourth element
24 * being its length.
25 *
26 * XXX - C-escape strings!
27 * XXX - update LINES_MAX if adding more!
28 */

30 /* Non-breaking, non-collapsing space uses unit separator. */
31 static const char ascii_nbrsp[2] = { ASCII_NBRSP, '\0' };

33 CHAR_TBL_START

35 /* Spacing. */
36 CHAR("c", "", 0)
37 CHAR("0", " ", 8194)
38 CHAR(" ", " ", ascii_nbrsp, 160)
39 CHAR("~", " ", ascii_nbrsp, 160)
40 CHAR("%", " ", 0)
41 CHAR("&", " ", 0)
42 CHAR("^", " ", 0)
43 CHAR("|", " ", 0)
44 CHAR("}", " ", 0)
45 CHAR("t", " ", 0)

47 /* Accents. */
48 CHAR("a\"", "\"", 779)
49 CHAR("a-", "-", 175)
50 CHAR("a.", ".", 729)
51 CHAR("a^", "^", 770)
52 CHAR("a'", "'", 769)
53 CHAR("aa", "aa", 769)
54 CHAR("ga", "ga", 768)
55 CHAR(" ", " ", 768)
56 CHAR("ab", "ab", 774)
57 CHAR("ac", "ac", 807)
58 CHAR("ad", "ad", 776)
59 CHAR("ah", "ah", 711)
60 CHAR("ao", "o", 730)

```

```

61 CHAR("a~", "~", 771)
62 CHAR("ho", "o", 808)
63 CHAR("ha", "a", 94)
64 CHAR("ti", "i", 126)

66 /* Quotes. */
67 CHAR("Bq", "q", 8222)
68 CHAR("bq", "q", 8218)
69 CHAR("lq", "q", 8220)
70 CHAR("rq", "q", 8221)
71 CHAR("oq", "q", 8216)
72 CHAR("cq", "q", 8217)
73 CHAR("aq", "q", 39)
74 CHAR("dq", "q", 34)
75 CHAR("Fo", "<<", 171)
76 CHAR("Fc", ">>", 187)
77 CHAR("fo", "<", 8249)
78 CHAR("fc", ">", 8250)

80 /* Brackets. */
81 CHAR("lB", "[", 91)
82 CHAR("rB", "]", 93)
83 CHAR("lC", "{", 123)
84 CHAR("rC", "}", 125)
85 CHAR("la", "<", 60)
86 CHAR("ra", ">", 62)
87 CHAR("bv", "v", 9130)
88 CHAR("braceex", " ", 9130)
89 CHAR("bracketlefttp", " ", 9121)
90 CHAR("bracketleftbp", " ", 9123)
91 CHAR("bracketleftex", " ", 9122)
92 CHAR("bracketrighttp", " ", 9124)
93 CHAR("bracketrightbp", " ", 9126)
94 CHAR("bracketrightex", " ", 9125)
95 CHAR("lt", "l-", 9127)
96 CHAR("bracelefttp", "l-", 9127)
97 CHAR("lk", "l", 9128)
98 CHAR("braceleftmid", "l", 9128)
99 CHAR("lb", "l-", 9129)
100 CHAR("braceleftbp", "l-", 9129)
101 CHAR("braceleftex", "l", 9130)
102 CHAR("rt", "r-", 9131)
103 CHAR("bracerighttp", "r-", 9131)
104 CHAR("rk", "r", 9132)
105 CHAR("bracerightmid", "r", 9132)
106 CHAR("rb", "r-", 9133)
107 CHAR("bracerightbp", "r-", 9133)
108 CHAR("bracerightex", "r", 9130)
109 CHAR("parenlefttp", "(", 9115)
110 CHAR("parenleftbp", "(", 9117)
111 CHAR("parenleftex", "(", 9116)
112 CHAR("parenrighttp", ")", 9118)
113 CHAR("parenrightbp", ")", 9120)
114 CHAR("parenrightex", ")", 9119)

116 /* Greek characters. */
117 CHAR("A", "A", 913)
118 CHAR("B", "B", 914)
119 CHAR("G", "G", 915)
120 CHAR("D", "D", 916)
121 CHAR("E", "E", 917)
122 CHAR("Z", "Z", 918)
123 CHAR("Y", "H", 919)
124 CHAR("H", "O", 920)
125 CHAR("I", "I", 921)
126 CHAR("K", "K", 922)

```

```

127 CHAR(" *L", "/\\", 923)
128 CHAR(" *M", "M", 924)
129 CHAR(" *N", "N", 925)
130 CHAR(" *C", "H", 926)
131 CHAR(" *O", "O", 927)
132 CHAR(" *P", "TT", 928)
133 CHAR(" *R", "P", 929)
134 CHAR(" *S", ">", 931)
135 CHAR(" *T", "T", 932)
136 CHAR(" *U", "Y", 933)
137 CHAR(" *F", "O", 934)
138 CHAR(" *X", "X", 935)
139 CHAR(" *Q", "Y", 936)
140 CHAR(" *W", "O", 937)
141 CHAR(" *a", "a", 945)
142 CHAR(" *b", "B", 946)
143 CHAR(" *g", "y", 947)
144 CHAR(" *d", "d", 948)
145 CHAR(" *e", "e", 949)
146 CHAR(" *z", "C", 950)
147 CHAR(" *y", "n", 951)
148 CHAR(" *h", "O", 952)
149 CHAR(" *i", "i", 953)
150 CHAR(" *k", "k", 954)
151 CHAR(" *l", "\\ ", 955)
152 CHAR(" *m", "u", 956)
153 CHAR(" *n", "v", 957)
154 CHAR(" *c", "E", 958)
155 CHAR(" *o", "o", 959)
156 CHAR(" *p", "n", 960)
157 CHAR(" *r", "p", 961)
158 CHAR(" *s", "o", 963)
159 CHAR(" *t", "t", 964)
160 CHAR(" *u", "u", 965)
161 CHAR(" *f", "o", 981)
162 CHAR(" *x", "x", 967)
163 CHAR(" *q", "u", 968)
164 CHAR(" *w", "w", 969)
165 CHAR(" *h", "O", 977)
166 CHAR(" *f", "o", 966)
167 CHAR(" *p", "w", 982)
168 CHAR(" *e", "e", 1013)
169 CHAR(" *s", "s", 962)

171 /* Accented letters. */
172 CHAR(" ,C", "C", 199)
173 CHAR(" ,c", "c", 231)
174 CHAR(" /L", "L", 321)
175 CHAR(" /O", "O", 216)
176 CHAR(" /l", "l", 322)
177 CHAR(" /o", "o", 248)
178 CHAR(" oA", "A", 197)
179 CHAR(" oa", "a", 229)
180 CHAR(" A", "A", 196)
181 CHAR(" :E", "E", 203)
182 CHAR(" :I", "I", 207)
183 CHAR(" :O", "O", 214)
184 CHAR(" :U", "U", 220)
185 CHAR(" :a", "a", 228)
186 CHAR(" :e", "e", 235)
187 CHAR(" :i", "i", 239)
188 CHAR(" :o", "o", 246)
189 CHAR(" :u", "u", 252)
190 CHAR(" :y", "y", 255)
191 CHAR(" \A", "A", 193)
192 CHAR(" \E", "E", 201)

```

```

193 CHAR(" \I", "I", 205)
194 CHAR(" \O", "O", 211)
195 CHAR(" \U", "U", 218)
196 CHAR(" \a", "a", 225)
197 CHAR(" \e", "e", 233)
198 CHAR(" \i", "i", 237)
199 CHAR(" \o", "o", 243)
200 CHAR(" \u", "u", 250)
201 CHAR(" ^A", "A", 194)
202 CHAR(" ^E", "E", 202)
203 CHAR(" ^I", "I", 206)
204 CHAR(" ^O", "O", 212)
205 CHAR(" ^U", "U", 219)
206 CHAR(" ^a", "a", 226)
207 CHAR(" ^e", "e", 234)
208 CHAR(" ^i", "i", 238)
209 CHAR(" o", "o", 244)
210 CHAR(" ^u", "u", 251)
211 CHAR(" A", "A", 192)
212 CHAR(" E", "E", 200)
213 CHAR(" I", "I", 204)
214 CHAR(" O", "O", 210)
215 CHAR(" U", "U", 217)
216 CHAR(" a", "a", 224)
217 CHAR(" e", "e", 232)
218 CHAR(" i", "i", 236)
219 CHAR(" o", "o", 242)
220 CHAR(" u", "u", 249)
221 CHAR(" ~A", "A", 195)
222 CHAR(" ~N", "N", 209)
223 CHAR(" ~O", "O", 213)
224 CHAR(" ~a", "a", 227)
225 CHAR(" ~n", "n", 241)
226 CHAR(" ~o", "o", 245)

228 /* Arrows and lines. */
229 CHAR(" <-", "<-", 8592)
230 CHAR(" >-", ">-", 8594)
231 CHAR(" <>", "<>", 8596)
232 CHAR(" da", "v", 8595)
233 CHAR(" ua", "v", 8593)
234 CHAR(" va", "v", 8597)
235 CHAR(" lA", "<=", 8656)
236 CHAR(" rA", ">=", 8658)
237 CHAR(" hA", "<=>", 8660)
238 CHAR(" dA", "v", 8659)
239 CHAR(" uA", "v", 8657)
240 CHAR(" vA", "v", 8661)

242 /* Logic. */
243 CHAR(" AN", "A", 8743)
244 CHAR(" OR", "v", 8744)
245 CHAR(" no", "~", 172)
246 CHAR(" tno", "~", 172)
247 CHAR(" te", "3", 8707)
248 CHAR(" fa", "V", 8704)
249 CHAR(" st", "-", 8715)
250 CHAR(" tf", ".:", 8756)
251 CHAR(" 3d", ".:", 8756)
252 CHAR(" or", "|", 124)

254 /* Mathematics. */
255 CHAR(" pl", "+", 43)
256 CHAR(" mi", "-", 8722)
257 CHAR(" =", "=", 45)
258 CHAR(" -+", "-+", 8723)

```

```

259 CHAR("+-", "+-", 177)
260 CHAR("+-", "+-", 177)
261 CHAR("pc", " ", 183)
262 CHAR("md", " ", 8901)
263 CHAR("mu", "x", 215)
264 CHAR("tmu", "x", 215)
265 CHAR("c*", "x", 8855)
266 CHAR("c+", "+", 8853)
267 CHAR("di", "-:-", 247)
268 CHAR("tdi", "-:-", 247)
269 CHAR("f/", "/", 8260)
270 CHAR("**", "**", 8727)
271 CHAR("<=", "<=", 8804)
272 CHAR(">=", ">=", 8805)
273 CHAR("<<", "<<", 8810)
274 CHAR(">>", ">>", 8811)
275 CHAR("eq", "=", 61)
276 CHAR("!=", "!=", 8800)
277 CHAR("==", "==", 8801)
278 CHAR("ne", "!=", 8802)
279 CHAR("=~", "=~", 8773)
280 CHAR("~~", "~~", 8771)
281 CHAR("ap", "~", 8764)
282 CHAR("~~", "~~", 8776)
283 CHAR("==", "==", 8780)
284 CHAR("pt", "oc", 8733)
285 CHAR("es", "{", 8709)
286 CHAR("mo", "E", 8712)
287 CHAR("nm", "IE", 8713)
288 CHAR("sb", "(=", 8834)
289 CHAR("nb", "(!=", 8836)
290 CHAR("sp", "=", 8835)
291 CHAR("nc", "!=)", 8837)
292 CHAR("ib", "(=", 8838)
293 CHAR("ip", "(=", 8839)
294 CHAR("ca", "(^", 8745)
295 CHAR("cu", "U", 8746)
296 CHAR("/_", "_", 8736)
297 CHAR("pp", "_|_", 8869)
298 CHAR("is", "I", 8747)
299 CHAR("integral", "I", 8747)
300 CHAR("sum", "E", 8721)
301 CHAR("product", "TT", 8719)
302 CHAR("coproduct", "U", 8720)
303 CHAR("gr", "v", 8711)
304 CHAR("sr", "\\", 8730)
305 CHAR("sqrt", "\\", 8730)
306 CHAR("lc", "|~", 8968)
307 CHAR("rc", "~|", 8969)
308 CHAR("lf", "|_", 8970)
309 CHAR("xf", "_|", 8971)
310 CHAR("if", "oo", 8734)
311 CHAR("Ah", "N", 8501)
312 CHAR("Im", "I", 8465)
313 CHAR("Re", "R", 8476)
314 CHAR("pd", "a", 8706)
315 CHAR("-h", "/h", 8463)
316 CHAR("12", "1/2", 189)
317 CHAR("14", "1/4", 188)
318 CHAR("34", "3/4", 190)

320 /* Ligatures. */
321 CHAR("ff", "ff", 64256)
322 CHAR("fi", "fi", 64257)
323 CHAR("fl", "fl", 64258)
324 CHAR("Fi", "ffi", 64259)

```

```

325 CHAR("Fl", "ffl", 64260)
326 CHAR("AE", "AE", 198)
327 CHAR("ae", "ae", 230)
328 CHAR("OE", "OE", 338)
329 CHAR("oe", "oe", 339)
330 CHAR("ss", "ss", 223)
331 CHAR("IJ", "IJ", 306)
332 CHAR("ij", "ij", 307)

334 /* Special letters. */
335 CHAR("-D", "D", 208)
336 CHAR("Sd", "o", 240)
337 CHAR("TP", "b", 222)
338 CHAR("Tp", "b", 254)
339 CHAR(".i", "i", 305)
340 CHAR(".j", "j", 567)

342 /* Currency. */
343 CHAR("Do", "$", 36)
344 CHAR("ct", "c", 162)
345 CHAR("Eu", "EUR", 8364)
346 CHAR("eu", "EUR", 8364)
347 CHAR("Ye", "Y", 165)
348 CHAR("Po", "L", 163)
349 CHAR("Cs", "x", 164)
350 CHAR("Fn", "f", 402)

352 /* Lines. */
353 CHAR("ba", "|", 124)
354 CHAR("br", "|", 9474)
355 CHAR("ul", " ", 95)
356 CHAR("rl", "-", 8254)
357 CHAR("bb", "|", 166)
358 CHAR("sl", "/", 47)
359 CHAR("rs", "\\ ", 92)

361 /* Text markers. */
362 CHAR("ci", "o", 9675)
363 CHAR("bu", "o", 8226)
364 CHAR("dd", "=", 8225)
365 CHAR("dg", "-", 8224)
366 CHAR("lz", "<>", 9674)
367 CHAR("sq", "[ ]", 9633)
368 CHAR("ps", "9|", 182)
369 CHAR("sc", "S", 167)
370 CHAR("lh", "<=", 9756)
371 CHAR("rh", "=>", 9758)
372 CHAR("at", "@", 64)
373 CHAR("sh", "#", 35)
374 CHAR("CR", "_|", 8629)
375 CHAR("OK", "\\", 10003)

377 /* Legal symbols. */
378 CHAR("co", "(C)", 169)
379 CHAR("rg", "(R)", 174)
380 CHAR("tm", "tm", 8482)

382 /* Punctuation. */
383 CHAR(".", ".", 46)
384 CHAR("r!", "i", 161)
385 CHAR("r?", "c", 191)
386 CHAR("em", "--", 8212)
387 CHAR("en", "-", 8211)
388 CHAR("hy", "-", 8208)
389 CHAR("e", "\\ ", 92)

```

```
391 /* Units. */
392 CHAR("de", "o", 176)
393 CHAR("%0", "%o", 8240)
394 CHAR("fm", "\'", 8242)
395 CHAR("sd", "\"", 8243)
396 CHAR("mc", "mu", 181)
398 CHAR_TBL_END
```

new/usr/src/cmd/mandoc/config.h

1

\*\*\*\*\*

1138 Fri Jul 25 15:00:11 2014

new/usr/src/cmd/mandoc/config.h

Update to 1.12.3.

\*\*\*\*\*

```
1 #ifndef MANDOC_CONFIG_H
2 #define MANDOC_CONFIG_H
```

```
4 #if defined(__linux__) || defined(__MINT__)
5 # define _GNU_SOURCE /* strptime(), getsubopt() */
6 #endif
```

```
8 #include <stdio.h>
```

```
10 #define VERSION "1.12.3"
10 #define HAVE_STRPTIME
11 #define HAVE_GETSUBOPT
12 #define HAVE_STRLCAT
13 #define HAVE_STRLCPY
```

```
12 #include <sys/types.h>
```

```
14 #if !defined(__BEGIN_DECLS)
15 # ifdef __cplusplus
16 # define __BEGIN_DECLS extern "C" {
17 # else
18 # define __BEGIN_DECLS
19 # endif
20 #endif
21 #if !defined(__END_DECLS)
22 # ifdef __cplusplus
23 # define __END_DECLS }
24 # else
25 # define __END_DECLS
26 # endif
27 #endif
```

```
29 #ifndef HAVE_BETOH64
30 # if defined(__APPLE__)
31 # define betoh64(x) OSSwapBigToHostInt64(x)
32 #if defined(__APPLE__)
33 # define htobe32(x) OSSwapHostToBigInt32(x)
34 # define betoh32(x) OSSwapBigToHostInt32(x)
32 # define htobe64(x) OSSwapHostToBigInt64(x)
33 # elif defined(__sun)
34 # define betoh64(x) BE_64(x)
35 # define htobe64(x) BE_64(x)
36 # else
36 # define betoh64(x) OSSwapBigToHostInt64(x)
37 #elif defined(__linux__)
38 # define betoh32(x) be32toh(x)
37 # define betoh64(x) be64toh(x)
38 # endif
39 #endif
```

```
41 #ifndef HAVE_STRLCAT
42 extern size_t strlcat(char *, const char *, size_t);
43 #endif
44 #ifndef HAVE_STRLCPY
45 extern size_t strlcpy(char *, const char *, size_t);
46 #endif
47 #ifndef HAVE_GETSUBOPT
48 extern int getsubopt(char **, char * const *, char **);
49 extern char *suboptarg;
50 #endif
51 #ifndef HAVE_FGETLN
```

new/usr/src/cmd/mandoc/config.h

2

```
52 extern char *fgetln(FILE *, size_t *);
53 #endif
```

```
55 #endif /* MANDOC_CONFIG_H */
```



```

*****
15031 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/html.c
Update to 1.12.3.
*****
1 /* $Id: html.c,v 1.152 2013/08/08 20:07:47 schwarze Exp $ */
1 /* $Id: html.c,v 1.150 2011/10/05 21:35:17 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdarg.h>
27 #include <stdio.h>
28 #include <stdint.h>
29 #include <stdlib.h>
30 #include <string.h>
31 #include <unistd.h>
32
33 #include "mandoc.h"
34 #include "libmandoc.h"
35 #include "out.h"
36 #include "html.h"
37 #include "main.h"
38
39 struct htmldata {
40     const char    *name;
41     int           flags;
42 #define HTML_CLRLINE    (1 << 0)
43 #define HTML_NOSTACK    (1 << 1)
44 #define HTML_AUTOCLOSE  (1 << 2) /* Tag has auto-closure. */
45 };
46
47 unchanged_portion_omitted
48
223 static void
224 print_metaf(struct html *h, enum mandoc_esc deco)
225 {
226     enum htmlfont    font;
227
228     switch (deco) {
229     case (ESCAPE_FONTPREV):
230         font = h->metal;
231         break;
232     case (ESCAPE_FONTITALIC):
233         font = HTMLFONT_ITALIC;
234         break;

```

```

235     case (ESCAPE_FONTBOLD):
236         font = HTMLFONT_BOLD;
237         break;
238     case (ESCAPE_FONTBI):
239         font = HTMLFONT_BI;
240         break;
241     case (ESCAPE_FONT):
242         /* FALLTHROUGH */
243     case (ESCAPE_FONTRoman):
244         font = HTMLFONT_NONE;
245         break;
246     default:
247         abort();
248         /* NOTREACHED */
249     }
250
251     if (h->metaf) {
252         print_tagq(h, h->metaf);
253         h->metaf = NULL;
254     }
255
256     h->metal = h->metac;
257     h->metac = font;
258
259     switch (font) {
260     case (HTMLFONT_ITALIC):
261         h->metaf = print_otag(h, TAG_I, 0, NULL);
262         break;
263     case (HTMLFONT_BOLD):
264         h->metaf = print_otag(h, TAG_B, 0, NULL);
265         break;
266     case (HTMLFONT_BI):
267         h->metaf = print_otag(h, TAG_B, 0, NULL);
268     if (HTMLFONT_NONE != font)
269         h->metaf = HTMLFONT_BOLD == font ?
270             print_otag(h, TAG_B, 0, NULL) :
271             print_otag(h, TAG_I, 0, NULL);
272         break;
273     default:
274         break;
275     }
276
277     int
278     html_strlen(const char *cp)
279     {
280         size_t    rsz;
281         int        skip, sz;
282         int        ssz, sz;
283         const char *seq, *p;
284
285         /*
286          * Account for escaped sequences within string length
287          * calculations. This follows the logic in term_strlen() as we
288          * must calculate the width of produced strings.
289          * Assume that characters are always width of "1". This is
290          * hacky, but it gets the job done for approximation of widths.
291          */
292
293         sz = 0;
294         skip = 0;
295         while (1) {
296             rsz = strcspn(cp, "\\");
297             if (rsz) {
298                 cp += rsz;
299                 if (skip) {

```

```

296         skip = 0;
297         rsz--;
298     }
299     sz += rsz;
300 }
301 if ('\0' == *cp)
302     break;
303 cp++;
304 switch (mandoc_escape(&cp, NULL, NULL)) {
277 while (NULL != (p = strchr(cp, '\\'))) {
278     sz += (int)(p - cp);
279     ++cp;
280     switch (mandoc_escape(&cp, &seq, &ssz)) {
305     case (ESCAPE_ERROR):
306         return(sz);
307     case (ESCAPE_UNICODE):
308         /* FALLTHROUGH */
309     case (ESCAPE_NUMBERED):
310         /* FALLTHROUGH */
311     case (ESCAPE_SPECIAL):
312         if (skip)
313             skip = 0;
314         else
315             sz++;
316         break;
317     case (ESCAPE_SKIPCHAR):
318         skip = 1;
319         break;
320     default:
321         break;
322     }
323 }
324 return(sz);
295 assert(sz >= 0);
296 return(sz + strlen(cp));
325 }

327 static int
328 print_encode(struct html *h, const char *p, int norecuse)
329 {
330     size_t      sz;
331     int         c, len, nospace;
332     const char *seq;
333     enum mandoc_esc esc;
334     static const char rejs[6] = { '\\', '<', '>', '&', ASCII_HYPH, '\0' };

336     nospace = 0;

338     while ('\0' != *p) {
339         if (HTML_SKIPCHAR & h->flags && '\\\0' != *p) {
340             h->flags &= ~HTML_SKIPCHAR;
341             p++;
342             continue;
343         }

345         sz = strcspn(p, rejs);

347         fwrite(p, 1, sz, stdout);
348         p += (int)sz;

350         if ('\0' == *p)
351             break;

353         switch (*p++) {
354     case ('<'):

```

```

355         printf("&lt;");
356         continue;
357     case ('>'):
358         printf("&gt;");
359         continue;
360     case ('&'):
361         printf("&amp;");
362         continue;
363     case (ASCII_HYPH):
364         putchar('-');
365         continue;
366     default:
367         break;
368 }

370     esc = mandoc_escape(&p, &seq, &len);
371     if (ESCAPE_ERROR == esc)
372         break;

374     switch (esc) {
375     case (ESCAPE_FONT):
376         /* FALLTHROUGH */
377     case (ESCAPE_FONTPREV):
378         /* FALLTHROUGH */
379     case (ESCAPE_FONTBOLD):
380         /* FALLTHROUGH */
381     case (ESCAPE_FONTTITALIC):
382         /* FALLTHROUGH */
383     case (ESCAPE_FONTBI):
384         /* FALLTHROUGH */
385     case (ESCAPE_FONTRoman):
386         if (0 == norecuse)
387             print_metaf(h, esc);
388         continue;
389     case (ESCAPE_SKIPCHAR):
390         h->flags |= HTML_SKIPCHAR;
391         continue;
392     default:
393         break;
394 }

396     if (h->flags & HTML_SKIPCHAR) {
397         h->flags &= ~HTML_SKIPCHAR;
398         continue;
399     }

401     switch (esc) {
402     case (ESCAPE_UNICODE):
403         /* Skip passed "u" header. */
404         c = mchars_num2uc(seq + 1, len - 1);
405         if ('\0' != c)
406             printf("&#x%x;", c);
407         break;
408     case (ESCAPE_NUMBERED):
409         c = mchars_num2char(seq, len);
410         if ('\0' != c)
411             putchar(c);
412         break;
413     case (ESCAPE_SPECIAL):
414         c = mchars_spec2cp(h->symtab, seq, len);
415         if (c > 0)
416             printf("&#%d;", c);
417         else if (-1 == c && 1 == len)
418             putchar((int)*seq);
419         break;
420     case (ESCAPE_FONT):

```

```

360         /* FALLTHROUGH */
361         case (ESCAPE_FONTPREV):
362         /* FALLTHROUGH */
363         case (ESCAPE_FONTBOLD):
364         /* FALLTHROUGH */
365         case (ESCAPE_FONTITALIC):
366         /* FALLTHROUGH */
367         case (ESCAPE_FONTRoman):
368             if (norecurse)
369                 break;
370             print_metaf(h, esc);
371             break;
420         case (ESCAPE_NOSPACE):
421             if ('\0' == *p)
422                 nospace = 1;
423             break;
424         default:
425             break;
426     }
427 }
429     return(nospace);
430 }

```

unchanged\_portion\_omitted

```

547 void
548 print_text(struct html *h, const char *word)
549 {

```

```

551     if (! (HTML_NOSPACE & h->flags)) {
552         /* Manage keeps! */
553         if (! (HTML_KEEP & h->flags)) {
554             if (HTML_PREKEEP & h->flags)
555                 h->flags |= HTML_KEEP;
556             putchar(' ');
557         } else
558             printf("&#160;");
559     }

```

```

561     assert(NULL == h->metaf);
562     switch (h->metac) {
563     case (HTMLFONT_ITALIC):
564         h->metaf = print_otag(h, TAG_I, 0, NULL);
565         break;
566     case (HTMLFONT_BOLD):
567         h->metaf = print_otag(h, TAG_B, 0, NULL);
568         break;
569     case (HTMLFONT_BI):
570         h->metaf = print_otag(h, TAG_B, 0, NULL);
571     if (HTMLFONT_NONE != h->metac)
572         h->metaf = HTMLFONT_BOLD == h->metac ?
573             print_otag(h, TAG_B, 0, NULL) :
574             print_otag(h, TAG_I, 0, NULL);
575         break;
576     default:
577         break;
578     }

```

```

577     assert(word);
578     if (! print_encode(h, word, 0)) {
579         if (! (h->flags & HTML_NONOSPACE))
580             h->flags &= ~HTML_NOSPACE;
581     } else
582         h->flags |= HTML_NOSPACE;

```

```

584     if (h->metaf) {

```

```

585         print_tagq(h, h->metaf);
586         h->metaf = NULL;
587     }
589     h->flags &= ~HTML_IGNDELIM;
590 }

```

unchanged\_portion\_omitted

\*\*\*\*\*

4241 Fri Jul 25 15:00:11 2014

new/usr/src/cmd/mandoc/html.h

Update to 1.12.3.

\*\*\*\*\*

```

1 /* $Id: html.h,v 1.49 2013/08/08 20:07:47 schwarze Exp $ */
1 /* $Id: html.h,v 1.47 2011/10/05 21:35:17 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifndef HTML_H
18 #define HTML_H
19
20 __BEGIN_DECLS
21
22 enum      htmltag {
23     TAG_HTML,
24     TAG_HEAD,
25     TAG_BODY,
26     TAG_META,
27     TAG_TITLE,
28     TAG_DIV,
29     TAG_H1,
30     TAG_H2,
31     TAG_SPAN,
32     TAG_LINK,
33     TAG_BR,
34     TAG_A,
35     TAG_TABLE,
36     TAG_TBODY,
37     TAG_COL,
38     TAG_TR,
39     TAG_TD,
40     TAG_LI,
41     TAG_UL,
42     TAG_OL,
43     TAG_DL,
44     TAG_DT,
45     TAG_DD,
46     TAG_BLOCKQUOTE,
47     TAG_P,
48     TAG_PRE,
49     TAG_B,
50     TAG_I,
51     TAG_CODE,
52     TAG_SMALL,
53     TAG_MAX
54 };
55
56 unchanged_portion_omitted
57
58 enum      htmlfont {
59     HTMLFONT_NONE = 0,
60     HTMLFONT_BOLD,
61     HTMLFONT_ITALIC,
62

```

```

78     HTMLFONT_BI,
79     HTMLFONT_MAX
80 };
81
82 unchanged_portion_omitted
83
84 struct    html {
85     int      flags;
86
87     #define HTML_NOSPACE      (1 << 0) /* suppress next space */
88     #define HTML_IGNDELIM    (1 << 1)
89     #define HTML_KEEP        (1 << 2)
90     #define HTML_PREKEEP     (1 << 3)
91     #define HTML_NONOSPACE   (1 << 4) /* never add spaces */
92     #define HTML_LITERAL     (1 << 5) /* literal (e.g., <PRE>) context */
93     #define HTML_SKIPCHAR    (1 << 6) /* skip the next character */
94
95     struct tagq      tags; /* stack of open tags */
96     struct rofftbl   tbl; /* current table */
97     struct tag       *tblt; /* current open table scope */
98     struct mchars    *symtab; /* character-escapes */
99     char             *base_man; /* base for manpage href */
100    char             *base_includes; /* base for include href */
101    char             *style; /* style-sheet URI */
102    char             buf[BUFSIZ]; /* see bufcat and friends */
103    size_t           buflen;
104
105     struct tag       *metaf; /* current open font scope */
106     enum htmlfont    metal; /* last used font */
107     enum htmlfont    metac; /* current font mode */
108     enum htmltype    type; /* output media type */
109     int              oflags; /* output options */
110
111     #define HTML_FRAGMENT    (1 << 0) /* don't emit HTML/HEAD/BODY */
112 };
113
114 unchanged_portion_omitted

```

```

*****
4877 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/lib.in
local changes
Update to 1.12.3.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
14  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
15 */

16 /*
17  * Note that we don't document "legacy" libraries that have moved into
18  * libc. While there will be section 3lib man pages for them, they
19  * won't be referenced in other man pages.
20  * TBD
21 */
22 LINE("libadm", "General Administrative Library (libadm, \\\-ladm)")
23 LINE("libbsdmalloc", "BSD Memory Allocation Library (libbsdmalloc, \\\-bsdmallo")
24 LINE("libbsm", "Security and Auditing Library (libbsm, \\\-bsm)")
25 LINE("libc", "Standard C Library (libc, \\\-lc)")
26 LINE("libc_db", "Threads Debugging Library (libc_db, \\\-lc_db)")
27 LINE("libc_fgsadm", "Configuration Administration Library (libc_fgsadm, \\\-lcfg")
28 LINE("libcommutil", "Communication Protocol Parser Utilities Library (libpcp")
29 LINE("libcontract", "Contract Management Library (libcontract, \\\-lcontract")
30 LINE("libcpuc", "CPU Performance Counters Library (libcpuc, \\\-lcpuc)")
31 LINE("libcurses", "Curses Library (libcurses, \\\-lcurses)")
32 LINE("libdat", "Direct Access Transport Library (libdat, \\\-ldat)")
33 LINE("libdevinfo", "Device ID Library (libdevinfo, \\\-ldevinfo)")
34 LINE("libdevinfo", "Device Information Library (libdevinfo, \\\-ldevinfo)")
35 LINE("libdlpi", "Data Link Provider Interface (DLPI) Library (libdlpi, \\\-")
36 LINE("libdns_sd", "DNS Service Discovery Library (libdns_sd, \\\-ldns_sd)")
37 LINE("libelf", "ELF Access Library (libelf, \\\-lelf)")
38 LINE("libexacct", "Extended Accounting File Access Library (libexacct, \\\-")
39 LINE("libfcoe", "FCoE Port Management Library (libfcoe, \\\-lfcoe)")
40 LINE("libfstyp", "File System Type Identification Library (libfstyp, \\\-l")
41 LINE("libgen", "String Pattern Matching Library (libgen, \\\-lgen)")
42 LINE("libgss", "Generic Security Services Library (libgss, \\\-lgss)")
43 LINE("libiscsit", "iSCSI Management Library (libiscsit, \\\-liscsit)")
44 LINE("libkstat", "Kernel Statistics Library (libkstat, \\\-lkstat)")
45 LINE("libkvm", "Kernel VM Library (libkvm, \\\-lkvm)")
46 LINE("libldap", "LDAP Library (libldap, \\\-lldap)")
47 LINE("liblgrp", "Locality Group Library (liblgrp, \\\-llgrp)")
48 LINE("libm", "Mathematical Library (libm, \\\-lm)")
49 LINE("libmail", "User Mailbox Library (libmail, \\\-lmail)")
50 LINE("libmalloc", "Memory Allocation Library (libmalloc, \\\-lmalloc)")
51 LINE("libmd", "Message Digest Library (libmd, \\\-lmd)")
52 LINE("libmp", "Multiple Precision Library (libmp, \\\-lmp)")
53 LINE("libmpapi", "Common Multipath Management Library (libmpapi, \\\-lmpapi)")
54 LINE("libnsl", "Network Services Library (libnsl, \\\-lnsl)")
55 LINE("libnvpair", "Name-Value Pair Library (libnvpair, \\\-lnvpair)")
56 LINE("libpam", "PAM (Pluggable Authentication Module) Library (libpam,")
57 LINE("libpicl", "PICL Library (libpicl, \\\-lpicl)")
58 LINE("libpicltree", "PICL Plug-In Library (libpicltree, \\\-lpicltree)")
59 LINE("libpkcs11", "PKCS#11 Cryptographic Framework Library (libpkcs11, \\\-")

```

```

59 LINE("libpool", "Pool Configuration Manipulation Library (libpool, \\\-lp")
60 LINE("libproc", "Process Control Library (libproc, \\\-lproc)")
61 LINE("libproject", "Project Database Access Library (libproject, \\\-lprojec")
62 LINE("libresolv", "Resolver Library (libresolv, \\\-lresolv \\\-lsocket \\\-l")
63 LINE("librpc", "RPC Service Library (librpcsvc, \\\-lrpc)")
64 LINE("librsm", "Remote Shared Memory Interface Library (librsm, \\\-lrs")
65 LINE("libsasl", "Simple Authentication and Security Library (libsasl, \\\-")
66 LINE("libscf", "Service Configuration Facility Library (libscf, \\\-lscf")
67 LINE("libsec", "File Access Control Library (libsec, \\\-lsec)")
68 LINE("libsecdb", "Security Attributes Database Library (libsecdb, \\\-lsec")
69 LINE("libsip", "Session Initiation Protocol Library (libsip, \\\-lsip)")
70 LINE("libslp", "Service Location Protocol Library (libslp, \\\-lsip)")
71 LINE("libsocket", "Sockets Library (libsocket, \\\-lsocket)")
72 LINE("libstmf", "SCSI Target Mode Framework Library (libstmf, \\\-lstmf)")
73 LINE("libsysv", "System Event Interface Library (libsysv, \\\-lseven")
74 LINE("libtecla", "Interactive Command Line Input Library (libtecla, \\\-lt")
75 LINE("libtntctl", "TNF Probe Control Library (libtntctl, \\\-ltnfctl)")
76 LINE("libtsol", "Trusted Extensions Library (libtsol, \\\-ltsol)")
77 LINE("libuuid", "UUID Library (libuuid, \\\-luuid)")
78 LINE("libvolmgt", "Volume Management Library (libvolmgt, \\\-lvolmgt)")
79 LINE("libxcurses", "X/Open Curses Library (libxcurses, \\\-lxcurses)")
80 LINE("libxnet", "X/Open Networking Library (libxnet, \\\-lxnet)")

```

new/usr/src/cmd/mandoc/libman.h

1

```
*****
3112 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/libman.h
Update to 1.12.3.
*****
1 /* $Id: libman.h,v 1.56 2012/11/17 00:26:33 schwarze Exp $ */
1 /* $Id: libman.h,v 1.55 2011/11/07 01:24:40 schwarze Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifndef LIBMAN_H
18 #define LIBMAN_H
19
20 enum man_next {
21     MAN_NEXT_SIBLING = 0,
22     MAN_NEXT_CHILD
23 };
24
25 unchanged portion omitted
26
42 #define MACRO_PROT_ARGS struct man *man, \
42 #define MACRO_PROT_ARGS struct man *m, \
43 enum mant tok, \
44 int line, \
45 int ppos, \
46 int *pos, \
47 char *buf
48
49 struct man_macro {
50     int (*fp)(MACRO_PROT_ARGS);
51     int flags;
52 #define MAN_SCOPED (1 << 0)
53 #define MAN_EXPLICIT (1 << 1) /* See blk_imp(). */
54 #define MAN_FSCOPED (1 << 2) /* See blk_imp(). */
55 #define MAN_NSOPED (1 << 3) /* See in_line_eoln(). */
56 #define MAN_NOCLOSE (1 << 4) /* See blk_exp(). */
57 #define MAN_BSCOPE (1 << 5) /* Break BLINE scope. */
58 };
59
60 extern const struct man_macro *const man_macros;
61
62 __BEGIN_DECLS
63
64 #define man_pmsg(man, l, p, t) \
65 mandoc_msg((t), (man)->parse, (l), (p), NULL)
66 #define man_nmsg(man, n, t) \
67 mandoc_msg((t), (man)->parse, (n)->line, (n)->pos, NULL)
68 #define man_pmsg(m, l, p, t) \
69 mandoc_msg((t), (m)->parse, (l), (p), NULL)
66 #define man_nmsg(m, n, t) \
67 mandoc_msg((t), (m)->parse, (n)->line, (n)->pos, NULL)
68 int man_word_alloc(struct man *, int, int, const char *);
69 int man_block_alloc(struct man *, int, int, enum mant);
70 int man_head_alloc(struct man *, int, int, enum mant);
71 int man_tail_alloc(struct man *, int, int, enum mant);
```

new/usr/src/cmd/mandoc/libman.h

2

```
72 int man_body_alloc(struct man *, int, int, enum mant);
73 int man_elem_alloc(struct man *, int, int, enum mant);
74 void man_node_delete(struct man *, struct man_node *);
75 void man_hash_init(void);
76 enum mant man_hash_find(const char *);
77 int man_macroend(struct man *);
78 int man_valid_post(struct man *);
79 int man_valid_pre(struct man *, struct man_node *);
80 int man_unscope(struct man *,
81     const struct man_node *, enum mandocerr);
82
83 __END_DECLS
84
85 #endif /* !LIBMAN_H */
```

new/usr/src/cmd/mandoc/libmandoc.h

1

```
*****
3355 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/libmandoc.h
Update to 1.12.3.
*****
1 /* $Id: libmandoc.h,v 1.35 2013/12/15 21:23:52 schwarze Exp $ */
1 /* $Id: libmandoc.h,v 1.29 2011/12/02 01:37:14 schwarze Exp $ */
2 /*
3 * Copyright (c) 2009, 2010, 2011, 2012 Kristaps Dzonsons <kristaps@bsd.lv>
4 * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
3 * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
5 *
6 * Permission to use, copy, modify, and distribute this software for any
7 * purpose with or without fee is hereby granted, provided that the above
8 * copyright notice and this permission notice appear in all copies.
9 *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifndef LIBMANDOC_H
19 #define LIBMANDOC_H

21 enum rofferr {
22     ROFF_CONT, /* continue processing line */
23     ROFF_RERUN, /* re-run roff interpreter with offset */
24     ROFF_APPEND, /* re-run main parser, appending next line */
25     ROFF_REPARSE, /* re-run main parser on the result */
26     ROFF_SO, /* include another file */
27     ROFF_IGN, /* ignore current line */
28     ROFF_TBL, /* a table row was successfully parsed */
29     ROFF_EQN, /* an equation was successfully parsed */
30     ROFF_ERR /* badness: puke and stop */
31 };

32 enum regs {
33     REG_NS = 0, /* nS register */
34     REG_MAX
35 };

33 __BEGIN_DECLS

35 struct roff;
36 struct mdoc;
37 struct man;

39 void mandoc_msg(enum mandocerr, struct mparse *,
40 int, int, const char *);
41 void mandoc_vmsg(enum mandocerr, struct mparse *,
42 int, int, const char *, ...);
43 char *mandoc_getarg(struct mparse *, char **, int, int *);
44 char *mandoc_normdate(struct mparse *, char *, int, int);
45 int mandoc_eos(const char *, size_t, int);
46 int mandoc_getcontrol(const char *, int *);
47 int mandoc_strntoi(const char *, size_t, int);
48 const char *mandoc_a2msec(const char *);

49 void mdoc_free(struct mdoc *);
50 struct mdoc *mdoc_alloc(struct roff *, struct mparse *, char *);
55 struct mdoc *mdoc_alloc(struct roff *, struct mparse *);
51 void mdoc_reset(struct mdoc *);
52 int mdoc_parseln(struct mdoc *, int, char *, int);
```

new/usr/src/cmd/mandoc/libmandoc.h

2

```
53 int mdoc_endparse(struct mdoc *);
54 int mdoc_addspan(struct mdoc *, const struct tbl_span *);
55 int mdoc_addeqn(struct mdoc *, const struct eqn *);

57 void man_free(struct man *);
58 struct man *man_alloc(struct roff *, struct mparse *);
59 void man_reset(struct man *);
60 int man_parseln(struct man *, int, char *, int);
61 int man_endparse(struct man *);
62 int man_addspan(struct man *, const struct tbl_span *);
63 int man_addeqn(struct man *, const struct eqn *);

65 void roff_free(struct roff *);
66 struct roff *roff_alloc(enum mparset, struct mparse *);
71 struct roff *roff_alloc(struct mparse *);
67 void roff_reset(struct roff *);
68 enum rofferr roff_parseln(struct roff *, int,
69 char **, size_t *, int, int *);
70 void roff_endparse(struct roff *);
71 void roff_setreg(struct roff *, const char *, int, char sign);
72 int roff_getreg(const struct roff *, const char *);
76 int roff_regisset(const struct roff *, enum regs);
77 unsigned int roff_regget(const struct roff *, enum regs);
78 void roff_regunset(struct roff *, enum regs);
73 char *roff_strdup(const struct roff *, const char *);
74 int roff_getcontrol(const struct roff *,
75 const char *, int *);

76 #if 0
77 char roff_eqndelim(const struct roff *);
78 void roff_openeqn(struct roff *, const char *,
79 int, int, const char *);
80 int roff_closeeqn(struct roff *);
81 #endif

83 const struct tbl_span *roff_span(const struct roff *);
84 const struct eqn *roff_eqn(const struct roff *);

86 __END_DECLS

88 #endif /* !LIBMANDOC_H */
```

```

*****
5162 Fri Jul 25 15:00:11 2014
new/usr/src/cmd/mandoc/libmdoc.h
Update to 1.12.3.
*****
1 /* $Id: libmdoc.h,v 1.82 2013/10/21 23:47:58 schwarze Exp $ */
1 /* $Id: libmdoc.h,v 1.78 2011/12/02 01:37:14 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifndef LIBMDOC_H
19 #define LIBMDOC_H

21 enum mdoc_next {
22     MDOC_NEXT_SIBLING = 0,
23     MDOC_NEXT_CHILD
24 };

26 struct mdoc {
27     struct mparse *parse; /* parse pointer */
28     char *defos; /* default argument for .Os */
29     int flags; /* parse flags */
30 #define MDOC_HALT (1 << 0) /* error in parse: halt */
31 #define MDOC_LITERAL (1 << 1) /* in a literal scope */
32 #define MDOC_PBODY (1 << 2) /* in the document body */
33 #define MDOC_NEWLINE (1 << 3) /* first macro/text in a line */
34 #define MDOC_PHRASELIT (1 << 4) /* literal within a partila phrase */
35 #define MDOC_PPHRASE (1 << 5) /* within a partial phrase */
36 #define MDOC_FREECOL (1 << 6) /* 'It' invocation should close */
37 #define MDOC_SYNOPSIS (1 << 7) /* SYNOPSIS-style formatting */
38 #define MDOC_KEEP (1 << 8) /* in a word keep */
39 #define MDOC_SMOFF (1 << 9) /* spacing is off */
40     enum mdoc_next next; /* where to put the next node */
41     struct mdoc_node *last; /* the last node parsed */
42     struct mdoc_node *first; /* the first node parsed */
43     struct mdoc_meta meta; /* document meta-data */
44     enum mdoc_sec lastnamed;
45     enum mdoc_sec lastsec;
46     struct roff *roff;
47 };

49 #define MACRO_PROT_ARGS struct mdoc *mdoc, \
45 #define MACRO_PROT_ARGS struct mdoc *m, \
50     enum mdoct tok, \
51     int line, \
52     int ppos, \
53     int *pos, \
54     char *buf

56 struct mdoc_macro {
57     int (*fp)(MACRO_PROT_ARGS);
58     int flags;
59 #define MDOC_CALLABLE (1 << 0)

```

```

60 #define MDOC_PARSED (1 << 1)
61 #define MDOC_EXPLICIT (1 << 2)
62 #define MDOC_PROLOGUE (1 << 3)
63 #define MDOC_IGNDELIM (1 << 4)
64 #define MDOC_JOIN (1 << 5)
65 /* Reserved words in arguments treated as text. */
66 };
67 #define unchanged_portion_omitted

102 extern const struct mdoc_macro *const mdoc_macros;

104 __BEGIN_DECLS

106 #define mdoc_pmsg(mdoc, l, p, t) \
107     mdoc_msg((t), (mdoc)->parse, (l), (p), NULL)
108 #define mdoc_nmsg(mdoc, n, t) \
109     mdoc_msg((t), (mdoc)->parse, (n)->line, (n)->pos, NULL)
102 #define mdoc_pmsg(m, l, p, t) \
103     mdoc_msg((t), (m)->parse, (l), (p), NULL)
104 #define mdoc_nmsg(m, n, t) \
105     mdoc_msg((t), (m)->parse, (n)->line, (n)->pos, NULL)
110 int mdoc_macro(MACRO_PROT_ARGS);
111 int mdoc_word_alloc(struct mdoc *,
112     int, int, const char *);
113 void mdoc_word_append(struct mdoc *, const char *);
114 int mdoc_elem_alloc(struct mdoc *, int, int,
115     enum mdoct, struct mdoc_arg *);
116 int mdoc_block_alloc(struct mdoc *, int, int,
117     enum mdoct, struct mdoc_arg *);
118 int mdoc_head_alloc(struct mdoc *, int, int, enum mdoct);
119 int mdoc_tail_alloc(struct mdoc *, int, int, enum mdoct);
120 int mdoc_body_alloc(struct mdoc *, int, int, enum mdoct);
121 int mdoc_endbody_alloc(struct mdoc *, int, int, enum mdoct,
122     struct mdoc_node *, enum mdoc_endbody);
116 int mdoc_endbody_alloc(struct mdoc *m, int line, int pos,
117     enum mdoct tok, struct mdoc_node *body,
118     enum mdoc_endbody end);
123 void mdoc_node_delete(struct mdoc *, struct mdoc_node *);
124 int mdoc_node_relink(struct mdoc *, struct mdoc_node *);
125 void mdoc_hash_init(void);
126 enum mdoct mdoc_hash_find(const char *);
127 const char *mdoc_a2att(const char *);
128 const char *mdoc_a2lib(const char *);
129 const char *mdoc_a2st(const char *);
130 const char *mdoc_a2arch(const char *);
131 const char *mdoc_a2vol(const char *);
132 int mdoc_valid_pre(struct mdoc *, struct mdoc_node *);
133 int mdoc_valid_post(struct mdoc *);
134 enum margverr mdoc_argv(struct mdoc *, int, enum mdoct,
135     struct mdoc_arg **, int *, char *);
136 void mdoc_argv_free(struct mdoc_arg *);
137 enum margserr mdoc_args(struct mdoc *, int,
138     int *, char *, enum mdoct, char **);
139 enum margserr mdoc_zargs(struct mdoc *, int,
140     int *, char *, char **);
141 int mdoc_macroend(struct mdoc *);
142 enum mdelim mdoc_isdelim(const char *);

144 __END_DECLS

146 #endif /* !LIBMDOC_H */

```



new/usr/src/cmd/mandoc/libroff.h

1

\*\*\*\*\*

2647 Fri Jul 25 15:00:12 2014

new/usr/src/cmd/mandoc/libroff.h

Update to 1.12.3.

\*\*\*\*\*

```
1 /* $Id: libroff.h,v 1.28 2013/05/31 21:37:17 schwarze Exp $ */
1 /* $Id: libroff.h,v 1.27 2011/07/25 15:37:00 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifndef LIBROFF_H
18 #define LIBROFF_H
19
20 __BEGIN_DECLS
21
22 enum tbl_part {
23     TBL_PART_OPTS, /* in options (first line) */
24     TBL_PART_LAYOUT, /* describing layout */
25     TBL_PART_DATA, /* creating data rows */
26     TBL_PART_CDATA /* continue previous row */
27 };
28
29 struct tbl_node {
30     struct mparse *parse; /* parse point */
31     int pos; /* invocation column */
32     int line; /* invocation line */
33     enum tbl_part part;
34     struct tbl_opts opts;
35     struct tbl_row *first_row;
36     struct tbl_row *last_row;
37     struct tbl_span *first_span;
38     struct tbl_span *current_span;
39     struct tbl_span *last_span;
40     struct tbl_head *first_head;
41     struct tbl_head *last_head;
42     struct tbl_node *next;
43 };
44
45 _____
46 unchanged_portion_omitted

```



```

136         version();
137         /* NOTREACHED */
138     default:
139         usage();
140         /* NOTREACHED */
141     }

```

```

143     curp.mp = mparse_alloc(type, curp.wlevel, mmsg, &curp, defos);
144     curp.mp = mparse_alloc(type, curp.wlevel, mmsg, &curp);

```

```

145     /*
146     * Conditionally start up the lookaside buffer before parsing.
147     */
148     if (OUTT_MAN == curp.outtype)
149         mparse_keep(curp.mp);

```

```

151     argc -= optind;
152     argv += optind;

```

```

154     rc = MANDOCLEVEL_OK;

```

```

156     if (NULL == *argv)
157         parse(&curp, STDIN_FILENO, "<stdin>", &rc);

```

```

159     while (*argv) {
160         parse(&curp, -1, *argv, &rc);
161         if (MANDOCLEVEL_OK != rc && curp.wstop)
162             break;
163         ++argv;
164     }

```

```

166     if (curp.outfree)
167         (*curp.outfree)(curp.outdata);
168     if (curp.mp)
169         mparse_free(curp.mp);
170     free(defos);

```

```

172     return((int)rc);
173 }

```

unchanged portion omitted

```

183 static void
184 usage(void)
185 {

```

```

187     fprintf(stderr, "usage: %s "
188                "[-V] "
189                "[-Ios=name] "
190                "[-foption] "
191                "[-mformat] "
192                "[-Ooption] "
193                "[-Toutput] "
194                "[-Wlevel]\n"
195                "\t\t [file ...]\n",
196                progname);

```

```

197     exit((int)MANDOCLEVEL_BADARG);
198 }

```

unchanged portion omitted

```

*****
14223 Fri Jul 25 15:00:12 2014
new/usr/src/cmd/mandoc/man.c
Update to 1.12.3.
*****
1 /*      $Id: man.c,v 1.121 2013/11/10 22:54:40 schwarze Exp $ */
1 /*      $Id: man.c,v 1.115 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif
21 #include <sys/types.h>
23 #include <assert.h>
24 #include <stdarg.h>
25 #include <stdlib.h>
26 #include <stdio.h>
27 #include <string.h>
29 #include "man.h"
30 #include "mandoc.h"
31 #include "libman.h"
32 #include "libmandoc.h"
34 const  char *const __man_macronames[MAN_MAX] = {
35     "br",      "TH",      "SH",      "SS",
36     "TP",      "LP",      "PP",      "P",
37     "IP",      "HP",      "SM",      "SB",
38     "BI",      "IB",      "BR",      "RB",
39     "R",       "B",       "I",       "IR",
40     "RI",      "na",     "sp",     "nf",
41     "fi",      "RE",     "RS",     "DT",
42     "UC",      "PD",     "AT",     "in",
43     "ft",      "OP",     "EX",     "EE",
44     "UR",      "UE",
43     "ft",      "OP"
45     };
47 const  char * const *man_macronames = __man_macronames;
49 static struct man_node *man_node_alloc(struct man *, int, int,
50     enum man_type, enum mant);
51 static int  man_node_append(struct man *,
52     struct man_node *);
53 static void man_node_free(struct man_node *);
54 static void man_node_unlink(struct man *,
55     struct man_node *);
56 static int  man_ptext(struct man *, int, char *, int);
57 static int  man_pmacro(struct man *, int, char *, int);
58 static void man_freel(struct man *);
59 static void man_alloc1(struct man *);

```

```

60 static int  man_descope(struct man *, int, int);
63 const struct man_node *
64 man_node(const struct man *man)
65 {
67     assert( ! (MAN_HALT & man->flags));
68     return(man->first);
66     assert( ! (MAN_HALT & m->flags));
67     return(m->first);
69 }
72 const struct man_meta *
73 man_meta(const struct man *man)
74 {
76     assert( ! (MAN_HALT & man->flags));
77     return(&man->meta);
75     assert( ! (MAN_HALT & m->flags));
76     return(&m->meta);
78 }
_____unchanged_portion_omitted_____
115 int
116 man_endparse(struct man *man)
117 man_endparse(struct man *m)
117 {
119     assert( ! (MAN_HALT & man->flags));
120     if (man_macroend(man))
118     assert( ! (MAN_HALT & m->flags));
119     if (man_macroend(m))
121         return(1);
122     man->flags |= MAN_HALT;
121     m->flags |= MAN_HALT;
123     return(0);
124 }
127 int
128 man_parseln(struct man *man, int ln, char *buf, int offs)
127 man_parseln(struct man *m, int ln, char *buf, int offs)
129 {
131     man->flags |= MAN_NEWLINE;
130     m->flags |= MAN_NEWLINE;
133     assert( ! (MAN_HALT & man->flags));
132     assert( ! (MAN_HALT & m->flags));
135     return (roff_getcontrol(man->roff, buf, &offs) ?
136         man_pmacro(man, ln, buf, offs) :
137         man_ptext(man, ln, buf, offs));
134     return (mandoc_getcontrol(buf, &offs) ?
135         man_pmacro(m, ln, buf, offs) :
136         man_ptext(m, ln, buf, offs));
138 }
_____unchanged_portion_omitted_____
160 static void

```

```

161 man_alloc1(struct man *man)
160 man_alloc1(struct man *m)
162 {

164     memset(&man->meta, 0, sizeof(struct man_meta));
165     man->flags = 0;
166     man->last = mandoc_calloc(1, sizeof(struct man_node));
167     man->first = man->last;
168     man->last->type = MAN_ROOT;
169     man->last->tok = MAN_MAX;
170     man->next = MAN_NEXT_CHILD;
163     memset(&m->meta, 0, sizeof(struct man_meta));
164     m->flags = 0;
165     m->last = mandoc_calloc(1, sizeof(struct man_node));
166     m->first = m->last;
167     m->last->type = MAN_ROOT;
168     m->last->tok = MAN_MAX;
169     m->next = MAN_NEXT_CHILD;
171 }

    unchanged_portion_omitted

237 static struct man_node *
238 man_node_alloc(struct man *man, int line, int pos,
237 man_node_alloc(struct man *m, int line, int pos,
239                 enum man_type type, enum mant tok)
240 {
241     struct man_node *p;

243     p = mandoc_calloc(1, sizeof(struct man_node));
244     p->line = line;
245     p->pos = pos;
246     p->type = type;
247     p->tok = tok;

249     if (MAN_NEWLINE & man->flags)
248     if (MAN_NEWLINE & m->flags)
250         p->flags |= MAN_LINE;
251     man->flags &= ~MAN_NEWLINE;
250     m->flags &= ~MAN_NEWLINE;
252     return(p);
253 }

256 int
257 man_elem_alloc(struct man *man, int line, int pos, enum mant tok)
256 man_elem_alloc(struct man *m, int line, int pos, enum mant tok)
258 {
259     struct man_node *p;

261     p = man_node_alloc(man, line, pos, MAN_ELEM, tok);
262     if ( ! man_node_append(man, p) )
260     p = man_node_alloc(m, line, pos, MAN_ELEM, tok);
261     if ( ! man_node_append(m, p) )
263         return(0);
264     man->next = MAN_NEXT_CHILD;
263     m->next = MAN_NEXT_CHILD;
265     return(1);
266 }

269 int
270 man_tail_alloc(struct man *man, int line, int pos, enum mant tok)
269 man_tail_alloc(struct man *m, int line, int pos, enum mant tok)
271 {
272     struct man_node *p;

```

```

274     p = man_node_alloc(man, line, pos, MAN_TAIL, tok);
275     if ( ! man_node_append(man, p) )
273     p = man_node_alloc(m, line, pos, MAN_TAIL, tok);
274     if ( ! man_node_append(m, p) )
276         return(0);
277     man->next = MAN_NEXT_CHILD;
276     m->next = MAN_NEXT_CHILD;
278     return(1);
279 }

282 int
283 man_head_alloc(struct man *man, int line, int pos, enum mant tok)
282 man_head_alloc(struct man *m, int line, int pos, enum mant tok)
284 {
285     struct man_node *p;

287     p = man_node_alloc(man, line, pos, MAN_HEAD, tok);
288     if ( ! man_node_append(man, p) )
286     p = man_node_alloc(m, line, pos, MAN_HEAD, tok);
287     if ( ! man_node_append(m, p) )
289         return(0);
290     man->next = MAN_NEXT_CHILD;
289     m->next = MAN_NEXT_CHILD;
291     return(1);
292 }

295 int
296 man_body_alloc(struct man *man, int line, int pos, enum mant tok)
295 man_body_alloc(struct man *m, int line, int pos, enum mant tok)
297 {
298     struct man_node *p;

300     p = man_node_alloc(man, line, pos, MAN_BODY, tok);
301     if ( ! man_node_append(man, p) )
299     p = man_node_alloc(m, line, pos, MAN_BODY, tok);
300     if ( ! man_node_append(m, p) )
302         return(0);
303     man->next = MAN_NEXT_CHILD;
302     m->next = MAN_NEXT_CHILD;
304     return(1);
305 }

308 int
309 man_block_alloc(struct man *man, int line, int pos, enum mant tok)
308 man_block_alloc(struct man *m, int line, int pos, enum mant tok)
310 {
311     struct man_node *p;

313     p = man_node_alloc(man, line, pos, MAN_BLOCK, tok);
314     if ( ! man_node_append(man, p) )
312     p = man_node_alloc(m, line, pos, MAN_BLOCK, tok);
313     if ( ! man_node_append(m, p) )
315         return(0);
316     man->next = MAN_NEXT_CHILD;
315     m->next = MAN_NEXT_CHILD;
317     return(1);
318 }

320 int
321 man_word_alloc(struct man *man, int line, int pos, const char *word)
320 man_word_alloc(struct man *m, int line, int pos, const char *word)
322 {

```

```

323     struct man_node *n;

325     n = man_node_alloc(man, line, pos, MAN_TEXT, MAN_MAX);
326     n->string = roff_strdup(man->roff, word);
324     n = man_node_alloc(m, line, pos, MAN_TEXT, MAN_MAX);
325     n->string = roff_strdup(m->roff, word);

328     if ( ! man_node_append(man, n))
327     if ( ! man_node_append(m, n))
329         return(0);

331     man->next = MAN_NEXT_SIBLING;
330     m->next = MAN_NEXT_SIBLING;
332     return(1);
333 }
_____unchanged_portion_omitted_____

350 void
351 man_node_delete(struct man *man, struct man_node *p)
350 man_node_delete(struct man *m, struct man_node *p)
352 {

354     while (p->child)
355         man_node_delete(man, p->child);
354         man_node_delete(m, p->child);

357     man_node_unlink(man, p);
356     man_node_unlink(m, p);
358     man_node_free(p);
359 }

361 int
362 man_addeqn(struct man *man, const struct eqn *ep)
361 man_addeqn(struct man *m, const struct eqn *ep)
363 {
364     struct man_node *n;

366     assert( ! (MAN_HALT & man->flags));
365     assert( ! (MAN_HALT & m->flags));

368     n = man_node_alloc(man, ep->ln, ep->pos, MAN_EQN, MAN_MAX);
367     n = man_node_alloc(m, ep->ln, ep->pos, MAN_EQN, MAN_MAX);
369     n->eqn = ep;

371     if ( ! man_node_append(man, n))
370     if ( ! man_node_append(m, n))
372         return(0);

374     man->next = MAN_NEXT_SIBLING;
375     return(man_descope(man, ep->ln, ep->pos));
373     m->next = MAN_NEXT_SIBLING;
374     return(man_descope(m, ep->ln, ep->pos));
376 }

378 int
379 man_addspan(struct man *man, const struct tbl_span *sp)
378 man_addspan(struct man *m, const struct tbl_span *sp)
380 {
381     struct man_node *n;

383     assert( ! (MAN_HALT & man->flags));
382     assert( ! (MAN_HALT & m->flags));

385     n = man_node_alloc(man, sp->line, 0, MAN_TBL, MAN_MAX);
384     n = man_node_alloc(m, sp->line, 0, MAN_TBL, MAN_MAX);

```

```

386     n->span = sp;

388     if ( ! man_node_append(man, n))
387     if ( ! man_node_append(m, n))
389         return(0);

391     man->next = MAN_NEXT_SIBLING;
392     return(man_descope(man, sp->line, 0));
390     m->next = MAN_NEXT_SIBLING;
391     return(man_descope(m, sp->line, 0));
393 }

395 static int
396 man_descope(struct man *man, int line, int offs)
395 man_descope(struct man *m, int line, int offs)
397 {
398     /*
399     * Co-ordinate what happens with having a next-line scope open:
400     * first close out the element scope (if applicable), then close
401     * out the block scope (also if applicable).
402     */

404     if (MAN_ELINE & man->flags) {
405         man->flags &= ~MAN_ELINE;
406         if ( ! man_unscope(man, man->last->parent, MANDOCERR_MAX))
403     if (MAN_ELINE & m->flags) {
404         m->flags &= ~MAN_ELINE;
405         if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
407             return(0);
408     }

410     if ( ! (MAN_BLINE & man->flags))
409     if ( ! (MAN_BLINE & m->flags))
411         return(1);
412     man->flags &= ~MAN_BLINE;
411     m->flags &= ~MAN_BLINE;

414     if ( ! man_unscope(man, man->last->parent, MANDOCERR_MAX))
413     if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
415         return(0);
416     return(man_body_alloc(man, line, offs, man->last->tok));
415     return(man_body_alloc(m, line, offs, m->last->tok));
417 }

419 static int
420 man_ptext(struct man *man, int line, char *buf, int offs)
419 man_ptext(struct man *m, int line, char *buf, int offs)
421 {
422     int         i;

424     /* Literal free-form text whitespace is preserved. */

426     if (MAN_LITERAL & man->flags) {
427         if ( ! man_word_alloc(man, line, offs, buf + offs))
425     if (MAN_LITERAL & m->flags) {
426         if ( ! man_word_alloc(m, line, offs, buf + offs))
428             return(0);
429         return(man_descope(man, line, offs));
428         return(man_descope(m, line, offs));
430     }

431     /* Pump blank lines directly into the backend. */

432     for (i = offs; ' ' == buf[i]; i++)
433         /* Skip leading whitespace. */ ;

```

```

435  /*
436  * Blank lines are ignored right after headings
437  * but add a single vertical space elsewhere.
438  */

440  if ('\0' == buf[i]) {
441      /* Allocate a blank entry. */
442      if (MAN_SH != man->last->tok &&
443          MAN_SS != man->last->tok) {
444          if (! man_elem_alloc(man, line, offs, MAN_sp))
445              return(0);
446          man->next = MAN_NEXT_SIBLING;
447          return(man_descope(m, line, offs));
448      }
449  }

451  /*
452  * Warn if the last un-escaped character is whitespace. Then
453  * strip away the remaining spaces (tabs stay!).
454  */

456  i = (int)strlen(buf);
457  assert(i);

459  if ( ' ' == buf[i - 1] || '\t' == buf[i - 1] ) {
460      if (i > 1 && '\\\'' != buf[i - 2])
461          man_pmsg(man, line, i - 1, MANDOCERR_EOLNSPACE);
462      man_pmsg(m, line, i - 1, MANDOCERR_EOLNSPACE);

463      for (--i; i && ' ' == buf[i]; i--)
464          /* Spin back to non-space. */ ;

466      /* Jump ahead of escaped whitespace. */
467      i += '\\\'' == buf[i] ? 2 : 1;

469      buf[i] = '\0';
470  }

472  if (! man_word_alloc(man, line, offs, buf + offs))
473      if (! man_word_alloc(m, line, offs, buf + offs))
474          return(0);

475  /*
476  * End-of-sentence check. If the last character is an unescaped
477  * EOS character, then flag the node as being the end of a
478  * sentence. The front-end will know how to interpret this.
479  */

481  assert(i);
482  if (mandoc_eos(buf, (size_t)i, 0))
483      man->last->flags |= MAN_EOS;
484      m->last->flags |= MAN_EOS;

485  return(man_descope(man, line, offs));
486  return(man_descope(m, line, offs));
487  }

488  static int
489  man_pmacro(struct man *man, int ln, char *buf, int offs)
490  {
491      int          i, ppos;
492      enum mant    tok;
493      char         mac[5];

```

```

494      struct man_node *n;

496      if ( ' ' == buf[offs] ) {
497          man_pmsg(man, ln, offs, MANDOCERR_BADCOMMENT);
498          man_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
499          return(1);
500      } else if ('\0' == buf[offs])
501          return(1);

502      ppos = offs;

504      /*
505      * Copy the first word into a nil-terminated buffer.
506      * Stop copying when a tab, space, or eoln is encountered.
507      */

509      i = 0;
510      while (i < 4 && '\0' != buf[offs] &&
511             ' ' != buf[offs] && '\t' != buf[offs])
512          mac[i++] = buf[offs++];

514      mac[i] = '\0';

516      tok = (i > 0 && i < 4) ? man_hash_find(mac) : MAN_MAX;

518      if (MAN_MAX == tok) {
519          mandoc_vmsg(MANDOCERR_MACRO, man->parse, ln,
520                    mandoc_vmsg(MANDOCERR_MACRO, m->parse, ln,
521                                ppos, "%s", buf + ppos - 1));
522          return(1);
523      }

524      /* The macro is sane. Jump to the next word. */

526      while (buf[offs] && ' ' == buf[offs])
527          offs++;

529      /*
530      * Trailing whitespace. Note that tabs are allowed to be passed
531      * into the parser as "text", so we only warn about spaces here.
532      */

534      if ('\0' == buf[offs] && ' ' == buf[offs - 1])
535          man_pmsg(man, ln, offs - 1, MANDOCERR_EOLNSPACE);
536          man_pmsg(m, ln, offs - 1, MANDOCERR_EOLNSPACE);

537      /*
538      * Remove prior ELINE macro, as it's being clobbered by a new
539      * macro. Note that NSCOPEd macros do not close out ELINE
540      * macros---they don't print text---so we let those slip by.
541      */

543      if (! (MAN_NSCOPEd & man_macros[tok].flags) &&
544            man->flags & MAN_ELINE) {
545          n = man->last;
546          m->flags & MAN_ELINE) {
547              n = m->last;
548              assert(MAN_TEXT != n->type);

549          /* Remove repeated NSCOPEd macros causing ELINE. */

550          if (MAN_NSCOPEd & man_macros[n->tok].flags)
551              n = n->parent;

553          mandoc_vmsg(MANDOCERR_LINESCOPE, man->parse, n->line,
554                    mandoc_vmsg(MANDOCERR_LINESCOPE, m->parse, n->line,

```

```

554         n->pos, "%s breaks %s", man_macronames[tok],
555         man_macronames[n->tok]);

557     man_node_delete(man, n);
558     man->flags &= ~MAN_ELINE;
559     man_node_delete(m, n);
560     m->flags &= ~MAN_ELINE;
561 }

562 /*
563  * Remove prior BLINE macro that is being clobbered.
564  */
565 if ((man->flags & MAN_BLINE) &&
566     if ((m->flags & MAN_BLINE) &&
567         (MAN_BSCOPE & man_macros[tok].flags)) {
568     n = man->last;
569     n = m->last;

570     /* Might be a text node like 8 in
571      * .TP 8
572      * .SH foo
573      */
574     if (MAN_TEXT == n->type)
575         n = n->parent;

576     /* Remove element that didn't end BLINE, if any. */
577     if ( ! (MAN_BSCOPE & man_macros[n->tok].flags))
578         n = n->parent;

579     assert(MAN_HEAD == n->type);
580     n = n->parent;
581     assert(MAN_BLOCK == n->type);
582     assert(MAN_SCOPED & man_macros[n->tok].flags);

583     mandoc_vmsg(MANDOCERR_LINESCOPE, man->parse, n->line,
584     mandoc_vmsg(MANDOCERR_LINESCOPE, m->parse, n->line,
585     n->pos, "%s breaks %s", man_macronames[tok],
586     man_macronames[n->tok]);

587     man_node_delete(man, n);
588     man->flags &= ~MAN_BLINE;
589     man_node_delete(m, n);
590     m->flags &= ~MAN_BLINE;
591 }

592 /*
593  * Save the fact that we're in the next-line for a block. In
594  * this way, embedded roff instructions can "remember" state
595  * when they exit.
596  */

597 if (MAN_BLINE & man->flags)
598     man->flags |= MAN_BPLINE;
599 if (MAN_BLINE & m->flags)
600     m->flags |= MAN_BPLINE;

601 /* Call to handler... */

602 assert(man_macros[tok].fp);
603 if ( ! (*man_macros[tok].fp)(man, tok, ln, ppos, &offs, buf))
604     if ( ! (*man_macros[tok].fp)(m, tok, ln, ppos, &offs, buf))
605         goto err;

606 /*
607  * We weren't in a block-line scope when entering the
608  * above-parsed macro, so return.

```

```

610     */

611     if ( ! (MAN_BPLINE & man->flags)) {
612         man->flags &= ~MAN_ILINE;
613     if ( ! (MAN_BPLINE & m->flags)) {
614         m->flags &= ~MAN_ILINE;
615         return(1);
616     }
617     man->flags &= ~MAN_BPLINE;
618     m->flags &= ~MAN_BPLINE;

619     /*
620      * If we're in a block scope, then allow this macro to slip by
621      * without closing scope around it.
622     */

623     if (MAN_ILINE & man->flags) {
624         man->flags &= ~MAN_ILINE;
625     if (MAN_ILINE & m->flags) {
626         m->flags &= ~MAN_ILINE;
627         return(1);
628     }

629     /*
630      * If we've opened a new next-line element scope, then return
631      * now, as the next line will close out the block scope.
632     */

633     if (MAN_ELINE & man->flags)
634         if (MAN_ELINE & m->flags)
635             return(1);

636     /* Close out the block scope opened in the prior line. */

637     assert(MAN_BLINE & man->flags);
638     man->flags &= ~MAN_BLINE;
639     assert(MAN_BLINE & m->flags);
640     m->flags &= ~MAN_BLINE;

641     if ( ! man_unscope(man, man->last->parent, MANDOCERR_MAX))
642         if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
643             return(0);
644     return(man_body_alloc(man, ln, ppos, man->last->tok));
645     return(man_body_alloc(m, ln, ppos, m->last->tok));

646 err:    /* Error out. */

647     man->flags |= MAN_HALT;
648     m->flags |= MAN_HALT;
649     return(0);
650 }

651 /*
652  * Unlink a node from its context. If "man" is provided, the last parse
653  * Unlink a node from its context. If "m" is provided, the last parse
654  * point will also be adjusted accordingly.
655  */
656 static void
657 man_node_unlink(struct man *man, struct man_node *n)
658 man_node_unlink(struct man *m, struct man_node *n)
659 {

660     /* Adjust siblings. */

661     if (n->prev)
662         n->prev->next = n->next;

```



```
663     if (n->next)
664         n->next->prev = n->prev;
665
666     /* Adjust parent. */
667
668     if (n->parent) {
669         n->parent->nchild--;
670         if (n->parent->child == n)
671             n->parent->child = n->prev ? n->prev : n->next;
672     }
673
674     /* Adjust parse point, if applicable. */
675
676     if (man && man->last == n) {
677         if (m && m->last == n) {
678             /*XXX: this can occur when bailing from validation. */
679             /*assert(NULL == n->next);*/
680             if (n->prev) {
681                 man->last = n->prev;
682                 man->next = MAN_NEXT_SIBLING;
683                 m->last = n->prev;
684                 m->next = MAN_NEXT_SIBLING;
685             } else {
686                 man->last = n->parent;
687                 man->next = MAN_NEXT_CHILD;
688                 m->last = n->parent;
689                 m->next = MAN_NEXT_CHILD;
690             }
691         }
692     }
693
694     if (man && man->first == n)
695         man->first = NULL;
696     if (m && m->first == n)
697         m->first = NULL;
698 }
699
700 const struct mparse *
701 man_mparse(const struct man *man)
702 man_mparse(const struct man *m)
703 {
704     assert(man && man->parse);
705     return(man->parse);
706 }
707
708 const struct mparse *
709 man_mparse(const struct man *m)
710 {
711     assert(m && m->parse);
712     return(m->parse);
713 }
714
715 unchanged_portion_omitted
```

new/usr/src/cmd/mandoc/man.h

1

\*\*\*\*\*

2731 Fri Jul 25 15:00:12 2014

new/usr/src/cmd/mandoc/man.h

Update to 1.12.3.

\*\*\*\*\*

```
1 /* $Id: man.h,v 1.62 2013/10/17 20:54:58 schwarze Exp $ */
1 /* $Id: man.h,v 1.60 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifndef MAN_H
18 #define MAN_H
19
20 enum mant {
21     MAN_br = 0,
22     MAN_th,
23     MAN_sh,
24     MAN_ss,
25     MAN_tp,
26     MAN_lp,
27     MAN_pp,
28     MAN_p,
29     MAN_ip,
30     MAN_hp,
31     MAN_sm,
32     MAN_sb,
33     MAN_bi,
34     MAN_ib,
35     MAN_br,
36     MAN_rb,
37     MAN_r,
38     MAN_b,
39     MAN_i,
40     MAN_ir,
41     MAN_ri,
42     MAN_na,
43     MAN_sp,
44     MAN_nf,
45     MAN_fi,
46     MAN_re,
47     MAN_rs,
48     MAN_dt,
49     MAN_uc,
50     MAN_pd,
51     MAN_at,
52     MAN_in,
53     MAN_ft,
54     MAN_op,
55     MAN_ex,
56     MAN_ee,
57     MAN_ur,
58     MAN_ue,
59     MAN_max
60 };
```

unchanged portion omitted

```

*****
14617 Fri Jul 25 15:00:12 2014
new/usr/src/cmd/mandoc/man_html.c
Update to 1.12.3.
*****
1 /* $Id: man_html.c,v 1.90 2013/10/17 20:54:58 schwarze Exp $ */
1 /* $Id: man_html.c,v 1.86 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29
30 #include "mandoc.h"
31 #include "out.h"
32 #include "html.h"
33 #include "man.h"
34 #include "main.h"
35
36 /* TODO: preserve ident widths. */
37 /* FIXME: have PD set the default vspace width. */
38
39 #define INDENT          5
40
41 #define MAN_ARGS        const struct man_meta *man, \
42 #define MAN_ARGS        const struct man_meta *m, \
43                        const struct man_node *n, \
44                        struct mhtml *mh, \
45                        struct html *h
46
47 struct mhtml {
48     int          fl;
49 #define MANH_LITERAL    (1 << 0) /* literal context */
50 };
51
52 unchanged_portion_omitted
53
54 static void          print_bvspace(struct html *,
55                                 const struct man_node *);
56 static void          print_man(MAN_ARGS);
57 static void          print_man_head(MAN_ARGS);
58 static void          print_man_nodelist(MAN_ARGS);
59 static void          print_man_node(MAN_ARGS);
60 static int           a2width(const struct man_node *,

```

```

63                                 struct roffsu *);
64 static int          man_B_pre(MAN_ARGS);
65 static int          man_HP_pre(MAN_ARGS);
66 static int          man_IP_pre(MAN_ARGS);
67 static int          man_I_pre(MAN_ARGS);
68 static int          man_OP_pre(MAN_ARGS);
69 static int          man_PP_pre(MAN_ARGS);
70 static int          man_RS_pre(MAN_ARGS);
71 static int          man_SH_pre(MAN_ARGS);
72 static int          man_SM_pre(MAN_ARGS);
73 static int          man_SS_pre(MAN_ARGS);
74 static int          man_UR_pre(MAN_ARGS);
75 static int          man_alt_pre(MAN_ARGS);
76 static int          man_br_pre(MAN_ARGS);
77 static int          man_ign_pre(MAN_ARGS);
78 static int          man_in_pre(MAN_ARGS);
79 static int          man_literal_pre(MAN_ARGS);
80 static void         man_root_post(MAN_ARGS);
81 static void         man_root_pre(MAN_ARGS);
82
83 static const struct htmlman mans[MAN_MAX] = {
84     { man_br_pre, NULL }, /* br */
85     { NULL, NULL }, /* TH */
86     { man_SH_pre, NULL }, /* SH */
87     { man_SS_pre, NULL }, /* SS */
88     { man_IP_pre, NULL }, /* TP */
89     { man_PP_pre, NULL }, /* LP */
90     { man_PP_pre, NULL }, /* PP */
91     { man_PP_pre, NULL }, /* P */
92     { man_IP_pre, NULL }, /* IP */
93     { man_HP_pre, NULL }, /* HP */
94     { man_SM_pre, NULL }, /* SM */
95     { man_SM_pre, NULL }, /* SB */
96     { man_alt_pre, NULL }, /* BI */
97     { man_alt_pre, NULL }, /* IB */
98     { man_alt_pre, NULL }, /* BR */
99     { man_alt_pre, NULL }, /* RB */
100    { NULL, NULL }, /* R */
101    { man_B_pre, NULL }, /* B */
102    { man_I_pre, NULL }, /* I */
103    { man_alt_pre, NULL }, /* IR */
104    { man_alt_pre, NULL }, /* RI */
105    { man_ign_pre, NULL }, /* na */
106    { man_br_pre, NULL }, /* sp */
107    { man_literal_pre, NULL }, /* nf */
108    { man_literal_pre, NULL }, /* fi */
109    { NULL, NULL }, /* RE */
110    { man_RS_pre, NULL }, /* RS */
111    { man_ign_pre, NULL }, /* DT */
112    { man_ign_pre, NULL }, /* UC */
113    { man_ign_pre, NULL }, /* PD */
114    { man_ign_pre, NULL }, /* AT */
115    { man_in_pre, NULL }, /* in */
116    { man_ign_pre, NULL }, /* ft */
117    { man_OP_pre, NULL }, /* OP */
118    { man_literal_pre, NULL }, /* EX */
119    { man_literal_pre, NULL }, /* EE */
120    { man_UR_pre, NULL }, /* UR */
121    { NULL, NULL }, /* UE */
122 };
123
124 unchanged_portion_omitted
125
126 void
127 html_man(void *arg, const struct man *man)
128 html_man(void *arg, const struct man *m)
129 {

```

```

150     struct mhtml    mh;

152     memset(&mh, 0, sizeof(struct mhtml));
153     print_man(man_meta(man), man_node(man), &mh, (struct html *)arg);
154     print_man(man_meta(m), man_node(m), &mh, (struct html *)arg);
155     putchar('\n');
156 }

157 static void
158 print_man(MAN_ARGS)
159 {
160     struct tag      *t, *tt;
161     struct htmlpair tag;

163     PAIR_CLASS_INIT(&tag, "mandoc");

165     if (! (HTML_FRAGMENT & h->oflags)) {
166         print_gen_decls(h);
167         t = print_otag(h, TAG_HTML, 0, NULL);
168         tt = print_otag(h, TAG_HEAD, 0, NULL);
169         print_man_head(man, n, mh, h);
170         print_man_head(m, n, mh, h);
171         print_tagq(h, tt);
172         print_otag(h, TAG_BODY, 0, NULL);
173         print_otag(h, TAG_DIV, 1, &tag);
174     } else
175         t = print_otag(h, TAG_DIV, 1, &tag);

176     print_man_nodelist(man, n, mh, h);
177     print_man_nodelist(m, n, mh, h);
178     print_tagq(h, t);

181 /* ARGSUSED */
182 static void
183 print_man_head(MAN_ARGS)
184 {
186     print_gen_head(h);
187     assert(man->title);
188     assert(man->msec);
189     bufcat_fmt(h, "%s(%s)", man->title, man->msec);
190     assert(m->title);
191     assert(m->msec);
192     bufcat_fmt(h, "%s(%s)", m->title, m->msec);
193     print_otag(h, TAG_TITLE, 0, NULL);
194     print_text(h, h->buf);
195 }

196 static void
197 print_man_nodelist(MAN_ARGS)
198 {
199     print_man_node(man, n, mh, h);
200     print_man_node(m, n, mh, h);
201     if (n->next)
202         print_man_nodelist(man, n->next, mh, h);
203     print_man_nodelist(m, n->next, mh, h);
204 }

205 static void
206 print_man_node(MAN_ARGS)
207 {

```

```

208     int             child;
209     struct tag      *t;

211     child = 1;
212     t = h->tags.head;

214     switch (n->type) {
215     case (MAN_ROOT):
216         man_root_pre(man, n, mh, h);
217         man_root_pre(m, n, mh, h);
218         break;
219     case (MAN_TEXT):
220         /*
221          * If we have a blank line, output a vertical space.
222          * If we have a space as the first character, break
223          * before printing the line's data.
224          */
225         if ('\0' == *n->string) {
226             print_otag(h, TAG_P, 0, NULL);
227             return;
228         }
229         if (' ' == *n->string && MAN_LINE & n->flags)
230             print_otag(h, TAG_BR, 0, NULL);
231         else if (MANH_LITERAL & mh->fl && n->prev)
232             print_otag(h, TAG_BR, 0, NULL);

233         print_text(h, n->string);
234         return;
235     case (MAN_EQN):
236         print_eqn(h, n->eqn);
237         break;
238     case (MAN_TBL):
239         /*
240          * This will take care of initialising all of the table
241          * state data for the first table, then tearing it down
242          * for the last one.
243          */
244         print_tbl(h, n->span);
245         return;
246     default:
247         /*
248          * Close out scope of font prior to opening a macro
249          * scope.
250          */
251         if (HTMLFONT_NONE != h->metac) {
252             h->metal = h->metac;
253             h->metac = HTMLFONT_NONE;
254         }

255         /*
256          * Close out the current table, if it's open, and unset
257          * the "meta" table state. This will be reopened on the
258          * next table element.
259          */
260         if (h->tblt) {
261             print_tblclose(h);
262             t = h->tags.head;
263         }
264         if (mans[n->tok].pre)
265             child = (*mans[n->tok].pre)(man, n, mh, h);
266         child = (*mans[n->tok].pre)(m, n, mh, h);
267         break;
268     }

269     if (child && n->child)

```

```

272     print_man_nodelist(man, n->child, mh, h);
266     print_man_nodelist(m, n->child, mh, h);

274     /* This will automatically close out any font scope. */
275     print_stagq(h, t);

277     switch (n->type) {
278     case (MAN_ROOT):
279         man_root_post(man, n, mh, h);
273         man_root_post(m, n, mh, h);
280         break;
281     case (MAN_EQN):
282         break;
283     default:
284         if (mans[n->tok].post)
285             (*mans[n->tok].post)(man, n, mh, h);
279             (*mans[n->tok].post)(m, n, mh, h);
286         break;
287     }
288 }
    unchanged_portion_omitted_

304 /* ARGSUSED */
305 static void
306 man_root_pre(MAN_ARGS)
307 {
308     struct htmlpair tag[3];
309     struct tag      *t, *tt;
310     char            b[BUFSIZ], title[BUFSIZ];

312     b[0] = 0;
313     if (man->vol)
314         (void)strlcat(b, man->vol, BUFSIZ);
307     if (m->vol)
308         (void)strlcat(b, m->vol, BUFSIZ);

316     assert(man->title);
317     assert(man->msec);
318     snprintf(title, BUFSIZ - 1, "%s(%s)", man->title, man->msec);
310     assert(m->title);
311     assert(m->msec);
312     snprintf(title, BUFSIZ - 1, "%s(%s)", m->title, m->msec);

320     PAIR_SUMMARY_INIT(&tag[0], "Document Header");
321     PAIR_CLASS_INIT(&tag[1], "head");
322     PAIR_INIT(&tag[2], ATTR_WIDTH, "100%");
323     t = print_otag(h, TAG_TABLE, 3, tag);
324     PAIR_INIT(&tag[0], ATTR_WIDTH, "30%");
325     print_otag(h, TAG_COL, 1, tag);
326     print_otag(h, TAG_COL, 1, tag);
327     print_otag(h, TAG_COL, 1, tag);

329     print_otag(h, TAG_TBODY, 0, NULL);

331     tt = print_otag(h, TAG_TR, 0, NULL);

333     PAIR_CLASS_INIT(&tag[0], "head-ltitle");
334     print_otag(h, TAG_TD, 1, tag);
335     print_text(h, title);
336     print_stagq(h, tt);

338     PAIR_CLASS_INIT(&tag[0], "head-vol");
339     PAIR_INIT(&tag[1], ATTR_ALIGN, "center");
340     print_otag(h, TAG_TD, 2, tag);
341     print_text(h, b);

```

```

342     print_stagq(h, tt);

344     PAIR_CLASS_INIT(&tag[0], "head-rtitle");
345     PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
346     print_otag(h, TAG_TD, 2, tag);
347     print_text(h, title);
348     print_tagq(h, t);
349 }

352 /* ARGSUSED */
353 static void
354 man_root_post(MAN_ARGS)
355 {
356     struct htmlpair tag[3];
357     struct tag      *t, *tt;

359     PAIR_SUMMARY_INIT(&tag[0], "Document Footer");
360     PAIR_CLASS_INIT(&tag[1], "foot");
361     PAIR_INIT(&tag[2], ATTR_WIDTH, "100%");
362     t = print_otag(h, TAG_TABLE, 3, tag);
363     PAIR_INIT(&tag[0], ATTR_WIDTH, "50%");
364     print_otag(h, TAG_COL, 1, tag);
365     print_otag(h, TAG_COL, 1, tag);

367     tt = print_otag(h, TAG_TR, 0, NULL);

369     PAIR_CLASS_INIT(&tag[0], "foot-date");
370     print_otag(h, TAG_TD, 1, tag);

372     assert(man->date);
373     print_text(h, man->date);
366     assert(m->date);
367     print_text(h, m->date);
374     print_stagq(h, tt);

376     PAIR_CLASS_INIT(&tag[0], "foot-os");
377     PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
378     print_otag(h, TAG_TD, 2, tag);

380     if (man->source)
381         print_text(h, man->source);
374     if (m->source)
375         print_text(h, m->source);
382     print_tagq(h, t);
383 }
    unchanged_portion_omitted_

431 /* ARGSUSED */
432 static int
433 man_alt_pre(MAN_ARGS)
434 {
435     const struct man_node *nn;
436     int                    i, savelit;
437     enum htmltag          fp;
438     struct tag            *t;

440     if ((savelit = mh->fl & MANH_LITERAL))
441         print_otag(h, TAG_BR, 0, NULL);

443     mh->fl &= ~MANH_LITERAL;

445     for (i = 0, nn = n->child; nn; nn = nn->next, i++) {
446         t = NULL;
447         switch (n->tok) {
448             case (MAN_BI):

```

```

449         fp = i % 2 ? TAG_I : TAG_B;
450         break;
451     case (MAN_IB):
452         fp = i % 2 ? TAG_B : TAG_I;
453         break;
454     case (MAN_RI):
455         fp = i % 2 ? TAG_I : TAG_MAX;
456         break;
457     case (MAN_IR):
458         fp = i % 2 ? TAG_MAX : TAG_I;
459         break;
460     case (MAN_BR):
461         fp = i % 2 ? TAG_MAX : TAG_B;
462         break;
463     case (MAN_RB):
464         fp = i % 2 ? TAG_B : TAG_MAX;
465         break;
466     default:
467         abort();
468         /* NOTREACHED */
469     }
471     if (i)
472         h->flags |= HTML_NOSPACE;
474     if (TAG_MAX != fp)
475         t = print_otag(h, fp, 0, NULL);
477     print_man_node(man, nn, mh, h);
478     print_man_node(m, nn, mh, h);
479     if (t)
480         print_tagq(h, t);
481 }
483     if (savelit)
484         mh->fl |= MANH_LITERAL;
486     return(0);
487 }

```

unchanged portion omitted

```

531 /* ARGSUSED */
532 static int
533 man_IP_pre(MAN_ARGS)
534 {
535     const struct man_node *nn;
537     if (MAN_BODY == n->type) {
538         print_otag(h, TAG_DD, 0, NULL);
539         return(1);
540     } else if (MAN_HEAD != n->type) {
541         print_otag(h, TAG_DL, 0, NULL);
542         return(1);
543     }
545     /* FIXME: width specification. */
547     print_otag(h, TAG_DT, 0, NULL);
549     /* For IP, only print the first header element. */
551     if (MAN_IP == n->tok && n->child)
552         print_man_node(man, n->child, mh, h);
553     print_man_node(m, n->child, mh, h);

```

```

554     /* For TP, only print next-line header elements. */
556     if (MAN_TP == n->tok)
557         for (nn = n->child; nn; nn = nn->next)
558             if (nn->line > n->line)
559                 print_man_node(man, nn, mh, h);
560                 print_man_node(m, nn, mh, h);
561     return(0);
562 }

```

unchanged portion omitted

```

642 /* ARGSUSED */
643 static int
644 man_literal_pre(MAN_ARGS)
645 {
647     if (MAN_fi == n->tok || MAN_EE == n->tok) {
648         if (MAN_nf != n->tok) {
649             print_otag(h, TAG_BR, 0, NULL);
650             mh->fl &= ~MANH_LITERAL;
651         } else
652             mh->fl |= MANH_LITERAL;
653     }
654     return(0);
655 }

```

unchanged portion omitted

```

673 /* ARGSUSED */
674 static int
675 man_RS_pre(MAN_ARGS)
676 {
677     struct htmlpair tag;
678     struct roffsu su;
680     if (MAN_HEAD == n->type)
681         return(0);
682     else if (MAN_BODY == n->type)
683         return(1);
685     SCALE_HS_INIT(&su, INDENT);
686     if (n->head->child)
687         a2width(n->head->child, &su);
689     bufinit(h);
690     bufcat_su(h, "margin-left", &su);
691     PAIR_STYLE_INIT(&tag, h);
692     print_otag(h, TAG_DIV, 1, &tag);
693     return(1);
694 }
696 /* ARGSUSED */
697 static int
698 man_UR_pre(MAN_ARGS)
699 {
700     struct htmlpair tag[2];
702     n = n->child;
703     assert(MAN_HEAD == n->type);
704     if (n->nchild) {
705         assert(MAN_TEXT == n->child->type);
706         PAIR_CLASS_INIT(&tag[0], "link-ext");
707         PAIR_HREF_INIT(&tag[1], n->child->string);
708         print_otag(h, TAG_A, 2, tag);
709     }

```

```
711     assert(MAN_BODY == n->next->type);
712     if (n->next->nchild)
713         n = n->next;
715     print_man_nodelist(man, n->child, mh, h);
717     return(0);
718 }
```

unchanged\_portion\_omitted

```

*****
11898 Fri Jul 25 15:00:12 2014
new/usr/src/cmd/mandoc/man_macro.c
Update to 1.12.3.
*****
1 /* $Id: man_macro.c,v 1.79 2013/12/25 00:50:05 schwarze Exp $ */
1 /* $Id: man_macro.c,v 1.71 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
5  * Copyright (c) 2013 Franco Fichtner <franco@lastsummer.de>
6  *
7  * Permission to use, copy, modify, and distribute this software for any
8  * purpose with or without fee is hereby granted, provided that the above
9  * copyright notice and this permission notice appear in all copies.
10 *
11 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
12 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
13 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
14 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
15 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
16 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
17 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
18 */
19 #ifdef HAVE_CONFIG_H
20 #include "config.h"
21 #endif
22
23 #include <assert.h>
24 #include <ctype.h>
25 #include <stdlib.h>
26 #include <string.h>
27
28 #include "man.h"
29 #include "mandoc.h"
30 #include "libmandoc.h"
31 #include "libman.h"
32
33 enum      rew {
34     REW_REWIND,
35     REW_NOHALT,
36     REW_HALT
37 };
38
39 static int      blk_close(MACRO_PROT_ARGS);
40 static int      blk_exp(MACRO_PROT_ARGS);
41 static int      blk_imp(MACRO_PROT_ARGS);
42 static int      in_line_eoln(MACRO_PROT_ARGS);
43 static int      man_args(struct man *, int,
44                          int *, char *, char **);
45
46 static int      rew_scope(enum man_type,
47                          struct man *, enum mant);
48 static enum rew rew_dohalt(enum mant, enum man_type,
49                          const struct man_node *);
50 static enum rew rew_block(enum mant, enum man_type,
51                          const struct man_node *);
52 static void     rew_warn(struct man *,
53                          struct man_node *, enum mandocerr);
54
55 const struct man_macro __man_macros[MAN_MAX] = {
56     { in_line_eoln, MAN_NSSCOPED }, /* br */
57     { in_line_eoln, MAN_BSCOPE }, /* TH */
58     { blk_imp, MAN_BSCOPE | MAN_SCOPED }, /* SH */
59     { blk_imp, MAN_BSCOPE | MAN_SCOPED }, /* SS */
60     { blk_imp, MAN_BSCOPE | MAN_SCOPED | MAN_FSCOPED }, /* TP */

```

```

61     { blk_imp, MAN_BSCOPE }, /* LP */
62     { blk_imp, MAN_BSCOPE }, /* PP */
63     { blk_imp, MAN_BSCOPE }, /* P */
64     { blk_imp, MAN_BSCOPE }, /* IP */
65     { blk_imp, MAN_BSCOPE }, /* HP */
66     { in_line_eoln, MAN_SCOPED }, /* SM */
67     { in_line_eoln, MAN_SCOPED }, /* SB */
68     { in_line_eoln, 0 }, /* BI */
69     { in_line_eoln, 0 }, /* IB */
70     { in_line_eoln, 0 }, /* BR */
71     { in_line_eoln, 0 }, /* RB */
72     { in_line_eoln, MAN_SCOPED }, /* R */
73     { in_line_eoln, MAN_SCOPED }, /* B */
74     { in_line_eoln, MAN_SCOPED }, /* I */
75     { in_line_eoln, 0 }, /* IR */
76     { in_line_eoln, 0 }, /* RI */
77     { in_line_eoln, MAN_NSSCOPED }, /* na */
78     { in_line_eoln, MAN_NSSCOPED }, /* sp */
79     { in_line_eoln, MAN_BSCOPE }, /* nf */
80     { in_line_eoln, MAN_BSCOPE }, /* fi */
81     { blk_close, 0 }, /* RE */
82     { blk_exp, MAN_BSCOPE | MAN_EXPLICIT }, /* RS */
83     { blk_exp, MAN_EXPLICIT }, /* RS */
84     { in_line_eoln, 0 }, /* DT */
85     { in_line_eoln, 0 }, /* UC */
86     { in_line_eoln, 0 }, /* PD */
87     { in_line_eoln, 0 }, /* AT */
88     { in_line_eoln, 0 }, /* in */
89     { in_line_eoln, 0 }, /* ft */
90     { in_line_eoln, MAN_BSCOPE }, /* EX */
91     { in_line_eoln, MAN_BSCOPE }, /* EE */
92     { blk_exp, MAN_BSCOPE | MAN_EXPLICIT }, /* UR */
93     { blk_close, 0 }, /* UE */
94 };
95
96 const struct man_macro * const man_macros = __man_macros;
97
98
99 /*
100  * Warn when "n" is an explicit non-roff macro.
101  */
102 static void
103 rew_warn(struct man *man, struct man_node *n, enum mandocerr er)
104 {
105     if (er == MANDOCERR_MAX || MAN_BLOCK != n->type)
106         return;
107     if (MAN_VALID & n->flags)
108         return;
109     if ( ! (MAN_EXPLICIT & man_macros[n->tok].flags))
110         return;
111
112     assert(er < MANDOCERR_FATAL);
113     man_nmsg(man, n, er);
114     man_nmsg(m, n, er);
115 }
116
117
118 /*
119  * Rewind scope. If a code "er" != MANDOCERR_MAX has been provided, it
120  * will be used if an explicit block scope is being closed out.
121  */
122 int
123 man_unscope(struct man *man, const struct man_node *to,

```



```

117 man_unscope(struct man *m, const struct man_node *to,
124             enum mandocerr er)
125 {
126     struct man_node *n;
127
128     assert(to);
129
130     man->next = MAN_NEXT_SIBLING;
131     m->next = MAN_NEXT_SIBLING;
132
133     /* LINTED */
134     while (man->last != to) {
135         while (m->last != to) {
136             /*
137              * Save the parent here, because we may delete the
138              * man->last node in the post-validation phase and reset
139              * it to man->last->parent, causing a step in the closing
140              * m->last node in the post-validation phase and reset
141              * it to m->last->parent, causing a step in the closing
142              * out to be lost.
143              */
144             n = man->last->parent;
145             rew_warn(man, man->last, er);
146             if (! man_valid_post(man))
147                 n = m->last->parent;
148             rew_warn(m, m->last, er);
149             if (! man_valid_post(m))
150                 return(0);
151             man->last = n;
152             assert(man->last);
153             m->last = n;
154             assert(m->last);
155         }
156     }
157
158     rew_warn(man, man->last, er);
159     if (! man_valid_post(man))
160         rew_warn(m, m->last, er);
161     if (! man_valid_post(m))
162         return(0);
163
164     return(1);
165 }
166
167 unchanged_portion_omitted
168
169 /*
170 * There are three scope levels: scoped to the root (all), scoped to the
171 * section (all less sections), and scoped to subsections (all less
172 * sections and subsections).
173 */
174 static enum rew
175 rew_dohalt(enum mant tok, enum man_type type, const struct man_node *n)
176 {
177     enum rew c;
178
179     /* We cannot progress beyond the root ever. */
180     if (MAN_ROOT == n->type)
181         return(REW_HALT);
182
183     assert(n->parent);
184
185     /* Normal nodes shouldn't go to the level of the root. */
186     if (MAN_ROOT == n->parent->type)
187         return(REW_REWIND);
188
189     /* Already-validated nodes should be closed out. */

```

```

188     if (MAN_VALID & n->flags)
189         return(REW_NOHALT);
190
191     /* First: rewind to ourselves. */
192     if (type == n->type && tok == n->tok) {
193         if (MAN_EXPLICIT & man_macros[n->tok].flags)
194             return(REW_HALT);
195         else
196             if (type == n->type && tok == n->tok)
197                 return(REW_REWIND);
198     }
199
200     /*
201     * Next follow the implicit scope-smashings as defined by man.7:
202     * section, sub-section, etc.
203     */
204     switch (tok) {
205     case (MAN_SH):
206         break;
207     case (MAN_SS):
208         /* Rewind to a section, if a block. */
209         if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
210             return(c);
211         break;
212     case (MAN_RS):
213         /* Preserve empty paragraphs before RS. */
214         if (0 == n->nchild && (MAN_P == n->tok ||
215             MAN_PP == n->tok || MAN_LP == n->tok))
216             return(REW_HALT);
217         /* Rewind to a subsection, if a block. */
218         if (REW_NOHALT != (c = rew_block(MAN_SS, type, n)))
219             return(c);
220         /* Rewind to a section, if a block. */
221         if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
222             return(c);
223         break;
224     default:
225         /* Rewind to an offsetter, if a block. */
226         if (REW_NOHALT != (c = rew_block(MAN_RS, type, n)))
227             return(c);
228         /* Rewind to a subsection, if a block. */
229         if (REW_NOHALT != (c = rew_block(MAN_SS, type, n)))
230             return(c);
231         /* Rewind to a section, if a block. */
232         if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
233             return(c);
234         break;
235     }
236
237     return(REW_NOHALT);
238 }
239
240
241 /*
242 * Rewinding entails ascending the parse tree until a coherent point,
243 * for example, the 'SH' macro will close out any intervening 'SS'
244 * scopes. When a scope is closed, it must be validated and actioned.
245 */
246 static int
247 rew_scope(enum man_type type, struct man *man, enum mant tok)
248 rew_scope(enum man_type type, struct man *m, enum mant tok)
249 {
250     struct man_node *n;
251     enum rew c;

```

```

252      /* LINTED */
253      for (n = man->last; n; n = n->parent) {
254      for (n = m->last; n; n = n->parent) {
255          /*
256           * Whether we should stop immediately (REW_HALT), stop
257           * and rewind until this point (REW_REWIND), or keep
258           * rewinding (REW_NOHALT).
259           */
260          c = rew_dohalt(tok, type, n);
261          if (REW_HALT == c)
262              return(1);
263          if (REW_REWIND == c)
264              break;
265      }
266
267      /*
268       * Rewind until the current point. Warn if we're a roff
269       * instruction that's mowing over explicit scopes.
270       */
271      assert(n);
272
273      return(man_unscope(man, n, MANDOCERR_MAX));
274      return(man_unscope(m, n, MANDOCERR_MAX));
275  }
276
277  /*
278   * Close out a generic explicit macro.
279   */
280  /* ARGSUSED */
281  int
282  blk_close(MACRO_PROT_ARGS)
283  {
284      enum mant          ntok;
285      const struct man_node *nn;
286
287      switch (tok) {
288      case (MAN_RE):
289          ntok = MAN_RS;
290          break;
291      case (MAN_UE):
292          ntok = MAN_UR;
293          break;
294      default:
295          abort();
296          /* NOTREACHED */
297      }
298
299      for (nn = man->last->parent; nn; nn = nn->parent)
300          if (ntok == nn->tok && MAN_BLOCK == nn->type)
301              for (nn = m->last->parent; nn; nn = nn->parent)
302                  if (ntok == nn->tok)
303                      break;
304
305      if (NULL == nn) {
306          man_pmsg(man, line, ppos, MANDOCERR_NOSCOPE);
307          if (!rew_scope(MAN_BLOCK, man, MAN_PP))
308              if (NULL == nn)
309                  man_pmsg(m, line, ppos, MANDOCERR_NOSCOPE);
310
311          if (!rew_scope(MAN_BODY, m, ntok))
312              return(0);
313      } else
314          man_unscope(man, nn, MANDOCERR_MAX);
315      if (!rew_scope(MAN_BLOCK, m, ntok))
316          return(0);
317  }

```

```

309      return(1);
310  }
311
312  /* ARGSUSED */
313  int
314  blk_exp(MACRO_PROT_ARGS)
315  {
316      struct man_node *n;
317      int          la;
318      char         *p;
319
320      /* Close out prior implicit scopes. */
321      /*
322       * Close out prior scopes. "Regular" explicit macros cannot be
323       * nested, but we allow roff macros to be placed just about
324       * anywhere.
325       */
326      if (!rew_scope(MAN_BLOCK, man, tok))
327          if (!man_block_alloc(m, line, ppos, tok))
328              return(0);
329
330      if (!man_block_alloc(man, line, ppos, tok))
331          if (!man_head_alloc(m, line, ppos, tok))
332              return(0);
333      if (!man_head_alloc(man, line, ppos, tok))
334          return(0);
335
336      for (;;) {
337          la = *pos;
338          if (!man_args(man, line, pos, buf, &p))
339              if (!man_args(m, line, pos, buf, &p))
340                  break;
341          if (!man_word_alloc(man, line, la, p))
342              if (!man_word_alloc(m, line, la, p))
343                  return(0);
344      }
345
346      assert(man);
347      assert(m);
348      assert(tok != MAN_MAX);
349
350      for (n = man->last; n; n = n->parent) {
351          if (n->tok != tok)
352              continue;
353          assert(MAN_HEAD == n->type);
354          man_unscope(man, n, MANDOCERR_MAX);
355          break;
356      }
357
358      return(man_body_alloc(man, line, ppos, tok));
359      if (!rew_scope(MAN_HEAD, m, tok))
360          return(0);
361      return(man_body_alloc(m, line, ppos, tok));
362  }
363
364  /*
365   * Parse an implicit-block macro. These contain a MAN_HEAD and a
366   * MAN_BODY contained within a MAN_BLOCK. Rules for closing out other
367   * scopes, such as 'SH' closing out an 'SS', are defined in the rew
368   * routines.
369   */

```

```

361 /* ARGSUSED */
362 int
363 blk_imp(MACRO_PROT_ARGS)
364 {
365     int          la;
366     char         *p;
367     struct man_node *n;
368
369     /* Close out prior scopes. */
370
371     if ( ! rew_scope(MAN_BODY, man, tok))
372     if ( ! rew_scope(MAN_BODY, m, tok))
373         return(0);
374     if ( ! rew_scope(MAN_BLOCK, man, tok))
375     if ( ! rew_scope(MAN_BLOCK, m, tok))
376         return(0);
377
378     /* Allocate new block & head scope. */
379
380     if ( ! man_block_alloc(man, line, ppos, tok))
381     if ( ! man_block_alloc(m, line, ppos, tok))
382         return(0);
383     if ( ! man_head_alloc(man, line, ppos, tok))
384     if ( ! man_head_alloc(m, line, ppos, tok))
385         return(0);
386
387     n = man->last;
388     n = m->last;
389
390     /* Add line arguments. */
391
392     for (;;) {
393         la = *pos;
394         if ( ! man_args(man, line, pos, buf, &p))
395         if ( ! man_args(m, line, pos, buf, &p))
396             break;
397         if ( ! man_word_alloc(man, line, la, p))
398         if ( ! man_word_alloc(m, line, la, p))
399             return(0);
400     }
401
402     /* Close out head and open body (unless MAN_SCOPE). */
403
404     if (MAN_SCOPED & man_macros[tok].flags) {
405         /* If we're forcing scope ('TP'), keep it open. */
406         if (MAN_FSCOPED & man_macros[tok].flags) {
407             man->flags |= MAN_BLINE;
408             m->flags |= MAN_BLINE;
409             return(1);
410         } else if (n == man->last) {
411             man->flags |= MAN_BLINE;
412         } else if (n == m->last) {
413             m->flags |= MAN_BLINE;
414             return(1);
415         }
416     }
417
418     if ( ! rew_scope(MAN_HEAD, man, tok))
419     if ( ! rew_scope(MAN_HEAD, m, tok))
420         return(0);
421     return(man_body_alloc(man, line, ppos, tok));
422     return(man_body_alloc(m, line, ppos, tok));
423 }
424
425 414 /* ARGSUSED */

```

```

426 int
427 in_line_eoln(MACRO_PROT_ARGS)
428 {
429     int          la;
430     char         *p;
431     struct man_node *n;
432
433     if ( ! man_elem_alloc(man, line, ppos, tok))
434     if ( ! man_elem_alloc(m, line, ppos, tok))
435         return(0);
436
437     n = man->last;
438     n = m->last;
439
440     for (;;) {
441         la = *pos;
442         if ( ! man_args(man, line, pos, buf, &p))
443         if ( ! man_args(m, line, pos, buf, &p))
444             break;
445         if ( ! man_word_alloc(man, line, la, p))
446         if ( ! man_word_alloc(m, line, la, p))
447             return(0);
448     }
449
450     /*
451     * Append MAN_EOS in case the last snipped argument
452     * ends with a dot, e.g. '.IR syslog (3).'
453     */
454
455     if (n != man->last &&
456         mandoc_eos(man->last->string, strlen(man->last->string), 0))
457         man->last->flags |= MAN_EOS;
458
459     /*
460     * If no arguments are specified and this is MAN_SCOPED (i.e.,
461     * next-line scoped), then set our mode to indicate that we're
462     * waiting for terms to load into our context.
463     */
464
465     if (n == man->last && MAN_SCOPED & man_macros[tok].flags) {
466         if (n == m->last && MAN_SCOPED & man_macros[tok].flags) {
467             assert( ! (MAN_NSOPED & man_macros[tok].flags));
468             man->flags |= MAN_ELINE;
469             m->flags |= MAN_ELINE;
470             return(1);
471         }
472     }
473
474     /* Set ignorable context, if applicable. */
475
476     if (MAN_NSOPED & man_macros[tok].flags) {
477         assert( ! (MAN_SCOPED & man_macros[tok].flags));
478         man->flags |= MAN_ILINE;
479         m->flags |= MAN_ILINE;
480     }
481
482     assert(MAN_ROOT != man->last->type);
483     man->next = MAN_NEXT_SIBLING;
484     assert(MAN_ROOT != m->last->type);
485     m->next = MAN_NEXT_SIBLING;
486
487     /*
488     * Rewind our element scope. Note that when TH is pruned, we'll
489     * be back at the root, so make sure that we don't clobber as
490     * its sibling.
491     */
492 }

```

```
472     for ( ; man->last; man->last = man->last->parent) {
473         if (man->last == n)
441     for ( ; m->last; m->last = m->last->parent) {
442         if (m->last == n)
474             break;
475         if (man->last->type == MAN_ROOT)
444         if (m->last->type == MAN_ROOT)
476             break;
477         if ( ! man_valid_post(man))
446         if ( ! man_valid_post(m))
478             return(0);
479     }

481     assert(man->last);
450     assert(m->last);

483     /*
484     * Same here regarding whether we're back at the root.
485     */

487     if (man->last->type != MAN_ROOT && ! man_valid_post(man))
456     if (m->last->type != MAN_ROOT && ! man_valid_post(m))
488         return(0);

490     return(1);
491 }

494 int
495 man_macroend(struct man *man)
464 man_macroend(struct man *m)
496 {

498     return(man_unscope(man, man->first, MANDOCERR_SCOPEEXIT));
467     return(man_unscope(m, m->first, MANDOCERR_SCOPEEXIT));
499 }

501 static int
502 man_args(struct man *man, int line, int *pos, char *buf, char **v)
471 man_args(struct man *m, int line, int *pos, char *buf, char **v)
503 {
504     char    *start;

506     assert(*pos);
507     *v = start = buf + *pos;
508     assert(' ' != *start);

510     if ('\0' == *start)
511         return(0);

513     *v = mandoc_getarg(man->parse, v, line, pos);
482     *v = mandoc_getarg(m->parse, v, line, pos);
514     return(1);
515 }

    unchanged_portion_omitted_
```

```

*****
23118 Fri Jul 25 15:00:12 2014
new/usr/src/cmd/mandoc/man_term.c
Update to 1.12.3.
*****
1 /* $Id: man_term.c,v 1.139 2013/12/22 23:34:13 schwarze Exp $ */
1 /* $Id: man_term.c,v 1.127 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29
30 #include "mandoc.h"
31 #include "out.h"
32 #include "man.h"
33 #include "term.h"
34 #include "main.h"
35
36 #define MAXMARGINS      64 /* maximum number of indented scopes */
37
38 /* FIXME: have PD set the default vspace width. */
39
40 struct mterm {
41     int      fl;
42 #define MANT_LITERAL    (1 << 0)
43     size_t   lmargin[MAXMARGINS]; /* margins (incl. visible page) *
44     int      lmargincur; /* index of current margin */
45     int      lmarginisz; /* actual number of nested margins */
46     size_t   offset; /* default offset to visible page */
47     int      pardist; /* vert. space before par., unit: [v] */
48 };
49
50 #define DECL_ARGS      struct term *p, \
51                       struct mterm *mt, \
52                       const struct man_node *n, \
53                       const struct man_meta *meta
54
55 struct termact {
56     int      (*pre)(DECL_ARGS);
57     void     (*post)(DECL_ARGS);

```

```

56     int      flags;
57 #define MAN_NOTEXT    (1 << 0) /* Never has text children. */
58 };
59
60 static int      a2width(const struct term *t, const char *c);
61 static size_t   a2height(const struct term *t, const char *c);
62
63 static void     print_man_nodelist(DECL_ARGS);
64 static void     print_man_node(DECL_ARGS);
65 static void     print_man_head(struct term *t, const void *c);
66 static void     print_man_foot(struct term *t, const void *c);
67 static void     print_bvspace(struct term *t, const void *c);
68
69     const struct man_node *n, int);
70     const struct man_node *);
71
72 static int      pre_B(DECL_ARGS);
73 static int      pre_HP(DECL_ARGS);
74 static int      pre_I(DECL_ARGS);
75 static int      pre_IP(DECL_ARGS);
76 static int      pre_OP(DECL_ARGS);
77 static int      pre_PD(DECL_ARGS);
78 static int      pre_PP(DECL_ARGS);
79 static int      pre_RS(DECL_ARGS);
80 static int      pre_SH(DECL_ARGS);
81 static int      pre_SS(DECL_ARGS);
82 static int      pre_TP(DECL_ARGS);
83 static int      pre_UR(DECL_ARGS);
84 static int      pre_alterate(DECL_ARGS);
85 static int      pre_ft(DECL_ARGS);
86 static int      pre_ign(DECL_ARGS);
87 static int      pre_in(DECL_ARGS);
88 static int      pre_literal(DECL_ARGS);
89 static int      pre_sp(DECL_ARGS);
90
91 static void     post_IP(DECL_ARGS);
92 static void     post_HP(DECL_ARGS);
93 static void     post_RS(DECL_ARGS);
94 static void     post_SH(DECL_ARGS);
95 static void     post_SS(DECL_ARGS);
96 static void     post_TP(DECL_ARGS);
97 static void     post_UR(DECL_ARGS);
98
99 static const struct termact termacts[MAN_MAX] = {
100     { pre_sp, NULL, MAN_NOTEXT }, /* br */
101     { NULL, NULL, 0 }, /* TH */
102     { pre_SH, post_SH, 0 }, /* SH */
103     { pre_SS, post_SS, 0 }, /* SS */
104     { pre_TP, post_TP, 0 }, /* TP */
105     { pre_PP, NULL, 0 }, /* LP */
106     { pre_PP, NULL, 0 }, /* PP */
107     { pre_PP, NULL, 0 }, /* P */
108     { pre_IP, post_IP, 0 }, /* IP */
109     { pre_HP, post_HP, 0 }, /* HP */
110     { NULL, NULL, 0 }, /* SM */
111     { pre_B, NULL, 0 }, /* SB */
112     { pre_alterate, NULL, 0 }, /* BI */
113     { pre_alterate, NULL, 0 }, /* IB */
114     { pre_alterate, NULL, 0 }, /* BR */
115     { pre_alterate, NULL, 0 }, /* RB */
116     { NULL, NULL, 0 }, /* R */
117     { pre_B, NULL, 0 }, /* B */
118     { pre_I, NULL, 0 }, /* I */
119     { pre_alterate, NULL, 0 }, /* IR */
120     { pre_alterate, NULL, 0 }, /* RI */
121     { pre_ign, NULL, MAN_NOTEXT }, /* na */
122     { pre_sp, NULL, MAN_NOTEXT }, /* sp */

```

```

121 { pre_literal, NULL, 0 }, /* nf */
122 { pre_literal, NULL, 0 }, /* fi */
123 { NULL, NULL, 0 }, /* RE */
124 { pre_RS, post_RS, 0 }, /* RS */
125 { pre_ign, NULL, 0 }, /* DT */
126 { pre_ign, NULL, 0 }, /* UC */
127 { pre_PD, NULL, MAN_NOTEXT }, /* PD */
125 { pre_ign, NULL, 0 }, /* PD */
128 { pre_ign, NULL, 0 }, /* AT */
129 { pre_in, NULL, MAN_NOTEXT }, /* in */
130 { pre_ft, NULL, MAN_NOTEXT }, /* ft */
131 { pre_OP, NULL, 0 }, /* OP */
132 { pre_literal, NULL, 0 }, /* EX */
133 { pre_literal, NULL, 0 }, /* EE */
134 { pre_UR, post_UR, 0 }, /* UR */
135 { NULL, NULL, 0 }, /* UE */
136 };

140 void
141 terminal_man(void *arg, const struct man *man)
142 {
143     struct term *p;
144     const struct man_node *n;
145     const struct man_meta *meta;
146     const struct man_meta *m;
147     struct mterm *mt;

148     p = (struct term *)arg;

149     if (0 == p->defindent)
150         p->defindent = 7;

151

152     p->overstep = 0;
153     p->maxrmargin = p->defrmargin;
154     p->tabwidth = term_len(p, 5);

155

156     if (NULL == p->syntab)
157         p->syntab = mchars_alloc();

158

159     n = man_node(man);
160     meta = man_meta(man);
161     m = man_meta(man);

162     term_begin(p, print_man_head, print_man_foot, meta);
163     term_begin(p, print_man_head, print_man_foot, m);
164     p->flags |= TERMP_NOSPACE;

165     memset(&mt, 0, sizeof(struct mterm));

166     mt.lmargin[mt.lmargincur] = term_len(p, p->defindent);
167     mt.offset = term_len(p, p->defindent);
168     mt.pardist = 1;

169     if (n->child)
170         print_man_nodelist(p, &mt, n->child, meta);
171         print_man_nodelist(p, &mt, n->child, m);

172     term_end(p);
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 /*
203 * Printing leading vertical space before a block.
204 * This is used for the paragraph macros.

```

```

205 * The rules are pretty simple, since there's very little nesting going
206 * on here. Basically, if we're the first within another block (SS/SH),
207 * then don't emit vertical space. If we are (RS), then do. If not the
208 * first, print it.
209 */
210 static void
211 print_bvspace(struct term *p, const struct man_node *n, int pardist)
212 {
213     int i;

214     term_newln(p);

215     if (n->body && n->body->child)
216         if (MAN_TBL == n->body->child->type)
217             return;

218     if (MAN_ROOT == n->parent->type || MAN_RS != n->parent->tok)
219         if (NULL == n->prev)
220             return;

221     for (i = 0; i < pardist; i++)
222         term_vspace(p);
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

285     }
286     assert(MAN_TEXT == n->type);
287     mt->pardist = atoi(n->string);
288     return(0);
289 }

291 /* ARGSUSED */
292 static int
293 pre_alternate(DECL_ARGS)
294 {
295     enum termfont      font[2];
296     const struct man_node *nn;
297     int                 savelit, i;

299     switch (n->tok) {
300     case (MAN_RB):
301         font[0] = TERMFONT_NONE;
302         font[1] = TERMFONT_BOLD;
303         break;
304     case (MAN_RI):
305         font[0] = TERMFONT_NONE;
306         font[1] = TERMFONT_UNDER;
307         break;
308     case (MAN_BR):
309         font[0] = TERMFONT_BOLD;
310         font[1] = TERMFONT_NONE;
311         break;
312     case (MAN_BI):
313         font[0] = TERMFONT_BOLD;
314         font[1] = TERMFONT_UNDER;
315         break;
316     case (MAN_IR):
317         font[0] = TERMFONT_UNDER;
318         font[1] = TERMFONT_NONE;
319         break;
320     case (MAN_IB):
321         font[0] = TERMFONT_UNDER;
322         font[1] = TERMFONT_BOLD;
323         break;
324     default:
325         abort();
326     }

328     savelit = MANT_LITERAL & mt->fl;
329     mt->fl &= ~MANT_LITERAL;

331     for (i = 0, nn = n->child; nn; nn = nn->next, i = 1 - i) {
332         term_fontrepl(p, font[i]);
333         if (savelit && NULL == nn->next)
334             mt->fl |= MANT_LITERAL;
335         print_man_node(p, mt, nn, meta);
336         print_man_node(p, mt, nn, m);
337         if (nn->next)
338             p->flags |= TERMP_NOSPACE;
339     }

340     return(0);
341 }

```

unchanged portion omitted

```

462 /* ARGSUSED */
463 static int
464 pre_sp(DECL_ARGS)
465 {
466     char *s;

```

```

467     size_t      i, len;
468     int         neg;

470     if ((NULL == n->prev && n->parent)) {
471         switch (n->parent->tok) {
472         case (MAN_SH):
473             /* FALLTHROUGH */
474         case (MAN_SS):
475             /* FALLTHROUGH */
476         case (MAN_PP):
477             /* FALLTHROUGH */
478         case (MAN_LP):
479             /* FALLTHROUGH */
480         case (MAN_P):
481             /* FALLTHROUGH */
482             if (MAN_SS == n->parent->tok)
483                 return(0);
484             default:
485                 break;
486             if (MAN_SH == n->parent->tok)
487                 return(0);
488         }
489     }

490     neg = 0;
491     switch (n->tok) {
492     case (MAN_br):
493         len = 0;
494         break;
495     default:
496         if (NULL == n->child) {
497             len = 1;
498             len = n->child ? a2height(p, n->child->string) : 1;
499             break;
500         }
501         s = n->child->string;
502         if ('-' == *s) {
503             neg = 1;
504             s++;
505         }
506         len = a2height(p, s);
507         break;
508     }

509     if (0 == len)
510         term_newln(p);
511     else if (neg)
512         p->skipvsp += len;
513     else
514         for (i = 0; i < len; i++)
515             term_vspace(p);

516     return(0);
517 }

519 /* ARGSUSED */
520 static int
521 pre_HP(DECL_ARGS)
522 {
523     size_t      len, one;
524     int         ival;
525     const struct man_node *nn;

527     switch (n->type) {
528     case (MAN_BLOCK):

```

```

529     print_bvspace(p, n, mt->pardist);
478     print_bvspace(p, n);
530     return(1);
531     case (MAN_BODY):
481         p->flags |= TERMP_NOBREAK;
482         p->flags |= TERMP_TWOSPACE;
532         break;
533     default:
534         return(0);
535 }

537 if ( ! (MANT_LITERAL & mt->fl) ) {
538     p->flags |= TERMP_NOBREAK;
539     p->trailspace = 2;
540 }

542 len = mt->lmargin[mt->lmargincur];
543 ival = -1;

545 /* Calculate offset. */

547 if (NULL != (nn = n->parent->head->child))
548     if ((ival = a2width(p, nn->string)) >= 0)
549         len = (size_t)ival;

551 one = term_len(p, 1);
552 if (len < one)
553     len = one;

555 p->offset = mt->offset;
556 p->rmargin = mt->offset + len;

558 if (ival >= 0)
559     mt->lmargin[mt->lmargincur] = (size_t)ival;

561 return(1);
562 }

565 /* ARGSUSED */
566 static void
567 post_HP(DECL_ARGS)
568 {
570     switch (n->type) {
571     case (MAN_BLOCK):
572         term_flushln(p);
573         break;
574     case (MAN_BODY):
575         term_newln(p);
576         term_flushln(p);
577         p->flags &= ~TERMP_NOBREAK;
578         p->trailspace = 0;
579         p->flags &= ~TERMP_TWOSPACE;
580         p->offset = mt->offset;
581         p->rmargin = p->maxrmargin;
582         break;
583     default:
584         break;
585 }
586 }

584 /* ARGSUSED */
585 static int
586 pre_PP(DECL_ARGS)

```

```

587 {
589     switch (n->type) {
590     case (MAN_BLOCK):
591         mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
592         print_bvspace(p, n, mt->pardist);
593         print_bvspace(p, n);
594         break;
595     default:
596         p->offset = mt->offset;
597         break;
598 }
599 }

603 /* ARGSUSED */
604 static int
605 pre_IP(DECL_ARGS)
606 {
607     const struct man_node *nn;
608     size_t len;
609     int savelit, ival;

611     switch (n->type) {
612     case (MAN_BODY):
613         p->flags |= TERMP_NOSPACE;
614         break;
615     case (MAN_HEAD):
616         p->flags |= TERMP_NOBREAK;
617         p->trailspace = 1;
618         break;
619     case (MAN_BLOCK):
620         print_bvspace(p, n, mt->pardist);
621         print_bvspace(p, n);
622         /* FALLTHROUGH */
623     default:
624         return(1);
625 }

626 len = mt->lmargin[mt->lmargincur];
627 ival = -1;

629 /* Calculate the offset from the optional second argument. */
630 if (NULL != (nn = n->parent->head->child))
631     if (NULL != (nn = nn->next))
632         if ((ival = a2width(p, nn->string)) >= 0)
633             len = (size_t)ival;

635     switch (n->type) {
636     case (MAN_HEAD):
637         /* Handle zero-width lengths. */
638         if (0 == len)
639             len = term_len(p, 1);

641         p->offset = mt->offset;
642         p->rmargin = mt->offset + len;
643         if (ival < 0)
644             break;

646         /* Set the saved left-margin. */
647         mt->lmargin[mt->lmargincur] = (size_t)ival;

649         savelit = MANT_LITERAL & mt->fl;
650         mt->fl &= ~MANT_LITERAL;

```



```

652         if (n->child)
653             print_man_node(p, mt, n->child, meta);
654             print_man_node(p, mt, n->child, m);
655
656         if (savelit)
657             mt->fl |= MANT_LITERAL;
658
659         return(0);
660     case (MAN_BODY):
661         p->offset = mt->offset + len;
662         p->rmargin = p->maxrmargin;
663         break;
664     default:
665         break;
666 }
667
668 return(1);
669 }
670
671 /* ARGSUSED */
672 static void
673 post_IP(DECL_ARGS)
674 {
675     switch (n->type) {
676     case (MAN_HEAD):
677         term_flushln(p);
678         p->flags &= ~TERMP_NOBREAK;
679         p->trailspace = 0;
680         p->rmargin = p->maxrmargin;
681         break;
682     case (MAN_BODY):
683         term_newln(p);
684         p->offset = mt->offset;
685         break;
686     default:
687         break;
688     }
689 }
690
691 /* ARGSUSED */
692 static int
693 pre_TP(DECL_ARGS)
694 {
695     const struct man_node *nn;
696     size_t len;
697     int savelit, ival;
698
699     switch (n->type) {
700     case (MAN_HEAD):
701         p->flags |= TERMP_NOBREAK;
702         p->trailspace = 1;
703         break;
704     case (MAN_BODY):
705         p->flags |= TERMP_NOSPACE;
706         break;
707     case (MAN_BLOCK):
708         print_bvspace(p, n, mt->pardist);
709         print_bvspace(p, n);
710         /* FALLTHROUGH */
711     default:
712         return(1);
713     }
714 }

```

```

716     len = (size_t)mt->lmargin[mt->lmargincur];
717     ival = -1;
718
719     /* Calculate offset. */
720
721     if (NULL != (nn = n->parent->head->child))
722         if (nn->string && nn->parent->line == nn->line)
723             if ((ival = a2width(p, nn->string)) >= 0)
724                 len = (size_t)ival;
725
726     switch (n->type) {
727     case (MAN_HEAD):
728         /* Handle zero-length properly. */
729         if (0 == len)
730             len = term_len(p, 1);
731
732         p->offset = mt->offset;
733         p->rmargin = mt->offset + len;
734
735         savelit = MANT_LITERAL & mt->fl;
736         mt->fl &= ~MANT_LITERAL;
737
738         /* Don't print same-line elements. */
739         for (nn = n->child; nn; nn = nn->next)
740             if (nn->line > n->line)
741                 print_man_node(p, mt, nn, meta);
742                 print_man_node(p, mt, nn, m);
743
744         if (savelit)
745             mt->fl |= MANT_LITERAL;
746             if (ival >= 0)
747                 mt->lmargin[mt->lmargincur] = (size_t)ival;
748
749     case (MAN_BODY):
750         p->offset = mt->offset + len;
751         p->rmargin = p->maxrmargin;
752         p->trailspace = 0;
753         p->flags &= ~TERMP_NOBREAK;
754         break;
755     default:
756         break;
757     }
758
759     return(1);
760 }
761
762 /* ARGSUSED */
763 static void
764 post_TP(DECL_ARGS)
765 {
766     switch (n->type) {
767     case (MAN_HEAD):
768         term_flushln(p);
769         p->flags &= ~TERMP_NOBREAK;
770         p->flags &= ~TERMP_TWOSPACE;
771         p->rmargin = p->maxrmargin;
772         break;
773     case (MAN_BODY):
774         term_newln(p);
775         p->offset = mt->offset;
776         break;
777     default:

```

```

777         break;
778     }
779 }

782 /* ARGSUSED */
783 static int
784 pre_SS(DECL_ARGS)
785 {
786     int i;

788     switch (n->type) {
789     case (MAN_BLOCK):
790         mt->fl &= ~MANT_LITERAL;
791         mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
792         mt->offset = term_len(p, p->defindent);
793         /* If following a prior empty 'SS', no vspace. */
794         if (n->prev && MAN_SS == n->prev->tok)
795             if (NULL == n->prev->body->child)
796                 break;
797         if (NULL == n->prev)
798             break;
799         for (i = 0; i < mt->pardist; i++)
800             term_vspace(p);
801         break;
802     case (MAN_HEAD):
803         term_fontrepl(p, TERMFONT_BOLD);
804         p->offset = term_len(p, 3);
805         p->offset = term_len(p, p->defindent/2);
806         break;
807     case (MAN_BODY):
808         p->offset = mt->offset;
809         break;
810     default:
811         break;
812     }

813     return(1);
814 }

```

unchanged\_portion\_omitted

```

835 /* ARGSUSED */
836 static int
837 pre_SH(DECL_ARGS)
838 {
839     int i;

841     switch (n->type) {
842     case (MAN_BLOCK):
843         mt->fl &= ~MANT_LITERAL;
844         mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
845         mt->offset = term_len(p, p->defindent);
846         /* If following a prior empty 'SH', no vspace. */
847         if (n->prev && MAN_SH == n->prev->tok)
848             if (NULL == n->prev->body->child)
849                 break;
850         /* If the first macro, no vspae. */
851         if (NULL == n->prev)
852             break;
853         for (i = 0; i < mt->pardist; i++)
854             term_vspace(p);
855         break;
856     case (MAN_HEAD):
857         term_fontrepl(p, TERMFONT_BOLD);
858         p->offset = 0;

```

```

859         break;
860     case (MAN_BODY):
861         p->offset = mt->offset;
862         break;
863     default:
864         break;
865     }

867     return(1);
868 }

```

unchanged\_portion\_omitted

```

952 /* ARGSUSED */
953 static int
954 pre_UR(DECL_ARGS)
955 {
957     return (MAN_HEAD != n->type);
958 }

960 /* ARGSUSED */
961 static void
962 post_UR(DECL_ARGS)
963 {
965     if (MAN_BLOCK != n->type)
966         return;

968     term_word(p, "<");
969     p->flags |= TERMP_NOSPACE;

971     if (NULL != n->child->child)
972         print_man_node(p, mt, n->child->child, meta);

974     p->flags |= TERMP_NOSPACE;
975     term_word(p, ">");
976 }

```

```

978 static void
979 print_man_node(DECL_ARGS)
980 {
981     size_t      rm, rmax;
982     int         c;

984     switch (n->type) {
985     case (MAN_TEXT):
986         /*
987          * If we have a blank line, output a vertical space.
988          * If we have a space as the first character, break
989          * before printing the line's data.
990          */
991         if ('\0' == *n->string) {
992             term_vspace(p);
993             return;
994         } else if (' ' == *n->string && MAN_LINE & n->flags)
995             term_newln(p);

997         term_word(p, n->string);
998         goto out;

994         /*
995          * If we're in a literal context, make sure that words
996          * together on the same line stay together. This is a
997          * POST-printing call, so we check the NEXT word. Since
998          * -man doesn't have nested macros, we don't need to be
999          * more specific than this.

```

```

920      */
921      if (MANT_LITERAL & mt->fl && ! (TERMP_NOBREAK & p->flags) &&
922          (NULL == n->next ||
923           n->next->line > n->line)) {
924          rm = p->rmargin;
925          rmax = p->maxrmargin;
926          p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
927          p->flags |= TERMP_NOSPACE;
928          term_flushln(p);
929          p->rmargin = rm;
930          p->maxrmargin = rmax;
931      }

933      if (MAN_EOS & n->flags)
934          p->flags |= TERMP_SENTENCE;
935      return;
1000 case (MAN_EQN):
1001     term_eqn(p, n->eqn);
1002     return;
1003 case (MAN_TBL):
1004     /*
1005      * Tables are preceded by a newline. Then process a
1006      * table line, which will cause line termination,
1007      */
1008     if (TBL_SPAN_FIRST & n->span->flags)
1009         term_newln(p);
1010     term_tbl(p, n->span);
1011     return;
1012 default:
1013     break;
1014 }

1016 if ( ! (MAN_NOTEXT & termacts[n->tok].flags))
1017     term_fontrepl(p, TERMFONT_NONE);

1019 c = 1;
1020 if (termacts[n->tok].pre)
1021     c = (*termacts[n->tok].pre)(p, mt, n, meta);
957     c = (*termacts[n->tok].pre)(p, mt, n, m);

1023 if (c && n->child)
1024     print_man_nodelist(p, mt, n->child, meta);
960     print_man_nodelist(p, mt, n->child, m);

1026 if (termacts[n->tok].post)
1027     (*termacts[n->tok].post)(p, mt, n, meta);
963     (*termacts[n->tok].post)(p, mt, n, m);
1028 if ( ! (MAN_NOTEXT & termacts[n->tok].flags))
1029     term_fontrepl(p, TERMFONT_NONE);

1031 out:
1032 /*
1033  * If we're in a literal context, make sure that words
1034  * together on the same line stay together. This is a
1035  * POST-printing call, so we check the NEXT word. Since
1036  * -man doesn't have nested macros, we don't need to be
1037  * more specific than this.
1038  */
1039 if (MANT_LITERAL & mt->fl && ! (TERMP_NOBREAK & p->flags) &&
1040     (NULL == n->next || n->next->line > n->line)) {
1041     rm = p->rmargin;
1042     rmax = p->maxrmargin;
1043     p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
1044     p->flags |= TERMP_NOSPACE;
1045     if (NULL != n->string && '\0' != *n->string)
1046         term_flushln(p);

```

```

1047     else
1048         term_newln(p);
1049     if (rm < rmax && n->parent->tok == MAN_HP) {
1050         p->offset = rm;
1051         p->rmargin = rmax;
1052     } else
1053         p->rmargin = rm;
1054     p->maxrmargin = rmax;
1055 }
1056 if (MAN_EOS & n->flags)
1057     p->flags |= TERMP_SENTENCE;
1058 }

1061 static void
1062 print_man_nodelist(DECL_ARGS)
1063 {

1065     print_man_node(p, mt, n, meta);
976     print_man_node(p, mt, n, m);
1066     if ( ! n->next)
1067         return;
1068     print_man_nodelist(p, mt, n->next, meta);
979     print_man_nodelist(p, mt, n->next, m);
1069 }

1072 static void
1073 print_man_foot(struct term *p, const void *arg)
1074 {
1075     char          title[BUFSIZ];
1076     size_t        datelen;
1077     const struct man_meta *meta;

1079     meta = (const struct man_meta *)arg;
1080     assert(meta->title);
1081     assert(meta->msec);
1082     assert(meta->date);

1084     term_fontrepl(p, TERMFONT_NONE);

1086     term_vspace(p);

1088     /*
1089      * Temporary, undocumented option to imitate mdoc(7) output.
1090      * In the bottom right corner, use the source instead of
1091      * the title.
1092      */

1094     if ( ! p->mdocstyle) {
1095         term_vspace(p);
1096         term_vspace(p);
1097         snprintf(title, BUFSIZ, "%s(%s)", meta->title, meta->msec);
1098     } else if (meta->source) {
1099         strlcpy(title, meta->source, BUFSIZ);
1100     } else {
1101         title[0] = '\0';
1102     }
1103     datelen = term_strlen(p, meta->date);

1105     /* Bottom left corner: manual source. */

1107     p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;
1108     p->trailspace = 1;
1109     p->offset = 0;
1110     p->rmargin = (p->maxrmargin - datelen + term_len(p, 1)) / 2;

```

```

1112     if (meta->source)
1113         term_word(p, meta->source);
1114     term_flushln(p);

1116     /* At the bottom in the middle: manual date. */

1118     p->flags |= TERMP_NOSPACE;
1119     p->offset = p->rmargin;
1120     p->rmargin = p->maxrmargin - term_strlen(p, title);
1121     if (p->offset + datelen >= p->rmargin)
1122         p->rmargin = p->offset + datelen;

1124     term_word(p, meta->date);
1125     term_flushln(p);

1127     /* Bottom right corner: manual title and section. */

1129     p->flags &= ~TERMP_NOBREAK;
1130     p->flags |= TERMP_NOSPACE;
1131     p->trailspace = 0;
1132     p->offset = p->rmargin;
1133     p->rmargin = p->maxrmargin;

1135     term_word(p, title);
1136     term_flushln(p);
1137 }

1140 static void
1141 print_man_head(struct term *p, const void *arg)
1142 {
1143     char          buf[BUFSIZ], title[BUFSIZ];
1144     size_t        buflen, titlen;
1145     const struct man_meta *meta;
1054     const struct man_meta *m;

1147     meta = (const struct man_meta *)arg;
1148     assert(meta->title);
1149     assert(meta->msec);
1056     m = (const struct man_meta *)arg;
1057     assert(m->title);
1058     assert(m->msec);

1151     if (meta->vol)
1152         strcpy(buf, meta->vol, BUFSIZ);
1060     if (m->vol)
1061         strcpy(buf, m->vol, BUFSIZ);
1153     else
1154         buf[0] = '\0';
1155     buflen = term_strlen(p, buf);

1157     /* Top left corner: manual title and section. */

1159     snprintf(title, BUFSIZ, "%s(%s)", meta->title, meta->msec);
1068     snprintf(title, BUFSIZ, "%s(%s)", m->title, m->msec);
1160     titlen = term_strlen(p, title);

1162     p->flags |= TERMP_NOBREAK | TERMP_NOSPACE;
1163     p->trailspace = 1;
1164     p->offset = 0;
1165     p->rmargin = 2 * (titlen+1) + buflen < p->maxrmargin ?
1166         (p->maxrmargin -
1167          term_strlen(p, buf) + term_len(p, 1)) / 2 :
1168         p->maxrmargin - buflen;

```

```

1170     term_word(p, title);
1171     term_flushln(p);

1173     /* At the top in the middle: manual volume. */

1175     p->flags |= TERMP_NOSPACE;
1176     p->offset = p->rmargin;
1177     p->rmargin = p->offset + buflen + titlen < p->maxrmargin ?
1178         p->maxrmargin - titlen : p->maxrmargin;

1180     term_word(p, buf);
1181     term_flushln(p);

1183     /* Top right corner: title and section, again. */

1185     p->flags &= ~TERMP_NOBREAK;
1186     p->trailspace = 0;
1187     if (p->rmargin + titlen <= p->maxrmargin) {
1188         p->flags |= TERMP_NOSPACE;
1189         p->offset = p->rmargin;
1190         p->rmargin = p->maxrmargin;
1191         term_word(p, title);
1192         term_flushln(p);
1193     }

1195     p->flags &= ~TERMP_NOSPACE;
1196     p->offset = 0;
1197     p->rmargin = p->maxrmargin;

1199     /*
1200     * Groff prints three blank lines before the content.
1201     * Do the same, except in the temporary, undocumented
1202     * mode imitating mdoc(7) output.
1203     */

1205     term_vspace(p);
1206     if (! p->mdocstyle) {
1207         term_vspace(p);
1208         term_vspace(p);
1209     }
1210 }

```

unchanged portion omitted

```

*****
12357 Fri Jul 25 15:00:13 2014
new/usr/src/cmd/mandoc/man_validate.c
Update to 1.12.3.
*****
1 /* $Id: man_validate.c,v 1.86 2013/10/17 20:54:58 schwarze Exp $ */
1 /* $Id: man_validate.c,v 1.80 2012/01/03 15:16:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif

22 #include <sys/types.h>

24 #include <assert.h>
25 #include <ctype.h>
26 #include <errno.h>
27 #include <limits.h>
28 #include <stdarg.h>
29 #include <stdlib.h>
30 #include <string.h>
31 #include <time.h>

33 #include "man.h"
34 #include "mandoc.h"
35 #include "libman.h"
36 #include "libmandoc.h"

38 #define CHKARGS struct man *man, struct man_node *n
38 #define CHKARGS struct man *m, struct man_node *n

40 typedef int (*v_check)(CHKARGS);

42 struct man_valid {
43     v_check *pres;
44     v_check *posts;
45 };

47 static int check_eq0(CHKARGS);
48 static int check_eq2(CHKARGS);
49 static int check_le1(CHKARGS);
50 static int check_ge2(CHKARGS);
51 static int check_le5(CHKARGS);
52 static int check_head1(CHKARGS);
53 static int check_par(CHKARGS);
54 static int check_part(CHKARGS);
55 static int check_root(CHKARGS);
56 static void check_text(CHKARGS);

58 static int post_AT(CHKARGS);

```

```

59 static int post_IP(CHKARGS);
60 static int post_vs(CHKARGS);
61 static int post_fi(CHKARGS);
62 static int post_ft(CHKARGS);
63 static int post_nf(CHKARGS);
64 static int post_sec(CHKARGS);
65 static int post_TH(CHKARGS);
66 static int post_UC(CHKARGS);
67 static int pre_sec(CHKARGS);

69 static v_check posts_at[] = { post_AT, NULL };
70 static v_check posts_br[] = { post_vs, check_eq0, NULL };
71 static v_check posts_eq0[] = { check_eq0, NULL };
72 static v_check posts_eq2[] = { check_eq2, NULL };
73 static v_check posts_fi[] = { check_eq0, post_fi, NULL };
74 static v_check posts_ft[] = { post_ft, NULL };
75 static v_check posts_ip[] = { post_IP, NULL };
76 static v_check posts_le1[] = { check_le1, NULL };
77 static v_check posts_nf[] = { check_eq0, post_nf, NULL };
78 static v_check posts_par[] = { check_par, NULL };
79 static v_check posts_part[] = { check_part, NULL };
80 static v_check posts_sec[] = { post_sec, NULL };
81 static v_check posts_sp[] = { post_vs, check_le1, NULL };
82 static v_check posts_th[] = { check_ge2, check_le5, post_TH, NULL };
83 static v_check posts_uc[] = { post_UC, NULL };
84 static v_check posts_ur[] = { check_head1, check_part, NULL };
85 static v_check pres_sec[] = { pre_sec, NULL };

87 static const struct man_valid man_valids[MAN_MAX] = {
88     { NULL, posts_br }, /* br */
89     { NULL, posts_th }, /* TH */
90     { pres_sec, posts_sec }, /* SH */
91     { pres_sec, posts_sec }, /* SS */
92     { NULL, NULL }, /* TP */
93     { NULL, posts_par }, /* LP */
94     { NULL, posts_par }, /* PP */
95     { NULL, posts_par }, /* P */
96     { NULL, posts_ip }, /* IP */
97     { NULL, NULL }, /* IP */
98     { NULL, NULL }, /* HP */
99     { NULL, NULL }, /* SM */
100    { NULL, NULL }, /* SB */
101    { NULL, NULL }, /* BI */
102    { NULL, NULL }, /* IB */
103    { NULL, NULL }, /* BR */
104    { NULL, NULL }, /* RB */
105    { NULL, NULL }, /* R */
106    { NULL, NULL }, /* B */
107    { NULL, NULL }, /* I */
108    { NULL, NULL }, /* IR */
109    { NULL, NULL }, /* RI */
110    { NULL, posts_eq0 }, /* na */
111    { NULL, posts_sp }, /* sp */
112    { NULL, posts_nf }, /* nf */
113    { NULL, posts_fi }, /* fi */
114    { NULL, NULL }, /* RE */
115    { NULL, posts_part }, /* RS */
116    { NULL, NULL }, /* DT */
117    { NULL, posts_uc }, /* UC */
118    { NULL, posts_le1 }, /* PD */
119    { NULL, NULL }, /* PD */
120    { NULL, posts_at }, /* AT */
121    { NULL, NULL }, /* in */
122    { NULL, posts_ft }, /* ft */
123    { NULL, posts_eq2 }, /* OP */
124    { NULL, posts_nf }, /* EX */

```

```

123     { NULL, posts_fi }, /* EE */
124     { NULL, posts_ur }, /* UR */
125     { NULL, NULL }, /* UE */
126 };

129 int
130 man_valid_pre(struct man *man, struct man_node *n)
131 man_valid_pre(struct man *m, struct man_node *n)
132 {
133     v_check      *cp;

134     switch (n->type) {
135     case (MAN_TEXT):
136         /* FALLTHROUGH */
137     case (MAN_ROOT):
138         /* FALLTHROUGH */
139     case (MAN_EQN):
140         /* FALLTHROUGH */
141     case (MAN_TBL):
142         return(1);
143     default:
144         break;
145     }

147     if (NULL == (cp = man_valids[n->tok].pres))
148         return(1);
149     for ( ; *cp; cp++)
150         if ( ! (*cp)(man, n) )
151             if ( ! (*cp)(m, n) )
152                 return(0);
153     return(1);
154 }

156 int
157 man_valid_post(struct man *man)
158 man_valid_post(struct man *m)
159 {
160     v_check      *cp;

161     if (MAN_VALID & man->last->flags)
162         if (MAN_VALID & m->last->flags)
163             return(1);
164     man->last->flags |= MAN_VALID;
165     m->last->flags |= MAN_VALID;

166     switch (man->last->type) {
167     case (MAN_TEXT):
168         check_text(man, man->last);
169         check_text(m, m->last);
170         return(1);
171     case (MAN_ROOT):
172         return(check_root(man, man->last));
173         return(check_root(m, m->last));
174     case (MAN_EQN):
175         /* FALLTHROUGH */
176     case (MAN_TBL):
177         return(1);
178     default:
179         break;
180     }

181     if (NULL == (cp = man_valids[man->last->tok].posts))
182     if (NULL == (cp = man_valids[m->last->tok].posts))

```

```

180         return(1);
181     for ( ; *cp; cp++)
182         if ( ! (*cp)(man, man->last))
183             if ( ! (*cp)(m, m->last))
184                 return(0);

185     return(1);
186 }

189 static int
190 check_root(CHKARGS)
191 {
192     if (MAN_BLINE & man->flags)
193         man_rmsg(man, n, MANDOCERR_SCOPEEXIT);
194     else if (MAN_ELINE & man->flags)
195         man_rmsg(man, n, MANDOCERR_SCOPEEXIT);
196     if (MAN_BLINE & m->flags)
197         man_rmsg(m, n, MANDOCERR_SCOPEEXIT);
198     else if (MAN_ELINE & m->flags)
199         man_rmsg(m, n, MANDOCERR_SCOPEEXIT);

200     man->flags &= ~MAN_BLINE;
201     man->flags &= ~MAN_ELINE;
202     m->flags &= ~MAN_BLINE;
203     m->flags &= ~MAN_ELINE;

204     if (NULL == man->first->child) {
205         man_rmsg(man, n, MANDOCERR_NODOCBODY);
206         if (NULL == m->first->child) {
207             man_rmsg(m, n, MANDOCERR_NODOCBODY);
208             return(0);
209         } else if (NULL == man->meta.title) {
210             man_rmsg(man, n, MANDOCERR_NOTITLE);
211         } else if (NULL == m->meta.title) {
212             man_rmsg(m, n, MANDOCERR_NOTITLE);

213             /*
214              * If a title hasn't been set, do so now (by
215              * implication, date and section also aren't set).
216              */

217             man->meta.title = mandoc_strdup("unknown");
218             man->meta.msec = mandoc_strdup("1");
219             man->meta.date = mandoc_normdate
220                 (man->parse, NULL, n->line, n->pos);
221             m->meta.title = mandoc_strdup("unknown");
222             m->meta.msec = mandoc_strdup("1");
223             m->meta.date = mandoc_normdate
224                 (m->parse, NULL, n->line, n->pos);
225         }

226     return(1);
227 }

228 static void
229 check_text(CHKARGS)
230 {
231     char          *cp, *p;

232     if (MAN_LITERAL & man->flags)
233         if (MAN_LITERAL & m->flags)
234             return;

235     cp = n->string;

```

```

230     for (p = cp; NULL != (p = strchr(p, '\t')); p++)
231         man_pmsg(man, n->line, (int)(p - cp), MANDOCERR_BADTAB);
232     man_pmsg(m, n->line, (int)(p - cp), MANDOCERR_BADTAB);
233 }
234 #define INEQ_DEFINE(x, ineq, name) \
235 static int \
236 check_##name(CHKARGS) \
237 { \
238     if (n->nchild ineq (x)) \
239         return(1); \
240     mandoc_vmsg(MANDOCERR_ARGCOUNT, man->parse, n->line, n->pos, \
241     mandoc_vmsg(MANDOCERR_ARGCOUNT, m->parse, n->line, n->pos, \
242     "line arguments %s %d (have %d)", \
243     #ineq, (x), n->nchild); \
244     return(1); \
245 }
246 INEQ_DEFINE(0, ==, eq0)
247 INEQ_DEFINE(2, ==, eq2)
248 INEQ_DEFINE(1, <=, le1)
249 INEQ_DEFINE(2, >=, ge2)
250 INEQ_DEFINE(5, <=, le5)
252 static int
253 check_head1(CHKARGS)
254 {
255     if (MAN_HEAD == n->type && 1 != n->nchild)
256         mandoc_vmsg(MANDOCERR_ARGCOUNT, man->parse, n->line,
257         n->pos, "line arguments eq 1 (have %d)", n->nchild);
260     return(1);
261 }
263 static int
264 post_ft(CHKARGS)
265 {
266     char *cp;
267     int ok;
269     if (0 == n->nchild)
270         return(1);
272     ok = 0;
273     cp = n->child->string;
274     switch (*cp) {
275     case ('1'):
276         /* FALLTHROUGH */
277     case ('2'):
278         /* FALLTHROUGH */
279     case ('3'):
280         /* FALLTHROUGH */
281     case ('4'):
282         /* FALLTHROUGH */
283     case ('I'):
284         /* FALLTHROUGH */
285     case ('P'):
286         /* FALLTHROUGH */
287     case ('R'):
288         if ('\0' == cp[1])
289             ok = 1;
290         break;
291     case ('B'):
292         if ('\0' == cp[1] || ('I' == cp[1] && '\0' == cp[2]))
293             ok = 1;

```

```

294         break;
295     case ('C'):
296         if ('W' == cp[1] && '\0' == cp[2])
297             ok = 1;
298         break;
299     default:
300         break;
301     }
303     if (0 == ok) {
304         mandoc_vmsg
305             (MANDOCERR_BADFONT, man->parse,
306             (MANDOCERR_BADFONT, m->parse,
307             n->line, n->pos, "%s", cp);
308         *cp = '\0';
310     if (1 < n->nchild)
311         mandoc_vmsg
312             (MANDOCERR_ARGCOUNT, man->parse, n->line,
313             (MANDOCERR_ARGCOUNT, m->parse, n->line,
314             n->pos, "want one child (have %d)",
315             n->nchild);
316     return(1);
317 }
319 static int
320 pre_sec(CHKARGS)
321 {
323     if (MAN_BLOCK == n->type)
324         man->flags &= ~MAN_LITERAL;
325     m->flags &= ~MAN_LITERAL;
326     return(1);
328 static int
329 post_sec(CHKARGS)
330 {
332     if (! (MAN_HEAD == n->type && 0 == n->nchild))
333         return(1);
335     man_nmsg(man, n, MANDOCERR_SYNTARGCOUNT);
336     man_nmsg(m, n, MANDOCERR_SYNTARGCOUNT);
337     return(0);
339 static int
340 check_part(CHKARGS)
341 {
343     if (MAN_BODY == n->type && 0 == n->nchild)
344         mandoc_msg(MANDOCERR_ARGCWARN, man->parse, n->line,
345         mandoc_msg(MANDOCERR_ARGCWARN, m->parse, n->line,
346         n->pos, "want children (have none)");
347     return(1);
348 }
351 static int
352 check_par(CHKARGS)
353 {

```

```

355     switch (n->type) {
356     case (MAN_BLOCK):
357         if (0 == n->body->nchild)
358             man_node_delete(man, n);
359             man_node_delete(m, n);
360         break;
361     case (MAN_BODY):
362         if (0 == n->nchild)
363             man_nmsg(man, n, MANDOCERR_IGNPAR);
364             man_nmsg(m, n, MANDOCERR_IGNPAR);
365         break;
366     case (MAN_HEAD):
367         if (n->nchild)
368             man_nmsg(man, n, MANDOCERR_ARGSLOST);
369             man_nmsg(m, n, MANDOCERR_ARGSLOST);
370         break;
371     default:
372         break;
373     }
374
375     return(1);
376 }
377
378 static int
379 post_IP(CHKARGS)
380 {
381     switch (n->type) {
382     case (MAN_BLOCK):
383         if (0 == n->head->nchild && 0 == n->body->nchild)
384             man_node_delete(man, n);
385             man_node_delete(m, n);
386         break;
387     case (MAN_BODY):
388         if (0 == n->parent->head->nchild && 0 == n->nchild)
389             man_nmsg(man, n, MANDOCERR_IGNPAR);
390             man_nmsg(m, n, MANDOCERR_IGNPAR);
391         break;
392     default:
393         break;
394     }
395     return(1);
396 }
397
398 static int
399 post_TH(CHKARGS)
400 {
401     const char *p;
402     int line, pos;
403
404     free(man->meta.title);
405     free(man->meta.vol);
406     free(man->meta.source);
407     free(man->meta.msec);
408     free(man->meta.date);
409     if (m->meta.title)
410         free(m->meta.title);
411     if (m->meta.vol)
412         free(m->meta.vol);
413     if (m->meta.source)
414         free(m->meta.source);
415     if (m->meta.msec)
416         free(m->meta.msec);
417     if (m->meta.date)
418         free(m->meta.date);
419
420     line = n->line;
421     pos = n->pos;

```

```

422     man->meta.title = man->meta.vol = man->meta.date =
423     man->meta.msec = man->meta.source = NULL;
424     m->meta.title = m->meta.vol = m->meta.date =
425     m->meta.msec = m->meta.source = NULL;
426
427     /* ->TITLE<- MSEC DATE SOURCE VOL */
428
429     n = n->child;
430     if (n && n->string) {
431         for (p = n->string; '\0' != *p; p++) {
432             /* Only warn about this once... */
433             if (isalpha((unsigned char)*p) &&
434                 ! isupper((unsigned char)*p)) {
435                 man_nmsg(man, n, MANDOCERR_UPPERCASE);
436                 man_nmsg(m, n, MANDOCERR_UPPERCASE);
437                 break;
438             }
439         }
440     }
441     man->meta.title = mandoc_strdup(n->string);
442     m->meta.title = mandoc_strdup(n->string);
443 } else
444     man->meta.title = mandoc_strdup("");
445     m->meta.title = mandoc_strdup("");
446
447     /* TITLE ->MSEC<- DATE SOURCE VOL */
448
449     if (n)
450         n = n->next;
451     if (n && n->string)
452         man->meta.msec = mandoc_strdup(n->string);
453         m->meta.msec = mandoc_strdup(n->string);
454     else
455         man->meta.msec = mandoc_strdup("");
456         m->meta.msec = mandoc_strdup("");
457
458     /* TITLE MSEC ->DATE<- SOURCE VOL */
459
460     if (n)
461         n = n->next;
462     if (n && n->string && '\0' != n->string[0]) {
463         pos = n->pos;
464         man->meta.date = mandoc_normdate
465             (man->parse, n->string, line, pos);
466         m->meta.date = mandoc_normdate
467             (m->parse, n->string, line, pos);
468     } else
469         man->meta.date = mandoc_strdup("");
470         m->meta.date = mandoc_strdup("");
471
472     /* TITLE MSEC DATE ->SOURCE<- VOL */
473
474     if (n && (n = n->next))
475         man->meta.source = mandoc_strdup(n->string);
476         m->meta.source = mandoc_strdup(n->string);
477
478     /* TITLE MSEC DATE SOURCE ->VOL<- */
479     /* If missing, use the default VOL name for MSEC. */
480
481     if (n && (n = n->next))
482         man->meta.vol = mandoc_strdup(n->string);
483     else if ('\0' != man->meta.msec[0] &&
484             (NULL != (p = mandoc_a2msec(man->meta.msec))))
485         man->meta.vol = mandoc_strdup(p);
486     m->meta.vol = mandoc_strdup(n->string);
487     else if ('\0' != m->meta.msec[0] &&
488             (NULL != (p = mandoc_a2msec(m->meta.msec))))

```



```

426     m->meta.vol = mandoc_strdup(p);

461     /*
462     * Remove the 'TH' node after we've processed it for our
463     * meta-data.
464     */
465     man_node_delete(man, man->last);
466     man_node_delete(m, m->last);
467     return(1);
468 }

469 static int
470 post_nf(CHKARGS)
471 {

473     if (MAN_LITERAL & man->flags)
474         man_rmsg(man, n, MANDOCERR_SCOPEREPP);
475     if (MAN_LITERAL & m->flags)
476         man_rmsg(m, n, MANDOCERR_SCOPEREPP);

477     man->flags |= MAN_LITERAL;
478     m->flags |= MAN_LITERAL;
479     return(1);
480 }

481 static int
482 post_fi(CHKARGS)
483 {

484     if ( ! (MAN_LITERAL & man->flags))
485         man_rmsg(man, n, MANDOCERR_WNOSCOPE);
486     if ( ! (MAN_LITERAL & m->flags))
487         man_rmsg(m, n, MANDOCERR_WNOSCOPE);

488     man->flags &= ~MAN_LITERAL;
489     m->flags &= ~MAN_LITERAL;
490     return(1);
491 }

492 static int
493 post_UC(CHKARGS)
494 {
495     static const char * const bsd_versions[] = {
496         "3rd Berkeley Distribution",
497         "4th Berkeley Distribution",
498         "4.2 Berkeley Distribution",
499         "4.3 Berkeley Distribution",
500         "4.4 Berkeley Distribution",
501     };

502     const char      *p, *s;

503     n = n->child;

504     if (NULL == n || MAN_TEXT != n->type)
505         p = bsd_versions[0];
506     else {
507         s = n->string;
508         if (0 == strcmp(s, "3"))
509             p = bsd_versions[0];
510         else if (0 == strcmp(s, "4"))
511             p = bsd_versions[1];
512         else if (0 == strcmp(s, "5"))
513             p = bsd_versions[2];
514         else if (0 == strcmp(s, "6"))
515             p = bsd_versions[3];
516     }

```

```

517         else if (0 == strcmp(s, "7"))
518             p = bsd_versions[4];
519         else
520             p = bsd_versions[0];
521     }

522     free(man->meta.source);
523     man->meta.source = mandoc_strdup(p);
524     if (m->meta.source)
525         free(m->meta.source);

526     m->meta.source = mandoc_strdup(p);
527     return(1);
528 }

529 static int
530 post_AT(CHKARGS)
531 {
532     static const char * const unix_versions[] = {
533         "7th Edition",
534         "System III",
535         "System V",
536         "System V Release 2",
537     };

538     const char      *p, *s;
539     struct man_node *nn;

540     n = n->child;

541     if (NULL == n || MAN_TEXT != n->type)
542         p = unix_versions[0];
543     else {
544         s = n->string;
545         if (0 == strcmp(s, "3"))
546             p = unix_versions[0];
547         else if (0 == strcmp(s, "4"))
548             p = unix_versions[1];
549         else if (0 == strcmp(s, "5")) {
550             nn = n->next;
551             if (nn && MAN_TEXT == nn->type && nn->string[0])
552                 p = unix_versions[3];
553             else
554                 p = unix_versions[2];
555         } else
556             p = unix_versions[0];
557     }

558     free(man->meta.source);
559     man->meta.source = mandoc_strdup(p);
560     if (m->meta.source)
561         free(m->meta.source);

562     m->meta.source = mandoc_strdup(p);
563     return(1);
564 }

565 static int
566 post_vs(CHKARGS)
567 {

568     if (NULL != n->prev)
569         return(1);

570     switch (n->parent->tok) {
571     case (MAN_SH):

```

```
576         /* FALLTHROUGH */
577     case (MAN_SS):
578         man_nmsg(man, n, MANDOCERR_IGNPAR);
579         /* FALLTHROUGH */
580     case (MAN_MAX):
581         /*
582          * Don't warn about this because it occurs in pod2man
583          * and would cause considerable (unfixable) warnage.
584          * Don't warn about this because it occurs in pod2man and would
585          * cause considerable (unfixable) warnage.
586          */
587         man_node_delete(man, n);
588         break;
589     default:
590         break;
591 }
592 if (NULL == n->prev && MAN_ROOT == n->parent->type)
593     man_node_delete(m, n);
594
595     return(1);
596 }
```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

new/usr/src/cmd/mandoc/mandoc.c

1

```
*****
12616 Fri Jul 25 15:00:13 2014
new/usr/src/cmd/mandoc/mandoc.c
Update to 1.12.3.
*****
1 /* $Id: mandoc.c,v 1.74 2013/12/30 18:30:32 schwarze Exp $ */
1 /* $Id: mandoc.c,v 1.62 2011/12/03 16:08:51 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHORS DISCLAIM ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <errno.h>
27 #include <limits.h>
28 #include <stdlib.h>
29 #include <stdio.h>
30 #include <string.h>
31 #include <time.h>
32
33 #include "mandoc.h"
34 #include "libmandoc.h"
35
36 #define DATESIZE 32
37
38 static int a2time(time_t *, const char *, const char *);
39 static char *time2a(time_t);
40 static int numescape(const char *);
41
42 enum mandoc_esc
43 mandoc_escape(const char **end, const char **start, int *sz)
44 /*
45  * Pass over recursive numerical expressions. This context of this
46  * function is important: it's only called within character-terminating
47  * escapes (e.g., \s[xxxxyy]), so all we need to do is handle initial
48  * recursion: we don't care about what's in these blocks.
49  * This returns the number of characters skipped or -1 if an error
50  * occurs (the caller should bail).
51 */
52 static int
53 numescape(const char *start)
54 {
55     const char *local_start;
56     int local_sz;
57     char term;
58     enum mandoc_esc gly;

```

new/usr/src/cmd/mandoc/mandoc.c

2

```
53     int i;
54     size_t sz;
55     const char *cp;
56
57     i = 0;
58
59     /* The expression consists of a subexpression. */
60
61     if ('\\"' == start[i]) {
62         cp = &start[++i];
63     }
64     /*
65     * When the caller doesn't provide return storage,
66     * use local storage.
67     * Read past the end of the subexpression.
68     * Bail immediately on errors.
69     */
70     if (ESCAPE_ERROR == mandoc_escape(&cp, NULL, NULL))
71         return(-1);
72     return(i + cp - &start[i]);
73
74
75     if (NULL == start)
76         start = &local_start;
77     if (NULL == sz)
78         sz = &local_sz;
79     if ('(' != start[i++])
80         return(0);
81
82     /*
83     * Beyond the backslash, at least one input character
84     * is part of the escape sequence. With one exception
85     * (see below), that character won't be returned.
86     * A parenthesised subexpression. Read until the closing
87     * parenthesis, making sure to handle any nested subexpressions
88     * that might ruin our parse.
89     */
90     while (')' != start[i]) {
91         sz = strcspn(&start[i], "\\");
92         i += (int)sz;
93
94         if ('\0' == start[i])
95             return(-1);
96         else if ('\' != start[i])
97             continue;
98
99         cp = &start[++i];
100        if (ESCAPE_ERROR == mandoc_escape(&cp, NULL, NULL))
101            return(-1);
102        i += cp - &start[i];
103    }
104
105    /* Read past the terminating ')'. */
106    return(++i);
107 }
108
109 enum mandoc_esc
110 mandoc_escape(const char **end, const char **start, int *sz)
111 {
112     char c, term, numeric;
113     int i, lim, ssz, rlim;
114     const char *cp, *rstart;
115     enum mandoc_esc gly;
116
117     cp = *end;
118     rstart = cp;

```

```

110     if (start)
111         *start = rstart;
112     i = lim = 0;
66     gly = ESCAPE_ERROR;
67     *start = ++end;
68     *sz = 0;
69     term = '\0';
114     term = numeric = '\0';

71     switch ((*start)[-1]) {
116     switch ((c = cp[i++])) {
72         /*
73          * First the glyphs.  There are several different forms of
74          * these, but each eventually returns a substring of the glyph
75          * name.
76          */
77     case ('('):
78         gly = ESCAPE_SPECIAL;
79         *sz = 2;
124     lim = 2;
80     break;
81     case ('['):
82         gly = ESCAPE_SPECIAL;
83         /*
84          * Unicode escapes are defined in groff as \[uXXXX] to
85          * \[ul0FFFF], where the contained value must be a valid
86          * Unicode codepoint.  Here, however, only check whether
87          * it's not a zero-width escape.
88          */
89     if ('u' == (*start)[0] && ']' != (*start)[1])
134     if ('u' == cp[i] && ']' != cp[i + 1])
90         gly = ESCAPE_UNICODE;
91         term = ']';
92     break;
93     case ('C'):
94     if ('\'' != **start)
139     if ('\'' != cp[i])
95         return(ESCAPE_ERROR);
96     *start = ++end;
97     if ('u' == (*start)[0] && '\'' != (*start)[1])
98         gly = ESCAPE_UNICODE;
99     else
100         gly = ESCAPE_SPECIAL;
101     term = '\'';
102     break;

104     /*
105     * Escapes taking no arguments at all.
106     */
107     case ('d'):
108         /* FALLTHROUGH */
109     case ('u'):
110         return(ESCAPE_IGNORE);

112     /*
113     * The \z escape is supposed to output the following
114     * character without advancing the cursor position.
115     * Since we are mostly dealing with terminal mode,
116     * let us just skip the next character.
117     */
118     case ('z'):
119         return(ESCAPE_SKIPCHAR);

121     /*
122     * Handle all triggers matching \X(xy, \Xx, and \X[xxxx], where
123     * 'X' is the trigger.  These have opaque sub-strings.

```

```

124     /*
125     case ('F'):
126         /* FALLTHROUGH */
127     case ('g'):
128         /* FALLTHROUGH */
129     case ('k'):
130         /* FALLTHROUGH */
131     case ('M'):
132         /* FALLTHROUGH */
133     case ('m'):
134         /* FALLTHROUGH */
135     case ('n'):
136         /* FALLTHROUGH */
137     case ('V'):
138         /* FALLTHROUGH */
139     case ('Y'):
140         gly = ESCAPE_IGNORE;
141         /* FALLTHROUGH */
142     case ('f'):
143         if (ESCAPE_ERROR == gly)
144             gly = ESCAPE_FONT;
145         switch (**start) {

170         rstart = &cp[i];
171         if (start)
172             *start = rstart;

174         switch (cp[i++]) {
146         case ('('):
147             *start = ++end;
148             *sz = 2;
176             lim = 2;
149             break;
150         case ('['):
151             *start = ++end;
152             term = ']';
153             break;
154         default:
155             *sz = 1;
182             lim = 1;
183             i--;
156             break;
157         }
158         break;

160     /*
161     * These escapes are of the form \X'Y', where 'X' is the trigger
162     * and 'Y' is any string.  These have opaque sub-strings.
163     */
164     case ('A'):
165         /* FALLTHROUGH */
166     case ('b'):
167         /* FALLTHROUGH */
168     case ('B'):
169         /* FALLTHROUGH */
170     case ('D'):
171         /* FALLTHROUGH */
172     case ('o'):
173         /* FALLTHROUGH */
174     case ('R'):
175         /* FALLTHROUGH */
176     case ('w'):
177         /* FALLTHROUGH */
178     case ('X'):
179         /* FALLTHROUGH */
180     case ('Z'):

```

```

181         if ('\'' != **start)
205         if ('\'' != cp[i++])
182             return(ESCAPE_ERROR);
183         gly = ESCAPE_IGNORE;
184         *start = ++end;
185         term = '\';
186         break;

188     /*
189     * These escapes are of the form \X'N', where 'X' is the trigger
190     * and 'N' resolves to a numerical expression.
191     */
215     case ('B'):
216         /* FALLTHROUGH */
192     case ('h'):
193         /* FALLTHROUGH */
194     case ('H'):
195         /* FALLTHROUGH */
196     case ('L'):
197         /* FALLTHROUGH */
198     case ('l'):
224         gly = ESCAPE_NUMBERED;
199         /* FALLTHROUGH */
200     case ('S'):
201         /* FALLTHROUGH */
202     case ('v'):
203         /* FALLTHROUGH */
230     case ('w'):
231         /* FALLTHROUGH */
204     case ('x'):
205         if ('\'' != **start)
206             return(ESCAPE_ERROR);
233         if (ESCAPE_ERROR == gly)
207             gly = ESCAPE_IGNORE;
208         *start = ++end;
209         term = '\';
235         if ('\'' != cp[i++])
236             return(ESCAPE_ERROR);
237         term = numeric = '\';
210         break;

212     /*
213     * Special handling for the numbered character escape.
214     * XXX Do any other escapes need similar handling?
215     */
216     case ('N'):
217         if ('\0' == **start)
245         if ('\0' == cp[i])
218             return(ESCAPE_ERROR);
219         (*end)++;
220         if (isdigit((unsigned char)**start)) {
221             *sz = 1;
247             *end = &cp[+i];
248             if (isdigit((unsigned char)cp[i-1]))
222                 return(ESCAPE_IGNORE);
223         }
224         (*start)++;
225         while (isdigit((unsigned char)**end))
226             (*end)++;
227         *sz = *end - *start;
252         if (start)
253             *start = &cp[i];
254         if (sz)
255             *sz = *end - &cp[i];
228         if ('\0' != **end)
229             (*end)++;

```

```

230         return(ESCAPE_NUMBERED);

232     /*
233     * Sizes get a special category of their own.
234     */
235     case ('s'):
236         gly = ESCAPE_IGNORE;

266         rstart = &cp[i];
267         if (start)
268             *start = rstart;

238     /* See +/- counts as a sign. */
239     if ('+' == **end || '-' == **end || ASCII_HYPH == **end)
240         (*end)++;
271     c = cp[i];
272     if ('+' == c || '-' == c || ASCII_HYPH == c)
273         ++i;

242     switch (**end) {
275     switch (cp[i++]) {
243     case (''):
244         *start = ++end;
245         *sz = 2;
277         lim = 2;
246         break;
247     case ('['):
248         *start = ++end;
249         term = ']';
280         term = numeric = '\';
250         break;
251     case ('\'):
252         *start = ++end;
253         term = '\';
283         term = numeric = '\';
254         break;
255     default:
256         *sz = 1;
286         lim = 1;
287         i--;
257         break;
258     }

291     /* See +/- counts as a sign. */
292     c = cp[i];
293     if ('+' == c || '-' == c || ASCII_HYPH == c)
294         ++i;

260         break;

262     /*
263     * Anything else is assumed to be a glyph.
264     * In this case, pass back the character after the backslash.
265     */
266     default:
267         gly = ESCAPE_SPECIAL;
268         *start = --end;
269         *sz = 1;
303         lim = 1;
304         i--;
270         break;
271     }

273     assert(ESCAPE_ERROR != gly);

310     rstart = &cp[i];

```

```

311     if (start)
312         *start = rstart;

275     /*
276     * Read up to the terminating character,
277     * paying attention to nested escapes.
315     * If a terminating block has been specified, we need to
316     * handle the case of recursion, which could have their
317     * own terminating blocks that mess up our parse. This, by the
318     * way, means that the "start" and "size" values will be
319     * effectively meaningless.
278     */

322     ssize = 0;
323     if (numeric && -1 == (ssize = numescape(&cp[i])))
324         return(ESCAPE_ERROR);

326     i += ssize;
327     rlim = -1;

329     /*
330     * We have a character terminator. Try to read up to that
331     * character. If we can't (i.e., we hit the nil), then return
332     * an error; if we can, calculate our length, read past the
333     * terminating character, and exit.
334     */

280     if ('\0' != term) {
281         while (**end != term) {
282             switch (**end) {
283                 case ('\0'):
337                     *end = strchr(&cp[i], term);
338                     if ('\0' == *end)
284                         return(ESCAPE_ERROR);
285                 case ('\n'):

341                     rlim = *end - &cp[i];
342                     if (sz)
343                         *sz = rlim;
286                     (*end)++;
287                     if (ESCAPE_ERROR ==
288                         mandoc_escape(end, NULL, NULL))
289                         return(ESCAPE_ERROR);
290                     break;
291                 default:
292                     (*end)++;
293                     break;
345                 goto out;
294             }
295         }
296         *sz = (*end)++ - *start;
297     } else {
298         assert(*sz > 0);
299         if ((size_t)*sz > strlen(*start))

348         assert(lim > 0);

350     /*
351     * We have a numeric limit. If the string is shorter than that,
352     * stop and return an error. Else adjust our endpoint, length,
353     * and return the current glyph.
354     */

356     if ((size_t)lim > strlen(&cp[i]))
300         return(ESCAPE_ERROR);
301     *end += *sz;

```

```

302     }

359     rlim = lim;
360     if (sz)
361         *sz = rlim;

363     *end = &cp[i] + lim;

365 out:
366     assert(rlim >= 0 && rstart);

304     /* Run post-processors. */

306     switch (gly) {
307     case (ESCAPE_FONT):
308         if (2 == *sz) {
309             if ('C' == **start) {
310                 /*
311                 * Treat constant-width font modes
312                 * just like regular font modes.
313                 * Pretend that the constant-width font modes are the
314                 * same as the regular font modes.
315                 */
316                 (*start)++;
317                 (*sz)--;
318             } else {
319                 if ('B' == (*start)[0] && 'I' == (*start)[1])
318                     gly = ESCAPE_FONTBI;
319                 if (2 == rlim && 'C' == *rstart)
317                     rstart++;
318                 else if (1 != rlim)
319                     break;
320             }
321         } else if (1 != *sz)
322             break;

324         switch (**start) {
381         switch (*rstart) {
325         case ('3'):
326             /* FALLTHROUGH */
327         case ('B'):
328             gly = ESCAPE_FONTBOLD;
329             break;
330         case ('2'):
331             /* FALLTHROUGH */
332         case ('I'):
333             gly = ESCAPE_FONTITALIC;
334             break;
335         case ('P'):
336             gly = ESCAPE_FONTPREV;
337             break;
338         case ('1'):
339             /* FALLTHROUGH */
340         case ('R'):
341             gly = ESCAPE_FONTROMAN;
342             break;
343         }
344         break;
345     case (ESCAPE_SPECIAL):
346         if (1 == *sz && 'c' == **start)
403             if (1 != rlim)
404                 break;
405             if ('c' == *rstart)
347                 gly = ESCAPE_NOSPACE;
348         break;
349     default:

```

```

350         break;
351     }

353     return(gly);
354 }
    unchanged portion omitted

424 /*
425  * Parse a quoted or unquoted roff-style request or macro argument.
426  * Return a pointer to the parsed argument, which is either the original
427  * pointer or advanced by one byte in case the argument is quoted.
428  * NUL-terminate the argument in place.
429  * Null-terminate the argument in place.
430  * Collapse pairs of quotes inside quoted arguments.
431  * Advance the argument pointer to the next argument,
432  * or to the NUL byte terminating the argument line.
433  * or to the null byte terminating the argument line.
434 */
435 char *
436 mandoc_getarg(struct mparse *parse, char **cpp, int ln, int *pos)
437 {
438     char    *start, *cp;
439     int     quoted, pairs, white;

440     /* Quoting can only start with a new word. */
441     start = *cpp;
442     quoted = 0;
443     if ('"' == *start) {
444         quoted = 1;
445         start++;
446     }

447     pairs = 0;
448     white = 0;
449     for (cp = start; '\0' != *cp; cp++) {

450         /*
451          * Move the following text left
452          * after quoted quotes and after "\" and "\t".
453          */
454         /* Move left after quoted quotes and escaped backslashes. */
455         if (pairs)
456             cp[-pairs] = cp[0];

457         if ('\\' == cp[0]) {
458             /*
459              * In copy mode, translate double to single
460              * backslashes and backslash-t to literal tabs.
461              */
462             switch (cp[1]) {
463             case ('t'):
464                 cp[0] = '\t';
465                 /* FALLTHROUGH */
466             case ('\\'):
467                 if ('\\' == cp[1]) {
468                     /* Poor man's copy mode. */
469                     pairs++;
470                     cp++;
471                     break;
472                 } else if (0 == quoted && ' ' == cp[1])
473                     /* Skip escaped blanks. */
474                     if (0 == quoted)
475                         cp++;
476                 break;
477             default:

```

```

477         break;
478     }
479     } else if (0 == quoted) {
480         if (' ' == cp[0]) {
481             /* Unescaped blanks end unquoted args. */
482             white = 1;
483             break;
484         }
485     } else if ('"' == cp[0]) {
486         if ('"' == cp[1]) {
487             /* Quoted quotes collapse. */
488             pairs++;
489             cp++;
490         } else {
491             /* Unquoted quotes end quoted args. */
492             quoted = 2;
493             break;
494         }
495     }
496 }

497 /* Quoted argument without a closing quote. */
498 if (1 == quoted)
499     mandoc_msg(MANDOCERR_BADQUOTE, parse, ln, *pos, NULL);

500
501 /* NUL-terminate this argument and move to the next one. */
502 /* Null-terminate this argument and move to the next one. */
503 if (pairs)
504     cp[-pairs] = '\0';
505 if ('\0' != *cp) {
506     *cp++ = '\0';
507     while (' ' == *cp)
508         cp++;
509 }
510 *pos += (int)(cp - start) + (quoted ? 1 : 0);
511 *cpp = cp;

512
513 if ('\0' == *cp && (white || ' ' == cp[-1]))
514     mandoc_msg(MANDOCERR_EOLNSPACE, parse, ln, *pos, NULL);

515
516 return(start);
517 }
    unchanged portion omitted

680 /*
681  * Find out whether a line is a macro line or not. If it is, adjust the
682  * current position and return one; if it isn't, return zero and don't
683  * change the current position.
684  */
685 int
686 mandoc_getcontrol(const char *cp, int *ppos)
687 {
688     int     pos;

689     pos = *ppos;

690
691     if ('\\' == cp[pos] && '.' == cp[pos + 1])
692         pos += 2;
693     else if ('.' == cp[pos] || '\\' == cp[pos])
694         pos++;
695     else
696         return(0);
697
698     while (' ' == cp[pos] || '\t' == cp[pos])
699         pos++;
700 }

```

```
702     *ppos = pos;
703     return(1);
704 }

639 /*
640  * Convert a string to a long that may not be <0.
641  * If the string is invalid, or is less than 0, return -1.
642  */
643 int
644 mandoc_strntoi(const char *p, size_t sz, int base)
645 {
646     char        buf[32];
647     char        *ep;
648     long        v;

650     if (sz > 31)
651         return(-1);

653     memcpy(buf, p, sz);
654     buf[(int)sz] = '\0';

656     errno = 0;
657     v = strtol(buf, &ep, base);

659     if (buf[0] == '\0' || *ep != '\0')
660         return(-1);

662     if (v > INT_MAX)
663         v = INT_MAX;
664     if (v < INT_MIN)
665         v = INT_MIN;

667     return((int)v);
668 }
unchanged_portion_omitted
```



```

*****
14191 Fri Jul 25 15:00:13 2014
new/usr/src/cmd/mandoc/mandoc.h
Update to 1.12.3.
*****
1 /* $Id: mandoc.h,v 1.112 2013/12/30 18:30:32 schwarze Exp $ */
1 /* $Id: mandoc.h,v 1.99 2012/02/16 20:51:31 joerg Exp $ */
2 /*
3 * Copyright (c) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4 * Copyright (c) 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
5 *
6 * Permission to use, copy, modify, and distribute this software for any
7 * purpose with or without fee is hereby granted, provided that the above
8 * copyright notice and this permission notice appear in all copies.
9 *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifndef MANDOC_H
19 #define MANDOC_H
20
21 #define ASCII_NBRSP 31 /* non-breaking space */
22 #define ASCII_HYPH 30 /* breakable hyphen */
23
24 /*
25 * Status level. This refers to both internal status (i.e., whilst
26 * running, when warnings/errors are reported) and an indicator of a
27 * threshold of when to halt (when said internal state exceeds the
28 * threshold).
29 */
30 enum mandoclevel {
31     MANDOCLEVEL_OK = 0,
32     MANDOCLEVEL_RESERVED,
33     MANDOCLEVEL_WARNING, /* warnings: syntax, whitespace, etc. */
34     MANDOCLEVEL_ERROR, /* input has been thrown away */
35     MANDOCLEVEL_FATAL, /* input is borked */
36     MANDOCLEVEL_BADARG, /* bad argument in invocation */
37     MANDOCLEVEL_SYSERR, /* system error */
38     MANDOCLEVEL_MAX
39 };
40
41 /*
42 * All possible things that can go wrong within a parse, be it libroff,
43 * libmdoc, or libman.
44 */
45 enum mandocerr {
46     MANDOCERR_OK,
47
48     MANDOCERR_WARNING, /* ===== start of warnings ===== */
49
50     /* related to the prologue */
51     MANDOCERR_NOTITLE, /* no title in document */
52     MANDOCERR_UPPERCASE, /* document title should be all caps */
53     MANDOCERR_BADMSEC, /* unknown manual section */
54     MANDOCERR_BADVOLARCH, /* unknown manual volume or arch */
55     MANDOCERR_NODATE, /* date missing, using today's date */
56     MANDOCERR_BADDATE, /* cannot parse date, using it verbatim */
57     MANDOCERR_PROLOGOOO, /* prologue macros out of order */
58     MANDOCERR_PROLOGREP, /* duplicate prologue macro */
59     MANDOCERR_BADPROLOG, /* macro not allowed in prologue */
60     MANDOCERR_BADBODY, /* macro not allowed in body */

```

```

62     /* related to document structure */
63     MANDOCERR_SO, /* .so is fragile, better use ln(1) */
64     MANDOCERR_NAMESECFIRST, /* NAME section must come first */
65     MANDOCERR_BADNAMESEC, /* bad NAME section contents */
66     MANDOCERR_NONAME, /* manual name not yet set */
67     MANDOCERR_SECOOO, /* sections out of conventional order */
68     MANDOCERR_SECREP, /* duplicate section name */
69     MANDOCERR_SECMSEC, /* section header suited to sections ... */
70     MANDOCERR_SECMSEC, /* section not in conventional manual section */
71
72     /* related to macros and nesting */
73     MANDOCERR_MACROOBS, /* skipping obsolete macro */
74     MANDOCERR_IGNPAR, /* skipping paragraph macro */
75     MANDOCERR_MOVEPAR, /* moving paragraph macro out of list */
76     MANDOCERR_IGNNS, /* skipping no-space macro */
77     MANDOCERR_SCOPENEST, /* blocks badly nested */
78     MANDOCERR_CHILD, /* child violates parent syntax */
79     MANDOCERR_NESTEDDISP, /* nested displays are not portable */
80     MANDOCERR_SCOPEREPEP, /* already in literal mode */
81     MANDOCERR_LINESCOPE, /* line scope broken */
82
83     /* related to missing macro arguments */
84     MANDOCERR_MACROEMPTY, /* skipping empty macro */
85     MANDOCERR_ARGCWRN, /* argument count wrong */
86     MANDOCERR_DISPTYPE, /* missing display type */
87     MANDOCERR_LISTFIRST, /* list type must come first */
88     MANDOCERR_NOWIDTHARG, /* tag lists require a width argument */
89     MANDOCERR_FONTTYPE, /* missing font type */
90     MANDOCERR_WNOSCOPE, /* skipping end of block that is not open */
91
92     /* related to bad macro arguments */
93     MANDOCERR_IGNARGV, /* skipping argument */
94     MANDOCERR_ARGVREP, /* duplicate argument */
95     MANDOCERR_DISPREP, /* duplicate display type */
96     MANDOCERR_LISTREP, /* duplicate list type */
97     MANDOCERR_BADATT, /* unknown AT&T UNIX version */
98     MANDOCERR_BADBOOL, /* bad Boolean value */
99     MANDOCERR_BADFONT, /* unknown font */
100     MANDOCERR_BADSTANDARD, /* unknown standard specifier */
101     MANDOCERR_BADWIDTH, /* bad width argument */
102
103     /* related to plain text */
104     MANDOCERR_NOBLANKLN, /* blank line in non-literal context */
105     MANDOCERR_BADTAB, /* tab in non-literal context */
106     MANDOCERR_EOLNSPACE, /* end of line whitespace */
107     MANDOCERR_BADCOMMENT, /* bad comment style */
108     MANDOCERR_BADESCAPE, /* unknown escape sequence */
109     MANDOCERR_BADQUOTE, /* unterminated quoted string */
110
111     /* related to equations */
112     MANDOCERR_EQNQUOTE, /* unexpected literal in equation */
113
114     MANDOCERR_ERROR, /* ===== start of errors ===== */
115
116     /* related to equations */
117     MANDOCERR_EQNNSCOPE, /* unexpected equation scope closure */
118     MANDOCERR_EQNSCOPE, /* equation scope open on exit */
119     MANDOCERR_EQNBADSCOPE, /* overlapping equation scopes */
120     MANDOCERR_EQNEOF, /* unexpected end of equation */
121     MANDOCERR_EQNSYNT, /* equation syntax error */
122
123     /* related to tables */
124     MANDOCERR_TBL, /* bad table syntax */
125     MANDOCERR_TBLOPT, /* bad table option */
126     MANDOCERR_TBLLAYOUT, /* bad table layout */

```

```

125 MANDOCERR_TBLNOLAYOUT, /* no table layout cells specified */
126 MANDOCERR_TBLNODATA, /* no table data cells specified */
127 MANDOCERR_TBLIGNDATA, /* ignore data in cell */
128 MANDOCERR_TBLEBLOCK, /* data block still open */
129 MANDOCERR_TBLEXTRADAT, /* ignoring extra data cells */

131 MANDOCERR_ROFFLOOP, /* input stack limit exceeded, infinite loop? */
132 MANDOCERR_BADCHAR, /* skipping bad character */
133 MANDOCERR_NAMESC, /* escaped character not allowed in a name */
134 MANDOCERR_NONAME, /* manual name not yet set */
135 MANDOCERR_NOTEXT, /* skipping text before the first section header */
136 MANDOCERR_MACRO, /* skipping unknown macro */
137 MANDOCERR_REQUEST, /* NOT IMPLEMENTED: skipping request */
138 MANDOCERR_ARGCOUNT, /* argument count wrong */
139 MANDOCERR_STRAYTA, /* skipping column outside column list */
140 MANDOCERR_NOSCOPE, /* skipping end of block that is not open */
141 MANDOCERR_SCOPEBROKEN, /* missing end of block */
142 MANDOCERR_SCOPEEXIT, /* scope open on exit */
143 MANDOCERR_UNAME, /* uname(3) system call failed */
144 /* FIXME: merge following with MANDOCERR_ARGCOUNT */
145 MANDOCERR_NOARGS, /* macro requires line argument(s) */
146 MANDOCERR_NOBODY, /* macro requires body argument(s) */
147 MANDOCERR_NOARGV, /* macro requires argument(s) */
148 MANDOCERR_NUMERIC, /* request requires a numeric argument */
149 MANDOCERR_LISTTYPE, /* missing list type */
150 MANDOCERR_ARGSLIST, /* line argument(s) will be lost */
151 MANDOCERR_BODYLOST, /* body argument(s) will be lost */

153 MANDOCERR_FATAL, /* ===== start of fatal errors ===== */

155 MANDOCERR_NOTMANUAL, /* manual isn't really a manual */
156 MANDOCERR_COLUMNS, /* column syntax is inconsistent */
157 MANDOCERR_BADDISP, /* NOT IMPLEMENTED: .Bd -file */
158 MANDOCERR_SYNTARGVCOUNT, /* argument count wrong, violates syntax */
159 MANDOCERR_SYNTCHILD, /* child violates parent syntax */
160 MANDOCERR_SYNTARGCOUNT, /* argument count wrong, violates syntax */
161 MANDOCERR_SOPATH, /* NOT IMPLEMENTED: .so with absolute path or ".." */
162 MANDOCERR_NODOCBODY, /* no document body */
163 MANDOCERR_NODOCPROLOG, /* no document prologue */
164 MANDOCERR_MEM, /* static buffer exhausted */
165 MANDOCERR_MAX
166 };

168 struct tbl_opts {
169 struct tbl {
170 char tab; /* cell-separator */
171 char decimal; /* decimal point */
172 int linesize;
173 int opts;
174 #define TBL_OPT_CENTRE (1 << 0)
175 #define TBL_OPT_EXPAND (1 << 1)
176 #define TBL_OPT_BOX (1 << 2)
177 #define TBL_OPT_DBOX (1 << 3)
178 #define TBL_OPT_ALLBOX (1 << 4)
179 #define TBL_OPT_NOKEEP (1 << 5)
180 #define TBL_OPT_NOSPACE (1 << 6)
181 int cols; /* number of columns */
182 };

183 enum tbl_headt {
184 TBL_HEAD_DATA, /* plug in data from tbl_dat */
185 TBL_HEAD_VERT, /* vertical spacer */
186 TBL_HEAD_DVERT /* double-vertical spacer */
187 };

```

```

184 * The head of a table specifies all of its columns. When formatting a
185 * tbl_span, iterate over these and plug in data from the tbl_span when
186 * appropriate, using tbl_cell as a guide to placement.
187 */
188 struct tbl_head {
189 enum tbl_headt pos;
190 int ident; /* 0 <= unique id < cols */
191 int vert; /* width of preceding vertical line */
192 struct tbl_head *next;
193 struct tbl_head *prev;
194 };

195 enum tbl_cellt {
196 TBL_CELL_CENTRE, /* c, C */
197 TBL_CELL_RIGHT, /* r, R */
198 TBL_CELL_LEFT, /* l, L */
199 TBL_CELL_NUMBER, /* n, N */
200 TBL_CELL_SPAN, /* s, S */
201 TBL_CELL_LONG, /* a, A */
202 TBL_CELL_DOWN, /* ^ */
203 TBL_CELL_HORIZ, /* -, - */
204 TBL_CELL_DHORIZ, /* = */
205 TBL_CELL_VERT, /* | */
206 TBL_CELL_DVERT, /* || */
207 TBL_CELL_MAX
208 };

209 /*
210 * A cell in a layout row.
211 */
212 struct tbl_cell {
213 struct tbl_cell *next;
214 int vert; /* width of preceding vertical line */
215 enum tbl_cellt pos;
216 size_t spacing;
217 int flags;
218 #define TBL_CELL_TALIGN (1 << 0) /* t, T */
219 #define TBL_CELL_BALIGN (1 << 1) /* d, D */
220 #define TBL_CELL_BOLD (1 << 2) /* fB, B, b */
221 #define TBL_CELL_ITALIC (1 << 3) /* fI, I, i */
222 #define TBL_CELL_EQUAL (1 << 4) /* e, E */
223 #define TBL_CELL_UP (1 << 5) /* u, U */
224 #define TBL_CELL_WIGN (1 << 6) /* z, Z */
225 struct tbl_head *head;
226 };

227 /*
228 * A row of data in a table.
229 */
230 struct tbl_span {
231 struct tbl_opts *opts;
232 struct tbl *tbl;
233 struct tbl_head *head;
234 struct tbl_row *layout; /* layout row */
235 struct tbl_dat *first;
236 struct tbl_dat *last;
237 int line; /* parse line */
238 int flags;
239 #define TBL_SPAN_FIRST (1 << 0)
240 #define TBL_SPAN_LAST (1 << 1)
241 enum tbl_spant pos;
242 struct tbl_span *next;
243 };

```

```

376 enum      mandoc_esc {
377     ESCAPE_ERROR = 0, /* bail! unparsable escape */
378     ESCAPE_IGNORE, /* escape to be ignored */
379     ESCAPE_SPECIAL, /* a regular special character */
380     ESCAPE_FONT, /* a generic font mode */
381     ESCAPE_FONTBOLD, /* bold font mode */
382     ESCAPE_FONTITALIC, /* italic font mode */
383     ESCAPE_FONTBI, /* bold italic font mode */
384     ESCAPE_FONTRoman, /* roman font mode */
385     ESCAPE_FONTPREV, /* previous font mode */
386     ESCAPE_NUMBERED, /* a numbered glyph */
387     ESCAPE_UNICODE, /* a unicode codepoint */
388     ESCAPE_NOSPACE, /* suppress space if the last on a line */
389     ESCAPE_SKIPCHAR /* skip the next character */
389     ESCAPE_NOSPACE /* suppress space if the last on a line */
390 };

392 typedef void      (*mandocmsg)(enum mandocerr, enum mandoclevel,
393                                const char *, int, int, const char *);

395 struct  mparse;
396 struct  mchars;
397 struct  mdoc;
398 struct  man;

400 __BEGIN_DECLS

402 void      *mandoc_calloc(size_t, size_t);
403 enum mandoc_esc  mandoc_escape(const char **, const char **, int *);
404 void      *mandoc_malloc(size_t);
405 void      *mandoc_realloc(void *, size_t);
406 char      *mandoc_strdup(const char *);
407 char      *mandoc_strndup(const char *, size_t);
408 struct mchars  *mchars_alloc(void);
409 void      mchars_free(struct mchars *);
410 char      mchars_num2char(const char *, size_t);
411 int       mchars_num2uc(const char *, size_t);
412 int       mchars_spec2cp(const struct mchars *,
413                           const char *, size_t);
414 const char *mchars_spec2str(const struct mchars *,
415                             const char *, size_t, size_t *);
416 struct mparse  *mparse_alloc(enum mparset, enum mandoclevel,
417                             mandocmsg, void *, char *);
416 struct mparse  *mparse_alloc(enum mparset,
417                             enum mandoclevel, mandocmsg, void *);
418 void      mparse_free(struct mparse *);
419 void      mparse_keep(struct mparse *);
420 enum mandoclevel  mparse_readfd(struct mparse *, int, const char *);
421 enum mandoclevel  mparse_readmem(struct mparse *, const void *, size_t,
422                                  const char *);
423 void      mparse_reset(struct mparse *);
424 void      mparse_result(struct mparse *,
425                          struct mdoc **, struct man **);
426 const char *mparse_getkeep(const struct mparse *);
427 const char *mparse_strerror(enum mandocerr);
428 const char *mparse_strlevel(enum mandoclevel);

430 __END_DECLS

432 #endif /*!MANDOC_H*/

```

```

*****
21593 Fri Jul 25 15:00:13 2014
new/usr/src/cmd/mandoc/mdoc.c
Update to 1.12.3.
*****
1 /* $Id: mdoc.c,v 1.206 2013/12/24 19:11:46 schwarze Exp $ */
1 /* $Id: mdoc.c,v 1.196 2011/09/30 00:13:28 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <stdarg.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29 #include <time.h>
30
31 #include "mdoc.h"
32 #include "mandoc.h"
33 #include "libmdoc.h"
34 #include "libmandoc.h"
35
36 const char *const __mdoc_macronames[MDOC_MAX] = {
37     "Ap", "Dd", "Dt", "Os",
38     "Sh", "Ss", "Pp", "Dl",
39     "Dl", "Bd", "Ed", "Bl",
40     "El", "It", "Ad", "An",
41     "Ar", "Cd", "Cm", "Dv",
42     "Er", "Ev", "Ex", "Fa",
43     "Fd", "Fl", "Fn", "Fr",
44     "Ic", "In", "Li", "Nd",
45     "Nm", "Op", "Ot", "Pa",
46     "Rv", "St", "Va", "Vt",
47     /* LINTED */
48     "Xr", "%A", "%B", "%D",
49     /* LINTED */
50     "%I", "%J", "%N", "%O",
51     /* LINTED */
52     "%P", "%R", "%T", "%V",
53     "Ac", "Ao", "Aq", "At",
54     "Bc", "Bf", "Bo", "Bq",
55     "Bsx", "Bx", "Dc", "Dd",
56     "Do", "Dq", "Ec", "Ef",
57     "Em", "Eo", "Fx", "Ms",
58     "No", "Ns", "Nx", "Ox",
59     "Pc", "Pf", "Po", "Pq",

```

```

60     "Qc", "Ql", "Qo", "Qq",
61     "Re", "Rs", "Sc", "So",
62     "Sq", "Sm", "Sx", "Sy",
63     "Tn", "Ux", "Xc", "Xo",
64     "Fo", "Fc", "Oo", "Oc",
65     "Bk", "Ek", "Bt", "Hf",
66     "Fr", "Ud", "Lb", "Lp",
67     "Lk", "Mt", "Brq", "Bro",
68     /* LINTED */
69     "Brc", "%C", "Es", "En",
70     /* LINTED */
71     "Dx", "%Q", "br", "sp",
72     /* LINTED */
73     "%U", "Ta"
74 };
75
76 const char *const __mdoc_argnames[MDOC_ARG_MAX] = {
77     "split", "nosplit", "ragged",
78     "unfilled", "literal", "file",
79     "offset", "bullet", "dash",
80     "hyphen", "item", "enum",
81     "tag", "diag", "hang",
82     "ohang", "inset", "column",
83     "width", "compact", "std",
84     "filled", "words", "emphasis",
85     "symbolic", "nested", "centered"
86 };
87
88 const char *const *mdoc_macronames = __mdoc_macronames;
89 const char *const *mdoc_argnames = __mdoc_argnames;
90
91 static void mdoc_node_free(struct mdoc_node *);
92 static void mdoc_node_unlink(struct mdoc_node *,
93     struct mdoc_node *);
94 static void mdoc_freel(struct mdoc *);
95 static void mdoc_alloc1(struct mdoc *);
96 static struct mdoc_node *mdoc_alloc(struct mdoc *, int, int,
97     enum mdoct, enum mdoc_type);
98 static int node_append(struct mdoc *,
99     struct mdoc_node *);
100 #if 0
101 static int mdoc_preptext(struct mdoc *, int, char *, int);
102 #endif
103 static int mdoc_ptext(struct mdoc *, int, char *, int);
104 static int mdoc_pmacro(struct mdoc *, int, char *, int);
105
106 const struct mdoc_node *
107 mdoc_node(const struct mdoc *mdoc)
108 {
109     assert(! (MDOC_HALT & mdoc->flags));
110     return(mdoc->first);
111     assert(! (MDOC_HALT & m->flags));
112     return(m->first);
113 }
114
115 const struct mdoc_meta *
116 mdoc_meta(const struct mdoc *mdoc)
117 {
118     assert(! (MDOC_HALT & mdoc->flags));
119     return(&mdoc->meta);
120     assert(! (MDOC_HALT & m->flags));

```

```

120     return(&m->meta);
121 }
_____ unchanged_portion_omitted _____

196 /*
197 * Allocate volatile and non-volatile parse resources.
198 */
199 struct mdoc *
200 mdoc_alloc(struct roff *roff, struct mparse *parse, char *defos)
201 mdoc_alloc(struct roff *roff, struct mparse *parse)
202 {
203     struct mdoc *p;
204
205     p = mandoc_calloc(1, sizeof(struct mdoc));
206
207     p->parse = parse;
208     p->defos = defos;
209     p->roff = roff;
210
211     mdoc_hash_init();
212     mdoc_alloc1(p);
213     return(p);
214 }
215
216 /*
217 * Climb back up the parse tree, validating open scopes. Mostly calls
218 * through to macro_end() in macro.c.
219 */
220 int
221 mdoc_endparse(struct mdoc *mdoc)
222 mdoc_endparse(struct mdoc *m)
223 {
224     assert( ! (MDOC_HALT & mdoc->flags));
225     if (mdoc_macroend(mdoc))
226         assert( ! (MDOC_HALT & m->flags));
227     if (mdoc_macroend(m))
228         return(1);
229     mdoc->flags |= MDOC_HALT;
230     m->flags |= MDOC_HALT;
231     return(0);
232 }
233
234 int
235 mdoc_addeqn(struct mdoc *mdoc, const struct eqn *ep)
236 mdoc_addeqn(struct mdoc *m, const struct eqn *ep)
237 {
238     struct mdoc_node *n;
239
240     assert( ! (MDOC_HALT & mdoc->flags));
241     assert( ! (MDOC_HALT & m->flags));
242
243     /* No text before an initial macro. */
244
245     if (SEC_NONE == mdoc->lastnamed) {
246         mdoc_pmsg(mdoc, ep->ln, ep->pos, MANDOCERR_NOTEXT);
247     }
248     if (SEC_NONE == m->lastnamed) {
249         mdoc_pmsg(m, ep->ln, ep->pos, MANDOCERR_NOTEXT);
250     }
251     return(1);
252 }
253
254 n = node_alloc(mdoc, ep->ln, ep->pos, MDOC_MAX, MDOC_EQN);
255 n = node_alloc(m, ep->ln, ep->pos, MDOC_MAX, MDOC_EQN);
256 n->eqn = ep;

```

```

248     if ( ! node_append(mdoc, n))
249     if ( ! node_append(m, n))
250         return(0);
251
252     mdoc->next = MDOC_NEXT_SIBLING;
253     m->next = MDOC_NEXT_SIBLING;
254     return(1);
255 }
256
257 int
258 mdoc_addspan(struct mdoc *mdoc, const struct tbl_span *sp)
259 mdoc_addspan(struct mdoc *m, const struct tbl_span *sp)
260 {
261     struct mdoc_node *n;
262
263     assert( ! (MDOC_HALT & mdoc->flags));
264     assert( ! (MDOC_HALT & m->flags));
265
266     /* No text before an initial macro. */
267
268     if (SEC_NONE == mdoc->lastnamed) {
269         mdoc_pmsg(mdoc, sp->line, 0, MANDOCERR_NOTEXT);
270     }
271     if (SEC_NONE == m->lastnamed) {
272         mdoc_pmsg(m, sp->line, 0, MANDOCERR_NOTEXT);
273     }
274     return(1);
275 }
276
277 n = node_alloc(mdoc, sp->line, 0, MDOC_MAX, MDOC_TBL);
278 n = node_alloc(m, sp->line, 0, MDOC_MAX, MDOC_TBL);
279 n->span = sp;
280
281 if ( ! node_append(mdoc, n))
282 if ( ! node_append(m, n))
283     return(0);
284
285 mdoc->next = MDOC_NEXT_SIBLING;
286 m->next = MDOC_NEXT_SIBLING;
287 return(1);
288 }
289
290 /*
291 * Main parse routine. Parses a single line -- really just hands off to
292 * the macro (mdoc_pmacro()) or text parser (mdoc_ptext()).
293 */
294 int
295 mdoc_parseln(struct mdoc *mdoc, int ln, char *buf, int offs)
296 mdoc_parseln(struct mdoc *m, int ln, char *buf, int offs)
297 {
298     assert( ! (MDOC_HALT & mdoc->flags));
299     assert( ! (MDOC_HALT & m->flags));
300
301     mdoc->flags |= MDOC_NEWLINE;
302     m->flags |= MDOC_NEWLINE;
303
304     /*
305      * Let the roff nS register switch SYNOPSIS mode early,
306      * such that the parser knows at all times
307      * whether this mode is on or off.
308      * Note that this mode is also switched by the Sh macro.
309      */
310     if (roff_getreg(mdoc->roff, "nS"))
311         mdoc->flags |= MDOC_SYNOPSIS;
312     if (roff_regisset(m->roff, REG_nS)) {

```

```

298     if (roff_regget(m->roff, REG_NS))
299         m->flags |= MDOC_SYNOPSIS;
300     else
301         mdoc->flags &= ~MDOC_SYNOPSIS;
301         m->flags &= ~MDOC_SYNOPSIS;
302     }
303     return(roff_getcontrol(mdoc->roff, buf, &offs) ?
304         mdoc_pmacro(mdoc, ln, buf, offs) :
305         mdoc_ptext(mdoc, ln, buf, offs));
304     return(mandoc_getcontrol(buf, &offs) ?
305         mdoc_pmacro(m, ln, buf, offs) :
306         mdoc_ptext(m, ln, buf, offs));
306 }
308 int
309 mdoc_macro(MACRO_PROT_ARGS)
310 {
311     assert(tok < MDOC_MAX);
313     /* If we're in the body, deny prologue calls. */
315     if (MDOC_PROLOGUE & mdoc_macros[tok].flags &&
316         MDOC_PBODY & mdoc->flags) {
317         mdoc_pmsg(mdoc, line, ppos, MANDOCERR_BADBODY);
317         MDOC_PBODY & m->flags) {
318         mdoc_pmsg(m, line, ppos, MANDOCERR_BADBODY);
318         return(1);
319     }
321     /* If we're in the prologue, deny "body" macros. */
323     if (! (MDOC_PROLOGUE & mdoc_macros[tok].flags) &&
324         ! (MDOC_PBODY & mdoc->flags)) {
325         mdoc_pmsg(mdoc, line, ppos, MANDOCERR_BADPROLOG);
326         if (NULL == mdoc->meta.msec)
327             mdoc->meta.msec = mandoc_strdup("1");
328         if (NULL == mdoc->meta.title)
329             mdoc->meta.title = mandoc_strdup("UNKNOWN");
330         if (NULL == mdoc->meta.vol)
331             mdoc->meta.vol = mandoc_strdup("LOCAL");
332         if (NULL == mdoc->meta.os)
333             mdoc->meta.os = mandoc_strdup("LOCAL");
334         if (NULL == mdoc->meta.date)
335             mdoc->meta.date = mandoc_normdate
336                 (mdoc->parse, NULL, line, ppos);
337         mdoc->flags |= MDOC_PBODY;
325         ! (MDOC_PBODY & m->flags)) {
326         mdoc_pmsg(m, line, ppos, MANDOCERR_BADPROLOG);
327         if (NULL == m->meta.msec)
328             m->meta.msec = mandoc_strdup("1");
329         if (NULL == m->meta.title)
330             m->meta.title = mandoc_strdup("UNKNOWN");
331         if (NULL == m->meta.vol)
332             m->meta.vol = mandoc_strdup("LOCAL");
333         if (NULL == m->meta.os)
334             m->meta.os = mandoc_strdup("LOCAL");
335         if (NULL == m->meta.date)
336             m->meta.date = mandoc_normdate
337                 (m->parse, NULL, line, ppos);
338         m->flags |= MDOC_PBODY;
340     return((*mdoc_macros[tok].fp)(mdoc, tok, line, ppos, pos, buf));
341     return((*mdoc_macros[tok].fp)(m, tok, line, ppos, pos, buf));
341 }

```

```

344 static int
345 node_append(struct mdoc *mdoc, struct mdoc_node *p)
346 {
348     assert(mdoc->last);
349     assert(mdoc->first);
350     assert(MDOC_ROOT != p->type);
352     switch (mdoc->next) {
353     case (MDOC_NEXT_SIBLING):
354         mdoc->last->next = p;
355         p->prev = mdoc->last;
356         p->parent = mdoc->last->parent;
357         break;
358     case (MDOC_NEXT_CHILD):
359         mdoc->last->child = p;
360         p->parent = mdoc->last;
361         break;
362     default:
363         abort();
364         /* NOTREACHED */
365     }
367     p->parent->nchild++;
369     /*
370     * Copy over the normalised-data pointer of our parent. Not
371     * everybody has one, but copying a null pointer is fine.
372     */
374     switch (p->type) {
375     case (MDOC_BODY):
376         if (ENDBODY_NOT != p->end)
377             break;
378         /* FALLTHROUGH */
379     case (MDOC_TAIL):
380         /* FALLTHROUGH */
381     case (MDOC_HEAD):
382         p->norm = p->parent->norm;
383         break;
384     default:
385         break;
386     }
388     if (! mdoc_valid_pre(mdoc, p))
389         return(0);
391     switch (p->type) {
392     case (MDOC_HEAD):
393         assert(MDOC_BLOCK == p->parent->type);
394         p->parent->head = p;
395         break;
396     case (MDOC_TAIL):
397         assert(MDOC_BLOCK == p->parent->type);
398         p->parent->tail = p;
399         break;
400     case (MDOC_BODY):
401         if (p->end)
402             break;
403         assert(MDOC_BLOCK == p->parent->type);
404         p->parent->body = p;
405         break;
406     default:
407         break;

```

```

408     }
410     mdoc->last = p;

412     switch (p->type) {
413     case (MDOC_TBL):
414         /* FALLTHROUGH */
415     case (MDOC_TEXT):
416         if ( ! mdoc_valid_post(mdoc))
417             return(0);
418         break;
419     default:
420         break;
421     }

423     return(1);
424 }

427 static struct mdoc_node *
428 node_alloc(struct mdoc *mdoc, int line, int pos,
429 node_alloc(struct mdoc *m, int line, int pos,
430             enum mdoct tok, enum mdoc_type type)
431 {
432     struct mdoc_node *p;

433     p = mandoc_calloc(1, sizeof(struct mdoc_node));
434     p->sec = mdoc->lastsec;
435     p->sec = m->lastsec;
436     p->line = line;
437     p->pos = pos;
438     p->lastline = line;
439     p->tok = tok;
440     p->type = type;

441     /* Flag analysis. */

442     if (MDOC_SYNOPSIS & mdoc->flags)
443         if (MDOC_SYNOPSIS & m->flags)
444             p->flags |= MDOC_SYNPRETTY;
445     else
446         p->flags &= ~MDOC_SYNPRETTY;
447     if (MDOC_NEWLINE & mdoc->flags)
448         if (MDOC_NEWLINE & m->flags)
449             p->flags |= MDOC_LINE;
450     mdoc->flags &= ~MDOC_NEWLINE;
451     m->flags &= ~MDOC_NEWLINE;

452     return(p);
453 }

455 int
456 mdoc_tail_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok)
457 mdoc_tail_alloc(struct mdoc *m, int line, int pos, enum mdoct tok)
458 {
459     struct mdoc_node *p;

460     p = node_alloc(mdoc, line, pos, tok, MDOC_TAIL);
461     if ( ! node_append(mdoc, p))
462         p = node_alloc(m, line, pos, tok, MDOC_TAIL);
463     if ( ! node_append(m, p))
464         return(0);
465     mdoc->next = MDOC_NEXT_CHILD;
466     m->next = MDOC_NEXT_CHILD;
467     return(1);

```

```

465 }

468 int
469 mdoc_head_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok)
470 mdoc_head_alloc(struct mdoc *m, int line, int pos, enum mdoct tok)
471 {
472     struct mdoc_node *p;

473     assert(mdoc->first);
474     assert(mdoc->last);
475     assert(m->first);
476     assert(m->last);

477     p = node_alloc(mdoc, line, pos, tok, MDOC_HEAD);
478     if ( ! node_append(mdoc, p))
479         p = node_alloc(m, line, pos, tok, MDOC_HEAD);
480     if ( ! node_append(m, p))
481         return(0);
482     mdoc->next = MDOC_NEXT_CHILD;
483     m->next = MDOC_NEXT_CHILD;
484     return(1);
485 }

488 int
489 mdoc_body_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok)
490 mdoc_body_alloc(struct mdoc *m, int line, int pos, enum mdoct tok)
491 {
492     struct mdoc_node *p;

493     p = node_alloc(mdoc, line, pos, tok, MDOC_BODY);
494     if ( ! node_append(mdoc, p))
495         p = node_alloc(m, line, pos, tok, MDOC_BODY);
496     if ( ! node_append(m, p))
497         return(0);
498     mdoc->next = MDOC_NEXT_CHILD;
499     m->next = MDOC_NEXT_CHILD;
500     return(1);
501 }

504 int
505 mdoc_endbody_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok,
506 mdoc_endbody_alloc(struct mdoc *m, int line, int pos, enum mdoct tok,
507                     struct mdoc_node *body, enum mdoc_endbody end)
508 {
509     struct mdoc_node *p;

510     p = node_alloc(mdoc, line, pos, tok, MDOC_BODY);
511     p = node_alloc(m, line, pos, tok, MDOC_BODY);
512     p->pending = body;
513     p->norm = body->norm;
514     p->end = end;
515     if ( ! node_append(mdoc, p))
516         if ( ! node_append(m, p))
517             return(0);
518     mdoc->next = MDOC_NEXT_SIBLING;
519     m->next = MDOC_NEXT_SIBLING;
520     return(1);
521 }

524 int
525 mdoc_block_alloc(struct mdoc *mdoc, int line, int pos,
526 mdoc_block_alloc(struct mdoc *m, int line, int pos,

```

```

516         enum mdoct tok, struct mdoc_arg *args)
517 {
518     struct mdoc_node *p;

520     p = node_alloc(mdoc, line, pos, tok, MDOC_BLOCK);
521     p = node_alloc(m, line, pos, tok, MDOC_BLOCK);
522     p->args = args;
523     if (p->args)
524         (args->refcnt)++;

525     switch (tok) {
526     case (MDOC_Bd):
527         /* FALLTHROUGH */
528     case (MDOC_Bf):
529         /* FALLTHROUGH */
530     case (MDOC_Bl):
531         /* FALLTHROUGH */
532     case (MDOC_Rs):
533         p->norm = mandoc_calloc(1, sizeof(union mdoc_data));
534         break;
535     default:
536         break;
537     }

539     if ( ! node_append(mdoc, p))
540     if ( ! node_append(m, p))
541         return(0);
542     mdoc->next = MDOC_NEXT_CHILD;
543     m->next = MDOC_NEXT_CHILD;
544     return(1);
545 }

546 int
547 mdoc_elem_alloc(struct mdoc *mdoc, int line, int pos,
548 mdoc_elem_alloc(struct mdoc *m, int line, int pos,
549         enum mdoct tok, struct mdoc_arg *args)
550 {
551     struct mdoc_node *p;

552     p = node_alloc(mdoc, line, pos, tok, MDOC_ELEM);
553     p = node_alloc(m, line, pos, tok, MDOC_ELEM);
554     p->args = args;
555     if (p->args)
556         (args->refcnt)++;

557     switch (tok) {
558     case (MDOC_An):
559         p->norm = mandoc_calloc(1, sizeof(union mdoc_data));
560         break;
561     default:
562         break;
563     }

565     if ( ! node_append(mdoc, p))
566     if ( ! node_append(m, p))
567         return(0);
568     mdoc->next = MDOC_NEXT_CHILD;
569     m->next = MDOC_NEXT_CHILD;
570     return(1);
571 }

572 int
573 mdoc_word_alloc(struct mdoc *mdoc, int line, int pos, const char *p)
574 mdoc_word_alloc(struct mdoc *m, int line, int pos, const char *p)
575 {

```

```

576     struct mdoc_node *n;

577     n = node_alloc(mdoc, line, pos, MDOC_MAX, MDOC_TEXT);
578     n->string = roff_strdup(mdoc->roff, p);
579     n = node_alloc(m, line, pos, MDOC_MAX, MDOC_TEXT);
580     n->string = roff_strdup(m->roff, p);

581     if ( ! node_append(mdoc, n))
582     if ( ! node_append(m, n))
583         return(0);

584     mdoc->next = MDOC_NEXT_SIBLING;
585     m->next = MDOC_NEXT_SIBLING;
586     return(1);
587 }

588 void
589 mdoc_word_append(struct mdoc *mdoc, const char *p)
590 {
591     struct mdoc_node *n;
592     char *addstr, *newstr;

593     n = mdoc->last;
594     addstr = roff_strdup(mdoc->roff, p);
595     if (-1 == asprintf(&newstr, "%s %s", n->string, addstr)) {
596         perror(NULL);
597         exit((int)MANDOCLEVEL_SYSERR);
598     }
599     free(addstr);
600     free(n->string);
601     n->string = newstr;
602     mdoc->next = MDOC_NEXT_SIBLING;
603 }

604 static void
605 mdoc_node_free(struct mdoc_node *p)
606 {
607     if (MDOC_BLOCK == p->type || MDOC_ELEM == p->type)
608         free(p->norm);
609     if (p->string)
610         free(p->string);
611     if (p->args)
612         mdoc_argv_free(p->args);
613     free(p);
614 }

615

616 static void
617 mdoc_node_unlink(struct mdoc *mdoc, struct mdoc_node *n)
618 mdoc_node_unlink(struct mdoc *m, struct mdoc_node *n)
619 {
620     /* Adjust siblings. */

621     if (n->prev)
622         n->prev->next = n->next;
623     if (n->next)
624         n->next->prev = n->prev;

625     /* Adjust parent. */

626     if (n->parent) {
627         n->parent->nchild--;
628         if (n->parent->nchild == 0)
629             n->parent->child = n->prev ? n->prev : n->next;
630     }

```



```

635         if (n->parent->last == n)
636             n->parent->last = n->prev ? n->prev : NULL;
637     }

639     /* Adjust parse point, if applicable. */

641     if (mdoc && mdoc->last == n) {
642         if (m && m->last == n) {
643             if (n->prev) {
644                 mdoc->last = n->prev;
644                 mdoc->next = MDOC_NEXT_SIBLING;
643                 m->last = n->prev;
644                 m->next = MDOC_NEXT_SIBLING;
645             } else {
646                 mdoc->last = n->parent;
647                 mdoc->next = MDOC_NEXT_CHILD;
646                 m->last = n->parent;
647                 m->next = MDOC_NEXT_CHILD;
648             }
649         }

651         if (mdoc && mdoc->first == n)
652             mdoc->first = NULL;
653         if (m && m->first == n)
654             m->first = NULL;
655     }

656 void
657 mdoc_node_delete(struct mdoc *mdoc, struct mdoc_node *p)
658 {
659     while (p->child) {
660         assert(p->nchild);
661         mdoc_node_delete(mdoc, p->child);
662         mdoc_node_delete(m, p->child);
663     }
664     assert(0 == p->nchild);

666     mdoc_node_unlink(mdoc, p);
667     mdoc_node_unlink(m, p);
668     mdoc_node_free(p);
669 }

670 int
671 mdoc_node_relink(struct mdoc *mdoc, struct mdoc_node *p)
672 {
673     mdoc_node_unlink(mdoc, p);
674     return(node_append(mdoc, p));
675 }

678 #if 0
679 /*
680  * Pre-treat a text line.
681  * Text lines can consist of equations, which must be handled apart from
682  * the regular text.
683  * Thus, use this function to step through a line checking if it has any
684  * equations embedded in it.
685  * This must handle multiple equations AND equations that do not end at
686  * the end-of-line, i.e., will re-enter in the next roff parse.
687  */
688 static int
689 mdoc_preptext(struct mdoc *mdoc, int line, char *buf, int offs)
690 mdoc_preptext(struct mdoc *m, int line, char *buf, int offs)

```

```

690 {
691     char          *start, *end;
692     char          delim;

694     while ('\0' != buf[offs]) {
695         /* Mark starting position if eqn is set. */
696         start = NULL;
697         if ('\0' != (delim = roff_eqndelim(mdoc->roff)))
698             if ('\0' != (delim = roff_eqndelim(m->roff)))
699                 if (NULL != (start = strchr(buf + offs, delim)))
700                     *start++ = '\0';

701         /* Parse text as normal. */
702         if (! mdoc_ptext(mdoc, line, buf, offs))
703             if (! mdoc_ptext(m, line, buf, offs))
704                 return(0);

705         /* Continue only if an equation exists. */
706         if (NULL == start)
707             break;

709         /* Read past the end of the equation. */
710         offs += start - (buf + offs);
711         assert(start == &buf[offs]);
712         if (NULL != (end = strchr(buf + offs, delim))) {
713             *end++ = '\0';
714             while (' ' == *end)
715                 end++;
716         }

718         /* Parse the equation itself. */
719         roff_openeqn(mdoc->roff, NULL, line, offs, buf);
720         roff_openeqn(m->roff, NULL, line, offs, buf);

722         /* Process a finished equation? */
723         if (roff_closeeqn(mdoc->roff))
724             if (! mdoc_addeqn(mdoc, roff_eqn(mdoc->roff)))
725                 if (roff_closeeqn(m->roff))
726                     if (! mdoc_addeqn(m, roff_eqn(m->roff)))
727                         return(0);
728         offs += (end - (buf + offs));
729     }

730     return(1);
731 }
732 #endif

733 /*
734  * Parse free-form text, that is, a line that does not begin with the
735  * control character.
736  */
737 static int
738 mdoc_ptext(struct mdoc *mdoc, int line, char *buf, int offs)
739 mdoc_ptext(struct mdoc *m, int line, char *buf, int offs)
740 {
741     char          *c, *ws, *end;
742     struct mdoc_node *n;

744     /* No text before an initial macro. */

746     if (SEC_NONE == mdoc->lastnamed) {
747         mdoc_pmsg(mdoc, line, offs, MANDOCERR_NOTEXT);
748         if (SEC_NONE == m->lastnamed) {
749             mdoc_pmsg(m, line, offs, MANDOCERR_NOTEXT);
750             return(1);
751         }
752     }

```

```

749  assert(mdoc->last);
750  n = mdoc->last;
721  assert(m->last);
722  n = m->last;

752  /*
753  * Divert directly to list processing if we're encountering a
754  * columnar MDOC_BLOCK with or without a prior MDOC_BLOCK entry
755  * (a MDOC_BODY means it's already open, in which case we should
756  * process within its context in the normal way).
757  */

759  if (MDOC_Bl == n->tok && MDOC_BODY == n->type &&
760      LIST_column == n->norm->Bl.type) {
761      /* 'Bl' is open without any children. */
762      mdoc->flags |= MDOC_FREECOL;
763      return(mdoc_macro(mdoc, MDOC_It, line, offs, &offs, buf));
764      m->flags |= MDOC_FREECOL;
765      return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
766  }

766  if (MDOC_It == n->tok && MDOC_BLOCK == n->type &&
767      NULL != n->parent &&
768      MDOC_Bl == n->parent->tok &&
769      LIST_column == n->parent->norm->Bl.type) {
770      /* 'Bl' has block-level 'It' children. */
771      mdoc->flags |= MDOC_FREECOL;
772      return(mdoc_macro(mdoc, MDOC_It, line, offs, &offs, buf));
773      m->flags |= MDOC_FREECOL;
774      return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
775  }

775  /*
776  * Search for the beginning of unescaped trailing whitespace (ws)
777  * and for the first character not to be output (end).
778  */

780  /* FIXME: replace with strcspn(). */
781  ws = NULL;
782  for (c = end = buf + offs; *c; c++) {
783      switch (*c) {
784          case ' ':
785              if (NULL == ws)
786                  ws = c;
787              continue;
788          case '\t':
789              /*
790               * Always warn about trailing tabs,
791               * even outside literal context,
792               * where they should be put on the next line.
793               */
794              if (NULL == ws)
795                  ws = c;
796              /*
797               * Strip trailing tabs in literal context only;
798               * outside, they affect the next line.
799               */
800              if (MDOC_LITERAL & mdoc->flags)
801                  if (MDOC_LITERAL & m->flags)
802                      continue;
803              break;
804          case '\\':
805              /* Skip the escaped character, too, if any. */
806              if (c[1])
807                  c++;

```

```

807      /* FALLTHROUGH */
808      default:
809          ws = NULL;
810          break;
811      }
812      end = c + 1;
813  }
814  *end = '\0';

816  if (ws)
817      mdoc_pmsg(mdoc, line, (int)(ws-buf), MANDOCERR_EOLNSPACE);
789      mdoc_pmsg(m, line, (int)(ws-buf), MANDOCERR_EOLNSPACE);

819  if ('\0' == buf[offs] && ! (MDOC_LITERAL & mdoc->flags)) {
820      mdoc_pmsg(mdoc, line, (int)(c-buf), MANDOCERR_NOBLANKLN);
791      if ('\0' == buf[offs] && ! (MDOC_LITERAL & m->flags)) {
792      mdoc_pmsg(m, line, (int)(c-buf), MANDOCERR_NOBLANKLN);

822  /*
823  * Insert a 'sp' in the case of a blank line. Technically,
824  * blank lines aren't allowed, but enough manuals assume this
825  * behaviour that we want to work around it.
826  */
827  if ( ! mdoc_elem_alloc(mdoc, line, offs, MDOC_sp, NULL))
799  if ( ! mdoc_elem_alloc(m, line, offs, MDOC_sp, NULL))
828      return(0);

830  mdoc->next = MDOC_NEXT_SIBLING;

832  return(mdoc_valid_post(mdoc));
802  m->next = MDOC_NEXT_SIBLING;
803  return(1);
833  }

835  if ( ! mdoc_word_alloc(mdoc, line, offs, buf+offs))
806  if ( ! mdoc_word_alloc(m, line, offs, buf+offs))
836      return(0);

838  if (MDOC_LITERAL & mdoc->flags)
809  if (MDOC_LITERAL & m->flags)
839      return(1);

841  /*
842  * End-of-sentence check. If the last character is an unescaped
843  * EOS character, then flag the node as being the end of a
844  * sentence. The front-end will know how to interpret this.
845  */

847  assert(buf < end);

849  if (mandoc_eos(buf+offs, (size_t)(end-buf-offs), 0))
850      mdoc->last->flags |= MDOC_EOS;
821  m->last->flags |= MDOC_EOS;

852  return(1);
853  }

856  /*
857  * Parse a macro line, that is, a line beginning with the control
858  * character.
859  */
860  static int
861  mdoc_pmacro(struct mdoc *mdoc, int ln, char *buf, int offs)
832  mdoc_pmacro(struct mdoc *m, int ln, char *buf, int offs)
862  {

```

```

863     enum mdoct      tok;
864     int             i, sv;
865     char            mac[5];
866     struct mdoc_node *n;

868     /* Empty post-control lines are ignored. */

870     if ('"' == buf[offs]) {
871         mdoc_pmsg(mdoc, ln, offs, MANDOCERR_BADCOMMENT);
872         mdoc_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
873         return(1);
874     } else if ('\0' == buf[offs])
875         return(1);

876     sv = offs;

878     /*
879     * Copy the first word into a nil-terminated buffer.
880     * Stop copying when a tab, space, or eoln is encountered.
881     */

883     i = 0;
884     while (i < 4 && '\0' != buf[offs] &&
885           ' ' != buf[offs] && '\t' != buf[offs])
886         mac[i++] = buf[offs++];

888     mac[i] = '\0';

890     tok = (i > 1 || i < 4) ? mdoc_hash_find(mac) : MDOC_MAX;

892     if (MDOC_MAX == tok) {
893         mandoc_vmsg(MANDOCERR_MACRO, mdoc->parse,
894                   mandoc_vmsg(MANDOCERR_MACRO, m->parse,
895                               ln, sv, "%s", buf + sv - 1));
896     }

898     /* Disregard the first trailing tab, if applicable. */

900     if ('\t' == buf[offs])
901         offs++;

903     /* Jump to the next non-whitespace word. */

905     while (buf[offs] && ' ' == buf[offs])
906         offs++;

908     /*
909     * Trailing whitespace. Note that tabs are allowed to be passed
910     * into the parser as "text", so we only warn about spaces here.
911     */

913     if ('\0' == buf[offs] && ' ' == buf[offs - 1])
914         mdoc_pmsg(mdoc, ln, offs - 1, MANDOCERR_EOLNSPACE);
915         mdoc_pmsg(m, ln, offs - 1, MANDOCERR_EOLNSPACE);

916     /*
917     * If an initial macro or a list invocation, divert directly
918     * into macro processing.
919     */

921     if (NULL == mdoc->last || MDOC_It == tok || MDOC_El == tok) {
922         if (! mdoc_macro(mdoc, tok, ln, sv, &offs, buf))
923             if (NULL == m->last || MDOC_It == tok || MDOC_El == tok) {
924                 if (! mdoc_macro(m, tok, ln, sv, &offs, buf))
925                     goto err;

```

```

924         return(1);
925     }

927     n = mdoc->last;
928     assert(mdoc->last);
929     n = m->last;
930     assert(m->last);

931     /*
932     * If the first macro of a 'Bl -column', open an 'It' block
933     * context around the parsed macro.
934     */

935     if (MDOC_Bl == n->tok && MDOC_BODY == n->type &&
936         LIST_column == n->norm->Bl.type) {
937         mdoc->flags |= MDOC_FREECOL;
938         if (! mdoc_macro(mdoc, MDOC_It, ln, sv, &sv, buf))
939             m->flags |= MDOC_FREECOL;
940         if (! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
941             goto err;
942         return(1);
943     }

944     /*
945     * If we're following a block-level 'It' within a 'Bl -column'
946     * context (perhaps opened in the above block or in ptext()),
947     * then open an 'It' block context around the parsed macro.
948     */

949     if (MDOC_It == n->tok && MDOC_BLOCK == n->type &&
950         NULL != n->parent &&
951         MDOC_Bl == n->parent->tok &&
952         LIST_column == n->parent->norm->Bl.type) {
953         mdoc->flags |= MDOC_FREECOL;
954         if (! mdoc_macro(mdoc, MDOC_It, ln, sv, &sv, buf))
955             m->flags |= MDOC_FREECOL;
956         if (! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
957             goto err;
958         return(1);
959     }

960     /* Normal processing of a macro. */

961     if (! mdoc_macro(mdoc, tok, ln, sv, &offs, buf))
962         if (! mdoc_macro(m, tok, ln, sv, &offs, buf))
963             goto err;

964     return(1);

966 err:    /* Error out. */

968     mdoc->flags |= MDOC_HALT;
969     m->flags |= MDOC_HALT;
970     return(0);
971 }

972 enum mdelim
973 mdoc_isdelim(const char *p)
974 {

976     if ('\0' == p[0])
977         return(DELMIM_NONE);

979     if ('\0' == p[1])
980         switch (p[0]) {
981             case(''):

```

```
982     /* FALLTHROUGH */
983     case '[':
984         return(DELIM_OPEN);
985     case '|':
986         return(DELIM_MIDDLE);
987     case '.':
988         /* FALLTHROUGH */
989     case ',':
990         /* FALLTHROUGH */
991     case ';':
992         /* FALLTHROUGH */
993     case ':':
994         /* FALLTHROUGH */
995     case '?':
996         /* FALLTHROUGH */
997     case '!':
998         /* FALLTHROUGH */
999     case ')':
1000         /* FALLTHROUGH */
1001     case ']':
1002         return(DELIM_CLOSE);
1003     default:
1004         return(DELIM_NONE);
1005     }
1007     if ('\\" != p[0])
1008         return(DELIM_NONE);
1010     if (0 == strcmp(p + 1, "."))
1011         return(DELIM_CLOSE);
1012     if (0 == strcmp(p + 1, "fR|\\fP"))
1013         return(DELIM_MIDDLE);
1015     return(DELIM_NONE);
1016 }
unchanged_portion_omitted
```

\*\*\*\*\*

8568 Fri Jul 25 15:00:13 2014

new/usr/src/cmd/mandoc/mdoc.h

Update to 1.12.3.

\*\*\*\*\*

```

1 /* $Id: mdoc.h,v 1.125 2013/12/24 19:11:45 schwarze Exp $ */
1 /* $Id: mdoc.h,v 1.122 2011/03/22 14:05:45 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifndef MDOC_H
18 #define MDOC_H

20 enum mdoct {
21     MDOC_Ap = 0,
22     MDOC_Dd,
23     MDOC_Dt,
24     MDOC_Os,
25     MDOC_Sh,
26     MDOC_Ss,
27     MDOC_Pp,
28     MDOC_Dl,
29     MDOC_Dl,
30     MDOC_Bd,
31     MDOC_Ed,
32     MDOC_Bl,
33     MDOC_El,
34     MDOC_It,
35     MDOC_Ad,
36     MDOC_An,
37     MDOC_Ar,
38     MDOC_Cd,
39     MDOC_Cm,
40     MDOC_Dv,
41     MDOC_Er,
42     MDOC_Ev,
43     MDOC_Ex,
44     MDOC_Fa,
45     MDOC_Fd,
46     MDOC_Fl,
47     MDOC_Fn,
48     MDOC_Ft,
49     MDOC_Ic,
50     MDOC_In,
51     MDOC_Li,
52     MDOC_Nd,
53     MDOC_Nm,
54     MDOC_Op,
55     MDOC_Ot,
56     MDOC_Pa,
57     MDOC_Rv,
58     MDOC_St,
59     MDOC_Va,
60     MDOC_Vt,

```

```

61     MDOC_Xr,
62     MDOC_A,
63     MDOC_B,
64     MDOC_D,
65     MDOC_I,
66     MDOC_J,
67     MDOC_N,
68     MDOC_O,
69     MDOC_P,
70     MDOC_R,
71     MDOC_T,
72     MDOC_V,
73     MDOC_Ac,
74     MDOC_Ao,
75     MDOC_Aq,
76     MDOC_At,
77     MDOC_Bc,
78     MDOC_Bf,
79     MDOC_Bo,
80     MDOC_Bq,
81     MDOC_Bsx,
82     MDOC_Bx,
83     MDOC_Db,
84     MDOC_Dc,
85     MDOC_Do,
86     MDOC_Dq,
87     MDOC_Ec,
88     MDOC_Ef,
89     MDOC_Em,
90     MDOC_Eo,
91     MDOC_Fx,
92     MDOC_Ms,
93     MDOC_No,
94     MDOC_Ns,
95     MDOC_Nx,
96     MDOC_Ox,
97     MDOC_Pc,
98     MDOC_Pf,
99     MDOC_Po,
100    MDOC_Pq,
101    MDOC_Qc,
102    MDOC_Ql,
103    MDOC_Qo,
104    MDOC_Qq,
105    MDOC_Re,
106    MDOC_Rs,
107    MDOC_Sc,
108    MDOC_So,
109    MDOC_Sq,
110    MDOC_Sm,
111    MDOC_Sx,
112    MDOC_Sy,
113    MDOC_Tn,
114    MDOC_Ux,
115    MDOC_Xc,
116    MDOC_Xo,
117    MDOC_Fo,
118    MDOC_Fc,
119    MDOC_Oo,
120    MDOC_Oc,
121    MDOC_Bk,
122    MDOC_Ek,
123    MDOC_Bt,
124    MDOC_Hf,
125    MDOC_Fr,
126    MDOC_Ud,

```

```

127     MDOC_Lb,
128     MDOC_Lp,
129     MDOC_Lk,
130     MDOC_Mt,
131     MDOC_Brq,
132     MDOC_Bro,
133     MDOC_Brc,
134     MDOC__C,
135     MDOC_Es,
136     MDOC_En,
137     MDOC_Dx,
138     MDOC__Q,
139     MDOC_br,
140     MDOC_sp,
141     MDOC__U,
142     MDOC_Ta,
143     MDOC_MAX
144 };
    unchanged_portion_omitted

```

```

307 struct mdoc_bl {
308     const char    *width; /* -width */
309     const char    *offs; /* -offset */
310     enum mdoc_list type; /* -tag, -enum, etc. */
311     int           comp; /* -compact */
312     size_t        ncols; /* -column arg count */
313     const char    **cols; /* -column val ptr */
314     int           count; /* -enum counter */
315 };
    unchanged_portion_omitted

```

```

342 /*
343  * Single node in tree-linked AST.
344  */
345 struct mdoc_node {
346     struct mdoc_node *parent; /* parent AST node */
347     struct mdoc_node *child; /* first child AST node */
348     struct mdoc_node *last; /* last child AST node */
349     struct mdoc_node *next; /* sibling AST node */
350     struct mdoc_node *prev; /* prior sibling AST node */
351     int               nchild; /* number children */
352     int               line; /* parse line */
353     int               pos; /* parse column */
354     int               lastline; /* the node ends on this line */
355     enum mdoct        tok; /* tok or MDOC_MAX if none */
356     int               flags;
357     #define MDOC_VALID (1 << 0) /* has been validated */
358     #define MDOC_EOS (1 << 2) /* at sentence boundary */
359     #define MDOC_LINE (1 << 3) /* first macro/text on line */
360     #define MDOC_SYNPRETTY (1 << 4) /* SYNOPSIS-style formatting */
361     #define MDOC_ENDED (1 << 5) /* rendering has been ended */
362     #define MDOC_DELIMO (1 << 6)
363     #define MDOC_DELIMC (1 << 7)
364     enum mdoc_type    type; /* AST node type */
365     enum mdoc_sec      sec; /* current named section */
366     union mdoc_data    *norm; /* normalised args */
367     const void         *prev_font; /* before entering this node */
368     /* FIXME: these can be union'd to shave a few bytes. */
369     struct mdoc_arg    *args; /* BLOCK/ELEM */
370     struct mdoc_node   *pending; /* BLOCK */
371     struct mdoc_node   *head; /* BLOCK */
372     struct mdoc_node   *body; /* BLOCK */
373     struct mdoc_node   *tail; /* BLOCK */
374     char               *string; /* TEXT */
375     const struct tbl_span *span; /* TBL */
376     const struct eqn    *eqn; /* EQN */

```

```

377         enum mdoc_endbody end; /* BODY */
378     };
    unchanged_portion_omitted

```

```

*****
16856 Fri Jul 25 15:00:13 2014
new/usr/src/cmd/mandoc/mdoc_argv.c
Update to 1.12.3.
*****
1 /* $Id: mdoc_argv.c,v 1.89 2013/12/25 00:50:05 schwarze Exp $ */
1 /* $Id: mdoc_argv.c,v 1.82 2012/03/23 05:50:24 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2012 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <stdlib.h>
26 #include <stdio.h>
27 #include <string.h>
28
29 #include "mdoc.h"
30 #include "mandoc.h"
31 #include "libmdoc.h"
32 #include "libmandoc.h"
33
34 #define MULTI_STEP      5 /* pre-allocate argument values */
35 #define DELIMSZ        6 /* max possible size of a delimiter */
36
37 enum      argsflag {
38     ARGSFL_NONE = 0,
39     ARGSFL_DELIM, /* handle delimiters of [[:delim:]]+ */
40     ARGSFL_TABSEP /* handle tab/'Ta' separated phrases */
41 };
42
43 enum      argvflag {
44     ARGV_NONE, /* no args to flag (e.g., -split) */
45     ARGV_SINGLE, /* one arg to flag (e.g., -file xxx) */
46     ARGV_MULTI /* multiple args (e.g., -column xxx yyy) */
47 };
48
49 #define ARGV_SINGLE /* multiple args (e.g., -column xxx yyy) */
50 #define ARGV_OPT_SINGLE /* optional arg (e.g., -offset [xxx]) */
51
52 #ifndef unchanged_portion_omitted
53
54 static void      argn_free(struct mdoc_arg *, int);
55 static enum margserr      args(struct mdoc *, int, int *,
56                               char *, enum argsflag, char **);
57 static int      args_checkpoint(const char *, int);
58 static int      argv_multi(struct mdoc *, int,
59                            struct mdoc_argv *, int *, char *);
60 static int      argv_opt_single(struct mdoc *, int,
61                                 struct mdoc_argv *, int *, char *);
62 static int      argv_single(struct mdoc *, int,

```

```

63 struct mdoc_argv *, int *, char *);
64
65 static const enum argvflag argvflags[MDOC_ARG_MAX] = {
66     ARGV_NONE, /* MDOC_Split */
67     ARGV_NONE, /* MDOC_Nosplit */
68     ARGV_NONE, /* MDOC_Ragged */
69     ARGV_NONE, /* MDOC_Unfilled */
70     ARGV_SINGLE, /* MDOC_Literal */
71     ARGV_SINGLE, /* MDOC_File */
72     ARGV_SINGLE, /* MDOC_Offset */
73     ARGV_OPT_SINGLE, /* MDOC_Offset */
74     ARGV_NONE, /* MDOC_Bullet */
75     ARGV_NONE, /* MDOC_Dash */
76     ARGV_NONE, /* MDOC_Hyphen */
77     ARGV_NONE, /* MDOC_Item */
78     ARGV_NONE, /* MDOC_Enum */
79     ARGV_NONE, /* MDOC_Tag */
80     ARGV_NONE, /* MDOC_Diag */
81     ARGV_NONE, /* MDOC_Hang */
82     ARGV_NONE, /* MDOC_Ohang */
83     ARGV_NONE, /* MDOC_Inset */
84     ARGV_MULT, /* MDOC_Column */
85     ARGV_SINGLE, /* MDOC_Width */
86     ARGV_OPT_SINGLE, /* MDOC_Width */
87     ARGV_NONE, /* MDOC_Compact */
88     ARGV_NONE, /* MDOC_Std */
89     ARGV_NONE, /* MDOC_Filled */
90     ARGV_NONE, /* MDOC_Words */
91     ARGV_NONE, /* MDOC_Emphasis */
92     ARGV_NONE, /* MDOC_Symbolic */
93     ARGV_NONE, /* MDOC_Symbolic */
94 };
95
96 #ifndef unchanged_portion_omitted
97
98 static const struct mdocarg mdocargs[MDOC_MAX] = {
99     { ARGV_SINGLE, NULL }, /* Ap */
100    { ARGV_SINGLE, NULL }, /* Ap */
101    { ARGV_SINGLE, NULL }, /* Dd */
102    { ARGV_SINGLE, NULL }, /* Dt */
103    { ARGV_SINGLE, NULL }, /* Os */
104    { ARGV_SINGLE, NULL }, /* Sh */
105    { ARGV_SINGLE, NULL }, /* Ss */
106    { ARGV_SINGLE, NULL }, /* Pp */
107    { ARGV_SINGLE, NULL }, /* D1 */
108    { ARGV_SINGLE, NULL }, /* D1 */
109    { ARGV_SINGLE, args_Bd }, /* Bd */
110    { ARGV_SINGLE, NULL }, /* Ed */
111    { ARGV_SINGLE, args_Bl }, /* Bl */
112    { ARGV_SINGLE, NULL }, /* El */
113    { ARGV_SINGLE, NULL }, /* It */
114    { ARGV_SINGLE, NULL }, /* Ad */
115    { ARGV_SINGLE, args_An }, /* An */
116    { ARGV_SINGLE, NULL }, /* Ar */
117    { ARGV_SINGLE, NULL }, /* Cd */
118    { ARGV_SINGLE, NULL }, /* Cd */
119    { ARGV_SINGLE, NULL }, /* Cm */
120    { ARGV_SINGLE, NULL }, /* Dv */
121    { ARGV_SINGLE, NULL }, /* Er */
122    { ARGV_SINGLE, NULL }, /* Ev */
123    { ARGV_SINGLE, args_Ex }, /* Ex */
124    { ARGV_SINGLE, NULL }, /* Fa */
125    { ARGV_SINGLE, NULL }, /* Fd */
126    { ARGV_SINGLE, NULL }, /* Fl */
127    { ARGV_SINGLE, NULL }, /* Fn */
128    { ARGV_SINGLE, NULL }, /* Ft */
129    { ARGV_SINGLE, NULL }, /* Ic */

```

```

176 { ARGSFL_DELIM, NULL }, /* In */
177 { ARGSFL_NONE, NULL }, /* In */
178 { ARGSFL_DELIM, NULL }, /* Li */
179 { ARGSFL_NONE, NULL }, /* Nd */
180 { ARGSFL_DELIM, NULL }, /* Nm */
181 { ARGSFL_NONE, NULL }, /* Op */
182 { ARGSFL_DELIM, NULL }, /* Ot */
183 { ARGSFL_NONE, args_Ex }, /* Rv */
184 { ARGSFL_DELIM, NULL }, /* St */
185 { ARGSFL_DELIM, NULL }, /* Va */
186 { ARGSFL_DELIM, NULL }, /* Vt */
187 { ARGSFL_DELIM, NULL }, /* Xr */
188 { ARGSFL_NONE, NULL }, /* %A */
189 { ARGSFL_NONE, NULL }, /* %B */
190 { ARGSFL_NONE, NULL }, /* %D */
191 { ARGSFL_NONE, NULL }, /* %I */
192 { ARGSFL_NONE, NULL }, /* %J */
193 { ARGSFL_NONE, NULL }, /* %N */
194 { ARGSFL_NONE, NULL }, /* %O */
195 { ARGSFL_NONE, NULL }, /* %P */
196 { ARGSFL_NONE, NULL }, /* %R */
197 { ARGSFL_NONE, NULL }, /* %T */
198 { ARGSFL_NONE, NULL }, /* %V */
199 { ARGSFL_DELIM, NULL }, /* Ac */
200 { ARGSFL_NONE, NULL }, /* Ao */
201 { ARGSFL_DELIM, NULL }, /* Aq */
202 { ARGSFL_DELIM, NULL }, /* At */
203 { ARGSFL_DELIM, NULL }, /* Bc */
204 { ARGSFL_NONE, args_Bf }, /* Bf */
205 { ARGSFL_NONE, NULL }, /* Bo */
206 { ARGSFL_DELIM, NULL }, /* Bq */
207 { ARGSFL_DELIM, NULL }, /* Bsx */
208 { ARGSFL_DELIM, NULL }, /* Bx */
209 { ARGSFL_NONE, NULL }, /* Db */
210 { ARGSFL_DELIM, NULL }, /* Dc */
211 { ARGSFL_NONE, NULL }, /* Do */
212 { ARGSFL_DELIM, NULL }, /* Dq */
213 { ARGSFL_DELIM, NULL }, /* Ec */
214 { ARGSFL_NONE, NULL }, /* Ef */
215 { ARGSFL_DELIM, NULL }, /* Em */
216 { ARGSFL_NONE, NULL }, /* Eo */
217 { ARGSFL_DELIM, NULL }, /* Fx */
218 { ARGSFL_DELIM, NULL }, /* Ms */
219 { ARGSFL_DELIM, NULL }, /* No */
220 { ARGSFL_DELIM, NULL }, /* Ns */
221 { ARGSFL_DELIM, NULL }, /* Nx */
222 { ARGSFL_DELIM, NULL }, /* Ox */
223 { ARGSFL_DELIM, NULL }, /* Pc */
224 { ARGSFL_DELIM, NULL }, /* Pf */
225 { ARGSFL_NONE, NULL }, /* Po */
226 { ARGSFL_DELIM, NULL }, /* Pq */
227 { ARGSFL_DELIM, NULL }, /* Qc */
228 { ARGSFL_DELIM, NULL }, /* Ql */
229 { ARGSFL_NONE, NULL }, /* Qo */
230 { ARGSFL_DELIM, NULL }, /* Qq */
231 { ARGSFL_NONE, NULL }, /* Re */
232 { ARGSFL_NONE, NULL }, /* Rs */
233 { ARGSFL_DELIM, NULL }, /* Sc */
234 { ARGSFL_NONE, NULL }, /* So */
235 { ARGSFL_DELIM, NULL }, /* Sq */
236 { ARGSFL_NONE, NULL }, /* Sm */
237 { ARGSFL_DELIM, NULL }, /* Sx */
238 { ARGSFL_DELIM, NULL }, /* Sy */
239 { ARGSFL_DELIM, NULL }, /* Tn */
240 { ARGSFL_DELIM, NULL }, /* Ux */

```

```

241 { ARGSFL_DELIM, NULL }, /* Xc */
242 { ARGSFL_NONE, NULL }, /* Xo */
243 { ARGSFL_NONE, NULL }, /* Fo */
244 { ARGSFL_DELIM, NULL }, /* Fc */
245 { ARGSFL_NONE, NULL }, /* Fc */
246 { ARGSFL_DELIM, NULL }, /* Oc */
247 { ARGSFL_NONE, args_Bk }, /* Bk */
248 { ARGSFL_NONE, NULL }, /* Ek */
249 { ARGSFL_NONE, NULL }, /* Bt */
250 { ARGSFL_NONE, NULL }, /* Hf */
251 { ARGSFL_NONE, NULL }, /* Fr */
252 { ARGSFL_NONE, NULL }, /* Ud */
253 { ARGSFL_DELIM, NULL }, /* Lb */
254 { ARGSFL_NONE, NULL }, /* Lb */
255 { ARGSFL_DELIM, NULL }, /* Lk */
256 { ARGSFL_DELIM, NULL }, /* Mt */
257 { ARGSFL_DELIM, NULL }, /* Brq */
258 { ARGSFL_NONE, NULL }, /* Bro */
259 { ARGSFL_DELIM, NULL }, /* Brc */
260 { ARGSFL_NONE, NULL }, /* %C */
261 { ARGSFL_NONE, NULL }, /* Es */
262 { ARGSFL_NONE, NULL }, /* En */
263 { ARGSFL_DELIM, NULL }, /* Dx */
264 { ARGSFL_NONE, NULL }, /* Dx */
265 { ARGSFL_NONE, NULL }, /* %Q */
266 { ARGSFL_NONE, NULL }, /* br */
267 { ARGSFL_NONE, NULL }, /* sp */
268 { ARGSFL_NONE, NULL }, /* %U */
269 };

```

```

272 /*
273  * Parse an argument from line text. This comes in the form of -key
274  * [value0...], which may either have a single mandatory value, at least
275  * one mandatory value, an optional single value, or no value.
276  */
277 enum margverr
278 mdoc_argv(struct mdoc *mdoc, int line, enum mdoct tok,
279 mdoc_argv(struct mdoc *m, int line, enum mdoct tok,
280 struct mdoc_arg **v, int *pos, char *buf)
281 {
282     char *p, sv;
283     struct mdoc_argv tmp;
284     struct mdoc_arg *arg;
285     const enum mdocargt *ap;
286
287     if ('\0' == buf[*pos])
288         return(ARGV_EOLN);
289     else if (NULL == (ap = mdocargs[tok].argsv))
290         return(ARGV_WORD);
291     else if ('-' != buf[*pos])
292         return(ARGV_WORD);
293
294     /* Seek to the first unescaped space. */
295
296     p = &buf[+(*pos)];
297
298     assert(*pos > 0);
299
300     for ( ; buf[*pos] ; (*pos)++)
301         if (' ' == buf[*pos] && '\\\'' != buf[*pos - 1])
302             break;

```



```

303  /*
304  * We want to nil-terminate the word to look it up (it's easier
305  * that way). But we may not have a flag, in which case we need
306  * to restore the line as-is. So keep around the stray byte,
307  * which we'll reset upon exiting (if necessary).
308  */
310  if ('\0' != (sv = buf[*pos]))
311      buf[(*pos)++] = '\0';
313  /*
314  * Now look up the word as a flag. Use temporary storage that
315  * we'll copy into the node's flags, if necessary.
316  */
318  memset(&tmp, 0, sizeof(struct mdoc_argv));
320  tmp.line = line;
321  tmp.pos = *pos;
322  tmp.arg = MDOC_ARG_MAX;
324  while (MDOC_ARG_MAX != (tmp.arg = *ap++))
325      if (0 == strcmp(p, mdoc_argnames[tmp.arg]))
326          break;
328  if (MDOC_ARG_MAX == tmp.arg) {
329      /*
330      * The flag was not found.
331      * Restore saved zeroed byte and return as a word.
332      */
333      if (sv)
334          buf[*pos - 1] = sv;
335      return(ARGV_WORD);
336  }
338  /* Read to the next word (the argument). */
340  while (buf[*pos] && ' ' == buf[*pos])
341      (*pos)++;
343  switch (argvflags[tmp.arg]) {
344  case (ARGV_SINGLE):
345      if ( ! argv_single(mdoc, line, &tmp, pos, buf))
346          if ( ! argv_single(m, line, &tmp, pos, buf))
347              return(ARGV_ERROR);
348      break;
349  case (ARGV_MULTII):
350      if ( ! argv_multi(mdoc, line, &tmp, pos, buf))
351          if ( ! argv_multi(m, line, &tmp, pos, buf))
352              return(ARGV_ERROR);
353      break;
354  case (ARGV_OPT_SINGLE):
355      if ( ! argv_opt_single(m, line, &tmp, pos, buf))
356          return(ARGV_ERROR);
357      break;
358  case (ARGV_NONE):
359      break;
360  }
362  if (NULL == (arg = *v))
363      arg = *v = mandoc_calloc(1, sizeof(struct mdoc_arg));
365  arg->argc++;
366  arg->argv = mandoc_realloc
367      (arg->argv, arg->argc * sizeof(struct mdoc_arg));

```

```

363  memcpy(&arg->argv[(int)arg->argc - 1],
364         &tmp, sizeof(struct mdoc_argv));
366  return(ARGV_ARG);
367 }
369 unchanged_portion_omitted
371 enum margserr
372 mdoc_args(struct mdoc *mdoc, int line, int *pos, char *buf, char **v)
373 mdoc_zargs(struct mdoc *m, int line, int *pos, char *buf, char **v)
374 {
375     return(args(mdoc, line, pos, buf, ARGSFL_NONE, v));
376     return(args(m, line, pos, buf, ARGSFL_NONE, v));
377 }
379 enum margserr
380 mdoc_args(struct mdoc *mdoc, int line, int *pos,
381           mdoc_zargs(struct mdoc *m, int line, int *pos,
382                     char *buf, enum mdoct tok, char **v)
383           {
384             enum argsflag fl;
385             struct mdoc_node *n;
386             fl = mdocargs[tok].flags;
387             if (MDOC_It != tok)
388                 return(args(mdoc, line, pos, buf, fl, v));
389                 return(args(m, line, pos, buf, fl, v));
390             /*
391             * We know that we're in an 'It', so it's reasonable to expect
392             * us to be sitting in a 'Bl'. Someday this may not be the case
393             * (if we allow random 'It's sitting out there), so provide a
394             * safe fall-back into the default behaviour.
395             */
396             for (n = mdoc->last; n; n = n->parent)
397                 for (n = m->last; n; n = n->parent)
398                     if (MDOC_Bl == n->tok)
399                         if (LIST_column == n->norm->Bl.type) {
400                             fl = ARGSFL_TABSEP;
401                             break;
402                         }
403             return(args(mdoc, line, pos, buf, fl, v));
404             return(args(m, line, pos, buf, fl, v));
405         }
406 static enum margserr
407 args(struct mdoc *mdoc, int line, int *pos,
408       mdoc_zargs(struct mdoc *m, int line, int *pos,
409                 char *buf, enum argsflag fl, char **v)
410       {
411         char *p, *pp;
412         int pairs;
413         enum margserr rc;
414         if ('\0' == buf[*pos]) {
415             if (MDOC_PPHRASE & mdoc->flags)
416                 if (MDOC_PPHRASE & m->flags)
417                     return(ARGS_EOLN);
418             /*
419             * If we're not in a partial phrase and the flag for
420             * being a phrase literal is still set, the punctuation
421             * is unterminated.

```

```

460      */
461      if (MDOC_PHRASELIT & mdoc->flags)
462          mdoc_pmsg(mdoc, line, *pos, MANDOCERR_BADQUOTE);
466      if (MDOC_PHRASELIT & m->flags)
467          mdoc_pmsg(m, line, *pos, MANDOCERR_BADQUOTE);

464      mdoc->flags &= ~MDOC_PHRASELIT;
469      m->flags &= ~MDOC_PHRASELIT;
465      return(ARGS_EOLN);
466  }

468  *v = &buf[*pos];

470  if (ARGSFL_DELIM == fl)
471      if (args_checkpunct(buf, *pos))
472          return(ARGS_PUNCT);

474  /*
475  * First handle TABSEP items, restricted to 'B1 -column'. This
476  * ignores conventional token parsing and instead uses tabs or
477  * 'Ta' macros to separate phrases. Phrases are parsed again
478  * for arguments at a later phase.
479  */

481  if (ARGSFL_TABSEP == fl) {
482      /* Scan ahead to tab (can't be escaped). */
483      p = strchr(*v, '\t');
484      pp = NULL;

486      /* Scan ahead to unescaped 'Ta'. */
487      if ( ! (MDOC_PHRASELIT & mdoc->flags) )
488          if ( ! (MDOC_PHRASELIT & m->flags) )
489              for (pp = *v; ; pp++) {
490                  if (NULL == (pp = strstr(pp, "Ta")))
491                      break;
492                  if (pp > *v && ' ' != *(pp - 1))
493                      continue;
494                  if (' ' == *(pp + 2) || '\0' == *(pp + 2))
495                      break;
496              }

497      /* By default, assume a phrase. */
498      rc = ARGS_PHRASE;

500      /*
501      * Adjust new-buffer position to be beyond delimiter
502      * mark (e.g., Ta -> end + 2).
503      */
504      if (p && pp) {
505          *pos += pp < p ? 2 : 1;
506          rc = pp < p ? ARGS_PHRASE : ARGS_PPHRASE;
507          p = pp < p ? pp : p;
508      } else if (p && !pp) {
509          rc = ARGS_PPHRASE;
510          *pos += 1;
511      } else if (pp && !p) {
512          p = pp;
513          *pos += 2;
514      } else {
515          rc = ARGS_PEND;
516          p = strchr(*v, 0);
517      }

519      /* Whitespace check for eoln case... */
520      if ('\0' == *p && ' ' == *(p - 1))
521          mdoc_pmsg(mdoc, line, *pos, MANDOCERR_EOLNSPACE);

```

```

526      mdoc_pmsg(m, line, *pos, MANDOCERR_EOLNSPACE);

523      *pos += (int)(p - *v);

525      /* Strip delimiter's preceding whitespace. */
526      pp = p - 1;
527      while (pp > *v && ' ' == *pp) {
528          if (pp > *v && '\\\'' == *(pp - 1))
529              break;
530          pp--;
531      }
532      *(pp + 1) = 0;

534      /* Strip delimiter's proceeding whitespace. */
535      for (pp = &buf[*pos]; ' ' == *pp; pp++, (*pos)++)
536          /* Skip ahead. */ ;

538      return(rc);
539  }

541  /*
542  * Process a quoted literal. A quote begins with a double-quote
543  * and ends with a double-quote NOT preceded by a double-quote.
544  * NUL-terminate the literal in place.
545  * Collapse pairs of quotes inside quoted literals.
546  * Whitespace is NOT involved in literal termination.
547  */

549  if (MDOC_PHRASELIT & mdoc->flags || '\"' == buf[*pos]) {
550      if ( ! (MDOC_PHRASELIT & mdoc->flags) )
552          if (MDOC_PHRASELIT & m->flags || '\"' == buf[*pos]) {
553              if ( ! (MDOC_PHRASELIT & m->flags) )
554                  *v = &buf[+(*pos)];

553          if (MDOC_PPHRASE & mdoc->flags)
554              mdoc->flags |= MDOC_PHRASELIT;
556          if (MDOC_PPHRASE & m->flags)
557              m->flags |= MDOC_PHRASELIT;

556      pairs = 0;
557      for ( ; buf[*pos]; (*pos)++) {
558          /* Move following text left after quoted quotes. */
559          if (pairs)
560              buf[*pos - pairs] = buf[*pos];
561          if ('\'' != buf[*pos])
562              continue;
563          /* Unquoted quotes end quoted args. */
564          if ('\'' != buf[*pos + 1])
565              break;
566          /* Quoted quotes collapse. */
567          pairs++;
568          (*pos)++;
569      }
570      if (pairs)
571          buf[*pos - pairs] = '\0';

573      if ('\0' == buf[*pos]) {
574          if (MDOC_PPHRASE & mdoc->flags)
575              if (MDOC_PPHRASE & m->flags)
576                  return(ARGS_QWORD);
577          mdoc_pmsg(mdoc, line, *pos, MANDOCERR_BADQUOTE);
578          return(ARGS_QWORD);
579      }

580      mdoc->flags &= ~MDOC_PHRASELIT;

```

```

574     m->flags &= ~MDOC_PHRASELIT;
581     buf[(*pos)++] = '\0';

583     if ('\0' == buf[*pos])
584         return(ARGS_QWORD);

586     while ( ' ' == buf[*pos])
587         (*pos)++;

589     if ('\0' == buf[*pos])
590         mdoc_pmsg(mdoc, line, *pos, MANDOCERR_EOLNSPACE);
584         mdoc_pmsg(m, line, *pos, MANDOCERR_EOLNSPACE);

592     return(ARGS_QWORD);
593 }

595     p = &buf[*pos];
596     *v = mandoc_getarg(mdoc->parse, &p, line, pos);
590     *v = mandoc_getarg(m->parse, &p, line, pos);

598     return(ARGS_WORD);
599 }

```

unchanged portion omitted

```

651 static int
652 argv_multi(struct mdoc *mdoc, int line,
646 argv_multi(struct mdoc *m, int line,
653             struct mdoc_argv *v, int *pos, char *buf)
654 {
655     enum margserr    ac;
656     char             *p;

658     for (v->sz = 0; ; v->sz++) {
659         if ('-' == buf[*pos])
660             break;
661         ac = args(mdoc, line, pos, buf, ARGSFL_NONE, &p);
655         ac = args(m, line, pos, buf, ARGSFL_NONE, &p);
662         if (ARGS_ERROR == ac)
663             return(0);
664         else if (ARGS_EOLN == ac)
665             break;

667         if (0 == v->sz % MULTI_STEP)
668             v->value = mandoc_realloc(v->value,
669                                     (v->sz + MULTI_STEP) * sizeof(char *));

671         v->value[(int)v->sz] = mandoc_strdup(p);
672     }

674     return(1);
675 }

677 static int
678 argv_single(struct mdoc *mdoc, int line,
672 argv_opt_single(struct mdoc *m, int line,
679                struct mdoc_argv *v, int *pos, char *buf)
680 {
681     enum margserr    ac;
682     char             *p;

684     ac = args(mdoc, line, pos, buf, ARGSFL_NONE, &p);
678     if ('-' == buf[*pos])
679         return(1);

681     ac = args(m, line, pos, buf, ARGSFL_NONE, &p);
685     if (ARGS_ERROR == ac)

```

```

686         return(0);
687     if (ARGS_EOLN == ac)
688         return(1);

690     v->sz = 1;
688     v->value = mandoc_malloc(sizeof(char *));
689     v->value[0] = mandoc_strdup(p);

691     return(1);
692 }

694 static int
695 argv_single(struct mdoc *m, int line,
696             struct mdoc_argv *v, int *pos, char *buf)
697 {
698     int             ppos;
699     enum margserr    ac;
700     char             *p;

702     ppos = *pos;

704     ac = args(m, line, pos, buf, ARGSFL_NONE, &p);
705     if (ARGS_EOLN == ac) {
706         mdoc_pmsg(m, line, ppos, MANDOCERR_SYNTARGVCOUNT);
707         return(0);
708     } else if (ARGS_ERROR == ac)
709         return(0);

711     v->sz = 1;
691     v->value = mandoc_malloc(sizeof(char *));
692     v->value[0] = mandoc_strdup(p);

694     return(1);
695 }

```

unchanged portion omitted

new/usr/src/cmd/mandoc/mdoc\_html.c

1

\*\*\*\*\*

43931 Fri Jul 25 15:00:14 2014

new/usr/src/cmd/mandoc/mdoc\_html.c

Update to 1.12.3.

\*\*\*\*\*

```
1 /* $Id: mdoc_html.c,v 1.186 2013/12/24 20:45:27 schwarze Exp $ */
1 /* $Id: mdoc_html.c,v 1.182 2011/11/03 20:37:00 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif
20
21 #include <sys/types.h>
22
23 #include <assert.h>
24 #include <ctype.h>
25 #include <stdio.h>
26 #include <stdlib.h>
27 #include <string.h>
28 #include <unistd.h>
29
30 #include "mandoc.h"
31 #include "out.h"
32 #include "html.h"
33 #include "mdoc.h"
34 #include "main.h"
35
36 #define INDENT 5
37
38 #define MDOC_ARGS const struct mdoc_meta *meta, \
39 #define MDOC_ARGS const struct mdoc_meta *m, \
40 #define MDOC_ARGS const struct mdoc_node *n, \
41 #define MDOC_ARGS struct html *h
42
43 #ifndef MIN
44 #define MIN(a,b) ((/*CONSTCOND*/(a)<(b))?(a):(b))
45 #endif
46
47 struct htmlmdoc {
48     int (*pre)(MDOC_ARGS);
49     void (*post)(MDOC_ARGS);
50 };
51
52 void
53 html_mdoc(void *arg, const struct mdoc *mdoc)
54 {
55     html_mdoc(void *arg, const struct mdoc *m)
56 {
57     print_mdoc(mdoc_meta(mdoc), mdoc_node(mdoc),
58                (struct html *)arg);
59     print_mdoc(mdoc_meta(m), mdoc_node(m), (struct html *)arg);
60 }
```

new/usr/src/cmd/mandoc/mdoc\_html.c

2

```
268     putchar('\n');
269 }
270
271 _____unchanged_portion_omitted_____
272
273 static void
274 print_mdoc(MDOC_ARGS)
275 {
276     struct tag *t, *tt;
277     struct htmlpair tag;
278
279     PAIR_CLASS_INIT(&tag, "mandoc");
280
281     if ( ! (HTML_FRAGMENT & h->oflags) ) {
282         print_gen_decls(h);
283         t = print_otag(h, TAG_HTML, 0, NULL);
284         tt = print_otag(h, TAG_HEAD, 0, NULL);
285         print_mdoc_head(meta, n, h);
286         print_mdoc_head(m, n, h);
287         print_tagq(h, tt);
288         print_otag(h, TAG_BODY, 0, NULL);
289         print_otag(h, TAG_DIV, 1, &tag);
290     } else
291         t = print_otag(h, TAG_DIV, 1, &tag);
292
293     print_mdoc_nodelist(meta, n, h);
294     print_mdoc_nodelist(m, n, h);
295     print_tagq(h, t);
296 }
297
298 /* ARGSUSED */
299 static void
300 print_mdoc_head(MDOC_ARGS)
301 {
302     print_gen_head(h);
303     bufinit(h);
304     bufcat_fmt(h, "%s(%s)", meta->title, meta->msec);
305     bufcat_fmt(h, "%s(%s)", m->title, m->msec);
306
307     if (meta->arch)
308         bufcat_fmt(h, " (%s)", meta->arch);
309     if (m->arch)
310         bufcat_fmt(h, " (%s)", m->arch);
311
312     print_otag(h, TAG_TITLE, 0, NULL);
313     print_text(h, h->buf);
314 }
315
316 static void
317 print_mdoc_nodelist(MDOC_ARGS)
318 {
319     print_mdoc_node(meta, n, h);
320     print_mdoc_node(m, n, h);
321     if (n->next)
322         print_mdoc_nodelist(meta, n->next, h);
323     print_mdoc_nodelist(m, n->next, h);
324 }
325
326 static void
327 print_mdoc_node(MDOC_ARGS)
328 {
```

```

407     int          child;
408     struct tag   *t;

410     child = 1;
411     t = h->tags.head;

413     switch (n->type) {
414     case (MDOC_ROOT):
415         child = mdoc_root_pre(meta, n, h);
416         child = mdoc_root_pre(m, n, h);
417         break;
418     case (MDOC_TEXT):
419         /* No tables in this mode... */
420         assert(NULL == h->tblt);

421         /*
422          * Make sure that if we're in a literal mode already
423          * (i.e., within a <PRE>) don't print the newline.
424          */
425         if (' ' == *n->string && MDOC_LINE & n->flags)
426             if (! (HTML_LITERAL & h->flags))
427                 print_otag(h, TAG_BR, 0, NULL);
428         if (MDOC_DELMIMC & n->flags)
429             h->flags |= HTML_NOSPACE;
430         print_text(h, n->string);
431         if (MDOC_DELIMO & n->flags)
432             h->flags |= HTML_NOSPACE;
433         return;
434     case (MDOC_EQN):
435         print_eqn(h, n->eqn);
436         break;
437     case (MDOC_TBL):
438         /*
439          * This will take care of initialising all of the table
440          * state data for the first table, then tearing it down
441          * for the last one.
442          */
443         print_tbl(h, n->span);
444         return;
445     default:
446         /*
447          * Close out the current table, if it's open, and unset
448          * the "meta" table state. This will be reopened on the
449          * next table element.
450          */
451         if (h->tblt) {
452             print_tblclose(h);
453             t = h->tags.head;
454         }

455         assert(NULL == h->tblt);
456         if (mdocs[n->tok].pre && ENDBODY_NOT == n->end)
457             child = (*mdocs[n->tok].pre)(meta, n, h);
458             child = (*mdocs[n->tok].pre)(m, n, h);
459         break;
460     }

462     if (HTML_KEEP & h->flags) {
463         if (n->prev ? (n->prev->lastline != n->line) :
464             (n->parent && n->parent->line != n->line)) {
465             if (n->prev && n->prev->line != n->line) {
466                 h->flags &= ~HTML_KEEP;
467                 h->flags |= HTML_PREKEEP;
468             } else if (NULL == n->prev) {
469                 if (n->parent && n->parent->line != n->line) {
470                     h->flags &= ~HTML_KEEP;

```

```

466         h->flags |= HTML_PREKEEP;
467     }
468 }
471 }

470     if (child && n->child)
471         print_mdoc_nodelist(meta, n->child, h);
472         print_mdoc_nodelist(m, n->child, h);

473     print_stagq(h, t);

475     switch (n->type) {
476     case (MDOC_ROOT):
477         mdoc_root_post(meta, n, h);
478         mdoc_root_post(m, n, h);
479         break;
480     case (MDOC_EQN):
481         break;
482     default:
483         if (mdocs[n->tok].post && ENDBODY_NOT == n->end)
484             (*mdocs[n->tok].post)(meta, n, h);
485             (*mdocs[n->tok].post)(m, n, h);
486         break;
487     }

488 /* ARGSUSED */
489 static void
490 mdoc_root_post(MDOC_ARGS)
491 {
492     struct htmlpair tag[3];
493     struct tag      *t, *tt;

494     PAIR_SUMMARY_INIT(&tag[0], "Document Footer");
495     PAIR_CLASS_INIT(&tag[1], "foot");
496     PAIR_INIT(&tag[2], ATTR_WIDTH, "100%");
497     t = print_otag(h, TAG_TABLE, 3, tag);
498     PAIR_INIT(&tag[0], ATTR_WIDTH, "50%");
499     print_otag(h, TAG_COL, 1, tag);
500     print_otag(h, TAG_COL, 1, tag);
501     print_otag(h, TAG_TBODY, 0, NULL);

502     tt = print_otag(h, TAG_TR, 0, NULL);

503     PAIR_CLASS_INIT(&tag[0], "foot-date");
504     print_otag(h, TAG_TD, 1, tag);
505     print_text(h, meta->date);
506     print_text(h, m->date);
507     print_stagq(h, tt);

508     PAIR_CLASS_INIT(&tag[0], "foot-os");
509     PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
510     print_otag(h, TAG_TD, 2, tag);
511     print_text(h, meta->os);
512     print_text(h, m->os);
513     print_tagq(h, t);
514 }

520 /* ARGSUSED */
521 static int
522 mdoc_root_pre(MDOC_ARGS)
523 {
524     struct htmlpair tag[3];
525     struct tag      *t, *tt;

```

```

526     char                b[BUFSIZ], title[BUFSIZ];

528     strcpy(b, meta->vol, BUFSIZ);
531     strcpy(b, m->vol, BUFSIZ);

530     if (meta->arch) {
533         if (m->arch) {
531             strcat(b, " ", BUFSIZ);
532             strcat(b, meta->arch, BUFSIZ);
535             strcat(b, m->arch, BUFSIZ);
533             strcat(b, " ", BUFSIZ);
534         }

536         snprintf(title, BUFSIZ - 1, "%s(%s)", meta->title, meta->msec);
539         snprintf(title, BUFSIZ - 1, "%s(%s)", m->title, m->msec);

538     PAIR_SUMMARY_INIT(&tag[0], "Document Header");
539     PAIR_CLASS_INIT(&tag[1], "head");
540     PAIR_INIT(&tag[2], ATTR_WIDTH, "100%");
541     t = print_otag(h, TAG_TABLE, 3, tag);
542     PAIR_INIT(&tag[0], ATTR_WIDTH, "30%");
543     print_otag(h, TAG_COL, 1, tag);
544     print_otag(h, TAG_COL, 1, tag);
545     print_otag(h, TAG_COL, 1, tag);

547     print_otag(h, TAG_TBODY, 0, NULL);

549     tt = print_otag(h, TAG_TR, 0, NULL);

551     PAIR_CLASS_INIT(&tag[0], "head-ltitle");
552     print_otag(h, TAG_TD, 1, tag);
553     print_text(h, title);
554     print_stagg(h, tt);

556     PAIR_CLASS_INIT(&tag[0], "head-vol");
557     PAIR_INIT(&tag[1], ATTR_ALIGN, "center");
558     print_otag(h, TAG_TD, 2, tag);
559     print_text(h, b);
560     print_stagg(h, tt);

562     PAIR_CLASS_INIT(&tag[0], "head-rtitle");
563     PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
564     print_otag(h, TAG_TD, 2, tag);
565     print_text(h, title);
566     print_tagq(h, t);
567     return(1);
568 }
    unchanged_portion_omitted

677 static int
678 mdoc_nm_pre(MDOC_ARGS)
679 {
680     struct htmlpair    tag;
681     struct roffsu      su;
682     int                len;

684     switch (n->type) {
685     case (MDOC_ELEM):
686         synopsis_pre(h, n);
687         PAIR_CLASS_INIT(&tag, "name");
688         print_otag(h, TAG_B, 1, &tag);
689         if (NULL == n->child && meta->name)
690             print_text(h, meta->name);
692         if (NULL == n->child && m->name)
693             print_text(h, m->name);

```

```

691         return(1);
692     case (MDOC_HEAD):
693         print_otag(h, TAG_TD, 0, NULL);
694         if (NULL == n->child && meta->name)
695             print_text(h, meta->name);
697         if (NULL == n->child && m->name)
698             print_text(h, m->name);
696         return(1);
697     case (MDOC_BODY):
698         print_otag(h, TAG_TD, 0, NULL);
699         return(1);
700     default:
701         break;
702     }

704     synopsis_pre(h, n);
705     PAIR_CLASS_INIT(&tag, "synopsis");
706     print_otag(h, TAG_TABLE, 1, &tag);

708     for (len = 0, n = n->child; n; n = n->next)
709         if (MDOC_TEXT == n->type)
710             len += html_strlen(n->string);

712     if (0 == len && meta->name)
713         len = html_strlen(meta->name);
715     if (0 == len && m->name)
716         len = html_strlen(m->name);

715     SCALE_HS_INIT(&su, (double)len);
716     bufinit(h);
717     bufcat_su(h, "width", &su);
718     PAIR_STYLE_INIT(&tag, h);
719     print_otag(h, TAG_COL, 1, &tag);
720     print_otag(h, TAG_COL, 0, NULL);
721     print_otag(h, TAG_TBODY, 0, NULL);
722     print_otag(h, TAG_TR, 0, NULL);
723     return(1);
724 }
    unchanged_portion_omitted

972 /* ARGSUSED */
973 static int
974 mdoc_bl_pre(MDOC_ARGS)
975 {
976     int                i;
977     struct htmlpair    tag[3];
978     struct roffsu      su;
979     char                buf[BUFSIZ];

984     bufinit(h);

981     if (MDOC_BODY == n->type) {
982         if (LIST_column == n->norm->Bl.type)
983             print_otag(h, TAG_TBODY, 0, NULL);
984         return(1);
985     }

987     if (MDOC_HEAD == n->type) {
988         if (LIST_column != n->norm->Bl.type)
989             return(0);

991     /*
992     * For each column, print out the <COL> tag with our
993     * suggested width. The last column gets min-width, as
994     * in terminal mode it auto-sizes to the width of the
995     * screen and we want to preserve that behaviour.

```

```

996          */
998          for (i = 0; i < (int)n->norm->Bl.ncols; i++) {
999              bufinit(h);
1000              a2width(n->norm->Bl.cols[i], &su);
1001              if (i < (int)n->norm->Bl.ncols - 1)
1002                  bufcat_su(h, "width", &su);
1003              else
1004                  bufcat_su(h, "min-width", &su);
1005              PAIR_STYLE_INIT(&tag[0], h);
1006              print_otag(h, TAG_COL, 1, tag);
1007          }
1009          return(0);
1010      }
1012      SCALE_VS_INIT(&su, 0);
1013      bufinit(h);
1014      bufcat_su(h, "margin-top", &su);
1015      bufcat_su(h, "margin-bottom", &su);
1016      PAIR_STYLE_INIT(&tag[0], h);
1018      assert(lists[n->norm->Bl.type]);
1019      strcpy(buf, "list ", BUFSIZ);
1020      strcat(buf, lists[n->norm->Bl.type], BUFSIZ);
1021      PAIR_INIT(&tag[1], ATTR_CLASS, buf);
1023      /* Set the block's left-hand margin. */
1025      if (n->norm->Bl.off5) {
1026          a2offs(n->norm->Bl.off5, &su);
1027          bufcat_su(h, "margin-left", &su);
1028      }
1030      switch (n->norm->Bl.type) {
1031      case(LIST_bullet):
1032          /* FALLTHROUGH */
1033      case(LIST_dash):
1034          /* FALLTHROUGH */
1035      case(LIST_hyphen):
1036          /* FALLTHROUGH */
1037      case(LIST_item):
1038          print_otag(h, TAG_UL, 2, tag);
1039          break;
1040      case(LIST_enum):
1041          print_otag(h, TAG_OL, 2, tag);
1042          break;
1043      case(LIST_diag):
1044          /* FALLTHROUGH */
1045      case(LIST_hang):
1046          /* FALLTHROUGH */
1047      case(LIST_inset):
1048          /* FALLTHROUGH */
1049      case(LIST_ohang):
1050          /* FALLTHROUGH */
1051      case(LIST_tag):
1052          print_otag(h, TAG_DL, 2, tag);
1053          break;
1054      case(LIST_column):
1055          print_otag(h, TAG_TABLE, 2, tag);
1056          break;
1057      default:
1058          abort();
1059          /* NOTREACHED */
1060      }

```

```

1062          return(1);
1063      }
1064      unchanged_portion_omitted
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074 /* ARGSUSED */
1075 static int
1076 mdoc_bd_pre(MDOC_ARGS)
1077 {
1078     struct htmlpair      tag[2];
1079     int                  comp, sv;
1080     const struct mdoc_node *nn;
1081     struct roffsu        su;
1082
1083     if (MDOC_HEAD == n->type)
1084         return(0);
1085
1086     if (MDOC_BLOCK == n->type) {
1087         comp = n->norm->Bd.comp;
1088         for (nn = n; nn && ! comp; nn = nn->parent) {
1089             if (MDOC_BLOCK != nn->type)
1090                 continue;
1091             if (MDOC_Ss == nn->tok || MDOC_Sh == nn->tok)
1092                 comp = 1;
1093             if (nn->prev)
1094                 break;
1095         }
1096         if (! comp)
1097             print_otag(h, TAG_P, 0, NULL);
1098         return(1);
1099     }
1100
1101     SCALE_HS_INIT(&su, 0);
1102     if (n->norm->Bd.off5)
1103         a2offs(n->norm->Bd.off5, &su);
1104
1105     bufinit(h);
1106     bufcat_su(h, "margin-left", &su);
1107     PAIR_STYLE_INIT(&tag[0], h);
1108
1109     if (DISP_unfilled != n->norm->Bd.type &&
1110         DISP_literal != n->norm->Bd.type) {
1111         PAIR_CLASS_INIT(&tag[1], "display");
1112         print_otag(h, TAG_DIV, 2, tag);
1113         return(1);
1114     }
1115
1116     PAIR_CLASS_INIT(&tag[1], "lit display");
1117     print_otag(h, TAG_PRE, 2, tag);
1118
1119     /* This can be recursive: save & set our literal state. */
1120
1121     sv = h->flags & HTML_LITERAL;
1122     h->flags |= HTML_LITERAL;
1123
1124     for (nn = n->child; nn; nn = nn->next) {
1125         print_mdoc_node(meta, nn, h);
1126         print_mdoc_node(m, nn, h);
1127         /*
1128          * If the printed node flushes its own line, then we
1129          * needn't do it here as well. This is hacky, but the
1130          * notion of selective eoln whitespace is pretty dumb
1131          * anyway, so don't sweat it.
1132          */
1133         switch (nn->tok) {
1134         case (MDOC_Sm):

```

```

1234         /* FALLTHROUGH */
1235     case (MDOC_br):
1236         /* FALLTHROUGH */
1237     case (MDOC_sp):
1238         /* FALLTHROUGH */
1239     case (MDOC_Bl):
1240         /* FALLTHROUGH */
1241     case (MDOC_Dl):
1242         /* FALLTHROUGH */
1243     case (MDOC_Dl):
1244         /* FALLTHROUGH */
1245     case (MDOC_Lp):
1246         /* FALLTHROUGH */
1247     case (MDOC_Pp):
1248         continue;
1249     default:
1250         break;
1251     }
1252     if (nn->next && nn->next->line == nn->line)
1253         continue;
1254     else if (nn->next)
1255         print_text(h, "\n");

1257     h->flags |= HTML_NOSPACE;
1258 }

1260     if (0 == sv)
1261         h->flags &= ~HTML_LITERAL;

1263     return(0);
1264 }

```

unchanged\_portion\_omitted

```

2222 /* ARGSUSED */
2223 static void
2224 mdoc_quote_post(MDOC_ARGS)
2225 {
2227     if (MDOC_BODY != n->type)
2228         return;

2230     h->flags |= HTML_NOSPACE;

2232     switch (n->tok) {
2233     case (MDOC_Ao):
2234         /* FALLTHROUGH */
2235     case (MDOC_Aq):
2236         print_text(h, "\\(ra");
2237         break;
2238     case (MDOC_Bro):
2239         /* FALLTHROUGH */
2240     case (MDOC_Brq):
2241         print_text(h, "\\(rC");
2242         break;
2243     case (MDOC_Oo):
2244         /* FALLTHROUGH */
2245     case (MDOC_Op):
2246         /* FALLTHROUGH */
2247     case (MDOC_Bo):
2248         /* FALLTHROUGH */
2249     case (MDOC_Bq):
2250         print_text(h, "\\(rB");
2251         break;
2252     case (MDOC_Eo):
2253         break;

```

```

2254     case (MDOC_Qo):
2255         /* FALLTHROUGH */
2256     case (MDOC_Qq):
2257         /* FALLTHROUGH */
2258     case (MDOC_Do):
2259         /* FALLTHROUGH */
2260     case (MDOC_Dq):
2261         print_text(h, "\\(rq");
2262         break;
2263     case (MDOC_Po):
2264         /* FALLTHROUGH */
2265     case (MDOC_Pq):
2266         print_text(h, "");
2267         break;
2268     case (MDOC_Ql):
2269         /* FALLTHROUGH */
2270     case (MDOC_So):
2271         /* FALLTHROUGH */
2272     case (MDOC_Sq):
2273         print_text(h, "\\(cq");
2274         print_text(h, "\\(aq");
2275         break;
2276     default:
2277         abort();
2278         /* NOTREACHED */
2279 }

```

unchanged\_portion\_omitted



```

*****
45476 Fri Jul 25 15:00:14 2014
new/usr/src/cmd/mandoc/mdoc_macro.c
Update to 1.12.3.
*****
1 /* $Id: mdoc_macro.c,v 1.125 2013/12/24 20:45:27 schwarze Exp $ */
1 /* $Id: mdoc_macro.c,v 1.115 2012/01/05 00:43:51 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdlib.h>
25 #include <stdio.h>
26 #include <string.h>
27 #include <time.h>
28
29 #include "mdoc.h"
30 #include "mandoc.h"
31 #include "libmdoc.h"
32 #include "libmandoc.h"
33
34 enum rew { /* see rew_dohalt() */
35     REWIND_NONE,
36     REWIND_THIS,
37     REWIND_MORE,
38     REWIND_FORCE,
39     REWIND_LATER,
40     REWIND_ERROR
41 };
42
43 static int blk_full(MACRO_PROT_ARGS);
44 static int blk_exp_close(MACRO_PROT_ARGS);
45 static int blk_part_exp(MACRO_PROT_ARGS);
46 static int blk_part_imp(MACRO_PROT_ARGS);
47 static int ctx_synopsis(MACRO_PROT_ARGS);
48 static int in_line_eoln(MACRO_PROT_ARGS);
49 static int in_line_argn(MACRO_PROT_ARGS);
50 static int in_line(MACRO_PROT_ARGS);
51 static int obsolete(MACRO_PROT_ARGS);
52 static int phrase_ta(MACRO_PROT_ARGS);
53
54 static int dword(struct mdoc *, int, int, const char *,
55                 enum mdelim, int);
54 static int dword(struct mdoc *, int, int,
55                 const char *, enum mdelim);
56 static int append_delims(struct mdoc *,

```

```

57     int, int *, char *);
58 static enum mdoct lookup(enum mdoct, const char *);
59 static enum mdoct lookup_raw(const char *);
60 static int make_pending(struct mdoc_node *, enum mdoct,
61                        struct mdoc *, int, int);
62 static int phrase(struct mdoc *, int, int, char *);
63 static enum mdoct rew_alt(enum mdoct);
64 static enum rew rew_dohalt(enum mdoct, enum mdoc_type,
65                            const struct mdoc_node *);
66 static int rew_elem(struct mdoc *, enum mdoct);
67 static int rew_last(struct mdoc *,
68                    const struct mdoc_node *);
69 static int rew_sub(enum mdoc_type, struct mdoc *,
70                  enum mdoct, int, int);
71
72 const struct mdoc_macro __mdoc_macros[MDOC_MAX] = {
73     { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ap */
73     { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ap */
74     { in_line_eoln, MDOC_PROLOGUE }, /* Dd */
75     { in_line_eoln, MDOC_PROLOGUE }, /* Dt */
76     { in_line_eoln, MDOC_PROLOGUE }, /* Os */
77     { blk_full, MDOC_PARSED | MDOC_JOIN }, /* Sh */
78     { blk_full, MDOC_PARSED | MDOC_JOIN }, /* Ss */
77     { blk_full, MDOC_PARSED }, /* Sh */
78     { blk_full, MDOC_PARSED }, /* Ss */
79     { in_line_eoln, 0 }, /* Pp */
80     { blk_part_imp, MDOC_PARSED | MDOC_JOIN }, /* D1 */
81     { blk_part_imp, MDOC_PARSED | MDOC_JOIN }, /* D1 */
80     { blk_part_imp, MDOC_PARSED }, /* D1 */
81     { blk_part_imp, MDOC_PARSED }, /* D1 */
82     { blk_full, MDOC_EXPLICIT }, /* Bd */
83     { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ed */
83     { blk_exp_close, MDOC_EXPLICIT }, /* Ed */
84     { blk_full, MDOC_EXPLICIT }, /* Bl */
85     { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* E1 */
86     { blk_full, MDOC_PARSED | MDOC_JOIN }, /* It */
85     { blk_exp_close, MDOC_EXPLICIT }, /* E1 */
86     { blk_full, MDOC_PARSED }, /* It */
87     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ad */
88     { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* An */
88     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* An */
89     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ar */
90     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Cd */
91     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Cm */
92     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Dv */
93     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Er */
94     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ev */
95     { in_line_eoln, 0 }, /* Ex */
96     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fa */
97     { in_line_eoln, 0 }, /* Fd */
98     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fl */
99     { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fn */
100    { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ft */
101    { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ic */
102    { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* In */
103    { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Li */
104    { blk_full, MDOC_JOIN }, /* Nd */
103    { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Li */
104    { blk_full, 0 }, /* Nd */
105    { ctx_synopsis, MDOC_CALLABLE | MDOC_PARSED }, /* Nm */
106    { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Op */
107    { obsolete, 0 }, /* Ot */
108    { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Pa */
109    { in_line_eoln, 0 }, /* Rv */
110    { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* St */
111    { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Va */

```

```

112 { ctx_synopsis, MDOC_CALLABLE | MDOC_PARSED }, /* Vt */
113 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Xr */
114 { in_line_eoln, MDOC_JOIN }, /* %A */
115 { in_line_eoln, MDOC_JOIN }, /* %B */
116 { in_line_eoln, MDOC_JOIN }, /* %D */
117 { in_line_eoln, MDOC_JOIN }, /* %I */
118 { in_line_eoln, MDOC_JOIN }, /* %J */
119 { in_line_eoln, 0 }, /* %A */
120 { in_line_eoln, 0 }, /* %B */
121 { in_line_eoln, 0 }, /* %D */
122 { in_line_eoln, 0 }, /* %I */
123 { in_line_eoln, 0 }, /* %J */
124 { in_line_eoln, 0 }, /* %N */
125 { in_line_eoln, MDOC_JOIN }, /* %O */
126 { in_line_eoln, 0 }, /* %O */
127 { in_line_eoln, 0 }, /* %P */
128 { in_line_eoln, MDOC_JOIN }, /* %R */
129 { in_line_eoln, MDOC_JOIN }, /* %T */
130 { in_line_eoln, 0 }, /* %R */
131 { in_line_eoln, 0 }, /* %T */
132 { in_line_eoln, 0 }, /* %V */
133 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
134   MDOC_EXPLICIT | MDOC_JOIN }, /* Ac */
135 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
136   MDOC_EXPLICIT | MDOC_JOIN }, /* Ao */
137 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Aq */
138 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Ac */
139 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Ao */
140 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Aq */
141 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* At */
142 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
143   MDOC_EXPLICIT | MDOC_JOIN }, /* Bc */
144 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Bc */
145 { blk_full, MDOC_EXPLICIT }, /* Bf */
146 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
147   MDOC_EXPLICIT | MDOC_JOIN }, /* Bo */
148 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Bq */
149 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bo */
150 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Bq */
151 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bs */
152 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bx */
153 { in_line_eoln, 0 }, /* Db */
154 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
155   MDOC_EXPLICIT | MDOC_JOIN }, /* Dc */
156 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
157   MDOC_EXPLICIT | MDOC_JOIN }, /* Do */
158 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Dq */
159 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Ec */
160 { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ef */
161 { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Em */
162 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Dc */
163 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Do */
164 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Dq */
165 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Ec */
166 { blk_exp_close, MDOC_EXPLICIT }, /* Ef */
167 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Em */
168 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Eo */
169 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Fx */
170 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ms */
171 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED |
172   MDOC_EXPLICIT | MDOC_JOIN }, /* No */
173 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED |
174   MDOC_EXPLICIT | MDOC_JOIN }, /* Ns */
175 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* No */
176 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Ns */
177 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Nx */

```

```

178 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ox */
179 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
180   MDOC_EXPLICIT | MDOC_JOIN }, /* Pc */
181 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Pc */
182 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Pf */
183 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
184   MDOC_EXPLICIT | MDOC_JOIN }, /* Po */
185 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Pq */
186 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
187   MDOC_EXPLICIT | MDOC_JOIN }, /* Qc */
188 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ql */
189 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
190   MDOC_EXPLICIT | MDOC_JOIN }, /* Qo */
191 { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Qq */
192 { blk_part_exp, MDOC_CALLABLE | MDOC_JOIN }, /* Re */
193 { blk_part_imp, MDOC_CALLABLE | MDOC_CALLABLE | MDOC_EXPLICIT }, /* Po */
194 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Pq */
195 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Qc */
196 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Ql */
197 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Qo */
198 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Qq */
199 { blk_exp_close, MDOC_EXPLICIT }, /* Re */
200 { blk_full, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_EXPLICIT }, /* Rf */
201 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
202   MDOC_EXPLICIT | MDOC_JOIN }, /* Sc */
203 { blk_part_exp, MDOC_CALLABLE | MDOC_EXPLICIT | MDOC_JOIN }, /* So */
204 { blk_part_imp, MDOC_CALLABLE | MDOC_EXPLICIT | MDOC_JOIN }, /* Sq */
205 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Sc */
206 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* So */
207 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Sq */
208 { in_line_eoln, 0 }, /* Sm */
209 { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Sx */
210 { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Sy */
211 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sx */
212 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sy */
213 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Tn */
214 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ux */
215 { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ux */
216 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Xc */
217 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Xo */
218 { blk_full, MDOC_EXPLICIT | MDOC_CALLABLE }, /* Fo */
219 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
220   MDOC_EXPLICIT | MDOC_JOIN }, /* Fc */
221 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
222   MDOC_EXPLICIT | MDOC_JOIN }, /* Oo */
223 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
224   MDOC_EXPLICIT | MDOC_JOIN }, /* Oc */
225 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Fc */
226 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Oo */
227 { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Oc */
228 { blk_full, MDOC_EXPLICIT }, /* Bk */
229 { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ek */
230 { blk_exp_close, MDOC_EXPLICIT }, /* Ek */
231 { in_line_eoln, 0 }, /* Bt */
232 { in_line_eoln, 0 }, /* Hf */
233 { obsolete, 0 }, /* Fr */
234 { in_line_eoln, 0 }, /* Ud */
235 { in_line, 0 }, /* Lb */
236 { in_line_eoln, 0 }, /* Lp */
237 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Lk */
238 { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Mt */
239 { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Brq */
240 { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
241   MDOC_EXPLICIT | MDOC_JOIN }, /* Bro */
242 { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |

```

```

204         MDOC_EXPLICIT | MDOC_JOIN }, /* Brc */
205     { in_line_eoln, MDOC_JOIN }, /* %C */
183     { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Brq */
184     { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bro */
185     { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Brc */
186     { in_line_eoln, 0 }, /* %C */
206     { obsolete, 0 }, /* Es */
207     { obsolete, 0 }, /* En */
208     { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Dx */
209     { in_line_eoln, MDOC_JOIN }, /* %Q */
190     { in_line_eoln, 0 }, /* %Q */
210     { in_line_eoln, 0 }, /* br */
211     { in_line_eoln, 0 }, /* sp */
212     { in_line_eoln, 0 }, /* %U */
213     { phrase_ta, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ta */
194     { phrase_ta, MDOC_CALLABLE | MDOC_PARSED }, /* Ta */
214 };

216 const struct mdoc_macro * const mdoc_macros = __mdoc_macros;

219 /*
220 * This is called at the end of parsing. It must traverse up the tree,
221 * closing out open [implicit] scopes. Obviously, open explicit scopes
222 * are errors.
223 */
224 int
225 mdoc_macroend(struct mdoc *mdoc)
206 mdoc_macroend(struct mdoc *m)
226 {
227     struct mdoc_node *n;

229     /* Scan for open explicit scopes. */

231     n = MDOC_VALID & mdoc->last->flags ?
232         mdoc->last->parent : mdoc->last;
212     n = MDOC_VALID & m->last->flags ? m->last->parent : m->last;

234     for ( ; n; n = n->parent)
235         if (MDOC_BLOCK == n->type &&
236             MDOC_EXPLICIT & mdoc_macros[n->tok].flags)
237             mdoc_nmsg(mdoc, n, MANDOCERR_SCOPEEXIT);
217             mdoc_nmsg(m, n, MANDOCERR_SCOPEEXIT);

239     /* Rewind to the first. */

241     return(rew_last(mdoc, mdoc->first));
221     return(rew_last(m, m->first));
242 }
    _____
    unchanged_portion_omitted_

274 static int
275 rew_last(struct mdoc *mdoc, const struct mdoc_node *to)
276 {
277     struct mdoc_node *n, *np;

279     assert(to);
280     mdoc->next = MDOC_NEXT_SIBLING;

282     /* LINTED */
283     while (mdoc->last != to) {
284         /*
285          * Save the parent here, because we may delete the
286          * mdoc->last node in the post-validation phase and reset
287          * it to mdoc->last->parent, causing a step in the closing

```

```

266     * m->last node in the post-validation phase and reset
267     * it to m->last->parent, causing a step in the closing
288     * out to be lost.
289     */
290     np = mdoc->last->parent;
291     if ( ! mdoc_valid_post(mdoc))
292         return(0);
293     n = mdoc->last;
294     mdoc->last = np;
295     assert(mdoc->last);
296     mdoc->last->last = n;
297 }

299     return(mdoc_valid_post(mdoc));
300 }
    _____
    unchanged_portion_omitted_

475 /*
476 * We are trying to close a block identified by tok,
477 * but the child block *broken is still open.
478 * Thus, postpone closing the tok block
479 * until the rew_sub call closing *broken.
480 */
481 static int
482 make_pending(struct mdoc_node *broken, enum mdoct tok,
483             struct mdoc *mdoc, int line, int ppos)
463             struct mdoc *m, int line, int ppos)
484 {
485     struct mdoc_node *breaker;

487     /*
488     * Iterate backwards, searching for the block matching tok,
489     * that is, the block breaking the *broken block.
490     */
491     for (breaker = broken->parent; breaker; breaker = breaker->parent) {

493         /*
494         * If the *broken block had already been broken before
495         * and we encounter its breaker, make the tok block
496         * pending on the inner breaker.
497         * Graphically, "[A breaker=[B broken=[C->B B] tok=A] C]"
498         * becomes "[A broken=[B [C->B B] tok=A] C]"
499         * and finally "[A [B->A [C->B B] A] C]".
500         */
501         if (breaker == broken->pending) {
502             broken = breaker;
503             continue;
504         }

506         if (REWIND_THIS != rew_dohalt(tok, MDOC_BLOCK, breaker))
507             continue;
508         if (MDOC_BODY == broken->type)
509             broken = broken->parent;

511         /*
512         * Found the breaker.
513         * If another, outer breaker is already pending on
514         * the *broken block, we must not clobber the link
515         * to the outer breaker, but make it pending on the
516         * new, now inner breaker.
517         * Graphically, "[A breaker=[B broken=[C->A A] tok=B] C]"
518         * becomes "[A breaker=[B->A broken=[C A] tok=B] C]"
519         * and finally "[A [B->A [C->B A] B] C]".
520         */
521         if (broken->pending) {

```

```

522         struct mdoc_node *taker;

524         /*
525          * If the breaker had also been broken before,
526          * it cannot take on the outer breaker itself,
527          * but must hand it on to its own breakers.
528          * Graphically, this is the following situation:
529          * "[A [B breaker=[C->B B] broken=[D->A A] tok=C] D]"
530          * "[A taker=[B->A breaker=[C->B B] [D->C A] C] D]"
531          */
532         taker = breaker;
533         while (taker->pending)
534             taker = taker->pending;
535         taker->pending = broken->pending;
536     }
537     broken->pending = breaker;
538     mandoc_vmsg(MANDOCERR_SCOPENEST, mdoc->parse, line, ppos,
539               mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
540                           "%s breaks %s", mdoc_macronames[tok],
541                           mdoc_macronames[broken->tok]));
542     return(1);
543 }

544 /*
545  * Found no matching block for tok.
546  * Are you trying to close a block that is not open?
547  */
548 return(0);
549 }

```

```

552 static int
553 rew_sub(enum mdoc_type t, struct mdoc *mdoc,
554         enum mdoc_type t, struct mdoc *m,
555         enum mdoct tok, int line, int ppos)
556 {
557     struct mdoc_node *n;

558     n = mdoc->last;
559     n = m->last;
560     while (n) {
561         switch (rew_dohalt(tok, t, n)) {
562             case (REWIND_NONE):
563                 return(1);
564             case (REWIND_THIS):
565                 n->lastline = line -
566                     (MDOC_NEWLINE & mdoc->flags &&
567                      ! (MDOC_EXPLICIT & mdoc_macros[tok].flags));
568                 break;
569             case (REWIND_FORCE):
570                 mandoc_vmsg(MANDOCERR_SCOPEBROKEN, mdoc->parse,
571                             mandoc_vmsg(MANDOCERR_SCOPEBROKEN, m->parse,
572                                         line, ppos, "%s breaks %s",
573                                         mdoc_macronames[tok],
574                                         mdoc_macronames[n->tok]));
575                 /* FALLTHROUGH */
576             case (REWIND_MORE):
577                 n->lastline = line -
578                     (MDOC_NEWLINE & mdoc->flags ? 1 : 0);
579                 n = n->parent;
580                 continue;
581             case (REWIND_LATER):
582                 if (make_pending(n, tok, mdoc, line, ppos) ||
583                     if (make_pending(n, tok, m, line, ppos) ||
584                         MDOC_BLOCK != t)
585                     return(1);

```

```

583         /* FALLTHROUGH */
584         case (REWIND_ERROR):
585             mandoc_vmsg(mdoc, line, ppos, MANDOCERR_NOSCOPE);
586             mdoc_vmsg(m, line, ppos, MANDOCERR_NOSCOPE);
587             return(1);
588         }
589     }
590     break;
591 }

592 assert(n);
593 if (! rew_last(mdoc, n))
594     if (! rew_last(m, n))
595         return(0);

596 /*
597  * The current block extends an enclosing block.
598  * Now that the current block ends, close the enclosing block, too.
599  */
600 while (NULL != (n = n->pending)) {
601     if (! rew_last(mdoc, n))
602         if (! rew_last(m, n))
603             return(0);
604     if (MDOC_HEAD == n->type &&
605         ! mdoc_body_alloc(mdoc, n->line, n->pos, n->tok))
606         ! mdoc_body_alloc(m, n->line, n->pos, n->tok))
607             return(0);
608 }

609 return(1);
610 }

611 /*
612  * Allocate a word and check whether it's punctuation or not.
613  * Punctuation consists of those tokens found in mdoc_isdelim().
614  */
615 static int
616 dword(struct mdoc *mdoc, int line, int col, const char *p,
617        enum mdelim d, int may_append)
618 {
619     struct mdoc *m;
620     int col, const char *p, enum mdelim d)
621 {
622     if (DELIM_MAX == d)
623         d = mdoc_isdelim(p);

624     if (may_append &&
625         ! ((MDOC_SYNOPSIS | MDOC_KEEP | MDOC_SMOFF) & mdoc->flags) &&
626         DELIM_NONE == d && MDOC_TEXT == mdoc->last->type &&
627         DELIM_NONE == mdoc_isdelim(mdoc->last->string)) {
628         mdoc_word_append(mdoc, p);
629         return(1);
630     }

631     if (! mdoc_word_alloc(mdoc, line, col, p))
632         if (! mdoc_word_alloc(m, line, col, p))
633             return(0);

634     if (DELIM_OPEN == d)
635         mdoc->last->flags |= MDOC_DELIMO;
636         m->last->flags |= MDOC_DELIMO;

637 /*
638  * Closing delimiters only suppress the preceding space
639  * when they follow something, not when they start a new
640  * block or element, and not when they follow 'No'.

```

```

641      * XXX Explicitly special-casing MDOC_No here feels
642      * like a layering violation. Find a better way
643      * and solve this in the code related to 'No'!
644      */

646      else if (DELIM_CLOSE == d && mdoc->last->prev &&
647              mdoc->last->prev->tok != MDOC_No &&
648              mdoc->last->parent->tok != MDOC_Fd)
649          mdoc->last->flags |= MDOC_DELIMC;
613      else if (DELIM_CLOSE == d && m->last->prev &&
614              m->last->prev->tok != MDOC_No)
615          m->last->flags |= MDOC_DELIMC;

651      return(1);
652  }

654  static int
655  append_delims(struct mdoc *mdoc, int line, int *pos, char *buf)
621  append_delims(struct mdoc *m, int line, int *pos, char *buf)
656  {
657      int          la;
658      enum margserr ac;
659      char         *p;

661      if ('\0' == buf[*pos])
662          return(1);

664      for (;;) {
665          la = *pos;
666          ac = mdoc_zargs(mdoc, line, pos, buf, &p);
632          ac = mdoc_zargs(m, line, pos, buf, &p);

668          if (ARGS_ERROR == ac)
669              return(0);
670          else if (ARGS_EOLN == ac)
671              break;

673          dword(mdoc, line, la, p, DELIM_MAX, 1);
639          dword(m, line, la, p, DELIM_MAX);

675          /*
676           * If we encounter end-of-sentence symbols, then trigger
677           * the double-space.
678           *
679           * XXX: it's easy to allow this to propagate outward to
680           * the last symbol, such that '.' will cause the
681           * correct double-spacing. However, (1) groff isn't
682           * smart enough to do this and (2) it would require
683           * knowing which symbols break this behaviour, for
684           * example, '.' shouldn't propagate the double-space.
685           */
686          if (mandoc_eos(p, strlen(p), 0))
687              mdoc->last->flags |= MDOC_EOS;
653              m->last->flags |= MDOC_EOS;
688      }

690      return(1);
691  }

694  /*
695   * Close out block partial/full explicit.
696   */
697  static int
698  blk_exp_close(MACRO_PROT_ARGS)
699  {

```

```

700      struct mdoc_node *body;          /* Our own body. */
701      struct mdoc_node *later;         /* A sub-block starting later. */
702      struct mdoc_node *n;             /* For searching backwards. */

704      int          j, lastarg, maxargs, flushed, nl;
705      enum margserr ac;
706      enum mdoct   atok, ntok;
707      char         *p;

709      nl = MDOC_NEWLINE & mdoc->flags;
675      nl = MDOC_NEWLINE & m->flags;

711      switch (tok) {
712      case (MDOC_Ec):
713          maxargs = 1;
714          break;
715      case (MDOC_Ek):
716          mdoc->flags &= ~MDOC_KEEP;
717      default:
718          maxargs = 0;
719          break;
720      }

722      /*
723       * Search backwards for beginnings of blocks,
724       * both of our own and of pending sub-blocks.
725       */
726      atok = rew_alt(tok);
727      body = later = NULL;
728      for (n = mdoc->last; n; n = n->parent) {
692          for (n = m->last; n; n = n->parent) {
729              if (MDOC_VALID & n->flags)
730                  continue;

732              /* Remember the start of our own body. */
733              if (MDOC_BODY == n->type && atok == n->tok) {
734                  if (ENDBODY_NOT == n->end)
735                      body = n;
736                  continue;
737              }

739              if (MDOC_BLOCK != n->type || MDOC_Nm == n->tok)
740                  continue;
741              if (atok == n->tok) {
742                  assert(body);

744                  /*
745                   * Found the start of our own block.
746                   * When there is no pending sub block,
747                   * just proceed to closing out.
748                   */
749                  if (NULL == later)
750                      break;

752                  /*
753                   * When there is a pending sub block,
754                   * postpone closing out the current block
755                   * until the rew_sub() closing out the sub-block.
756                   */
757                  make_pending(later, tok, mdoc, line, ppos);
721                  make_pending(later, tok, m, line, ppos);

759                  /*
760                   * Mark the place where the formatting - but not
761                   * the scope - of the current block ends.
762                   */

```

```

763     if ( ! mdoc_endbody_alloc(mdoc, line, ppos,
727         if ( ! mdoc_endbody_alloc(m, line, ppos,
764             atok, body, ENDBODY_SPACE))
765                 return(0);
766     break;
767 }
769 /*
770  * When finding an open sub block, remember the last
771  * open explicit block, or, in case there are only
772  * implicit ones, the first open implicit block.
773  */
774 if (later &&
775     MDOC_EXPLICIT & mdoc_macros[later->tok].flags)
776     continue;
777 if (MDOC_It != n->tok)
778     if (MDOC_CALLABLE & mdoc_macros[n->tok].flags)
779         later = n;
781 if ( ! (MDOC_CALLABLE & mdoc_macros[tok].flags)) {
782     /* FIXME: do this in validate */
783     if (buf[*pos])
784         mdoc_pmsg(mdoc, line, ppos, MANDOCERR_ARGSLOST);
785     mdoc_pmsg(m, line, ppos, MANDOCERR_ARGSLOST);
786     if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
787         if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
788             return(0);
789     return(rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos));
790     return(rew_sub(MDOC_BLOCK, m, tok, line, ppos));
791 }
792 if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
793     if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
794         return(0);
795 if (NULL == later && maxargs > 0)
796     if ( ! mdoc_tail_alloc(mdoc, line, ppos, rew_alt(tok)))
797         if ( ! mdoc_tail_alloc(m, line, ppos, rew_alt(tok)))
798             return(0);
799 for (flushed = j = 0; ; j++) {
800     lastarg = *pos;
801     if (j == maxargs && ! flushed) {
802         if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
803             if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
804                 return(0);
805         flushed = 1;
806     }
807     ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
808     ac = mdoc_args(m, line, pos, buf, tok, &p);
809     if (ARGS_ERROR == ac)
810         return(0);
811     if (ARGS_PUNCT == ac)
812         break;
813     if (ARGS_EOLN == ac)
814         break;
815     ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);
816     if (MDOC_MAX == ntok) {
817         if ( ! dword(mdoc, line, lastarg, p, DELIM_MAX,

```

```

820         MDOC_JOIN & mdoc_macros[tok].flags))
821             if ( ! dword(m, line, lastarg, p, DELIM_MAX))
822                 return(0);
823     continue;
824 }
825 if ( ! flushed) {
826     if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
827         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
828             return(0);
829     flushed = 1;
830 }
831 mdoc->flags &= ~MDOC_NEWLINE;
832 if ( ! mdoc_macro(mdoc, ntok, line, lastarg, pos, buf))
833     if ( ! mdoc_macro(m, ntok, line, lastarg, pos, buf))
834         return(0);
835 break;
836 }
837 if ( ! flushed && ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
838     if ( ! flushed && ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
839         return(0);
840 if ( ! nl)
841     return(1);
842 return(append_delims(mdoc, line, pos, buf));
843 return(append_delims(m, line, pos, buf));
844 }
845 static int
846 in_line(MACRO_PROT_ARGS)
847 {
848     int             la, scope, cnt, nc, nl;
849     enum margverr   av;
850     enum mdoct      ntok;
851     enum margserr   ac;
852     enum mdelim     d;
853     struct mdoc_arg *arg;
854     char            *p;
855     nl = MDOC_NEWLINE & mdoc->flags;
856     nl = MDOC_NEWLINE & m->flags;
857 /*
858  * Whether we allow ignored elements (those without content,
859  * usually because of reserved words) to squeak by.
860  */
861 switch (tok) {
862 case (MDOC_An):
863     /* FALLTHROUGH */
864 case (MDOC_Ar):
865     /* FALLTHROUGH */
866 case (MDOC_Fl):
867     /* FALLTHROUGH */
868 case (MDOC_Mt):
869     /* FALLTHROUGH */
870 case (MDOC_Nm):
871     /* FALLTHROUGH */
872 case (MDOC_Pa):
873     nc = 1;
874     break;
875 default:

```

```

880         nc = 0;
881         break;
882     }

884     for (arg = NULL;; ) {
885         la = *pos;
886         av = mdoc_argv(mdoc, line, tok, &arg, pos, buf);
887         av = mdoc_argv(m, line, tok, &arg, pos, buf);

888         if (ARGV_WORD == av) {
889             *pos = la;
890             break;
891         }
892         if (ARGV_EOLN == av)
893             break;
894         if (ARGV_ARG == av)
895             continue;

897         mdoc_argv_free(arg);
898         return(0);
899     }

901     for (cnt = scope = 0;; ) {
902         la = *pos;
903         ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
904         ac = mdoc_args(m, line, pos, buf, tok, &p);

905         if (ARGS_ERROR == ac)
906             return(0);
907         if (ARGS_EOLN == ac)
908             break;
909         if (ARGS_PUNCT == ac)
910             break;

912         ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

914         /*
915          * In this case, we've located a submacro and must
916          * execute it. Close out scope, if open. If no
917          * elements have been generated, either create one (nc)
918          * or raise a warning.
919          */

921         if (MDOC_MAX != ntok) {
922             if (scope && !rew_elem(mdoc, tok))
923                 if (scope && !rew_elem(m, tok))
924                     return(0);
925             if (nc && 0 == cnt) {
926                 if ( ! mdoc_elem_alloc(mdoc, line,
927                                     ppos, tok, arg))
928                     if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
929                         return(0);
930                 if ( ! rew_last(mdoc, mdoc->last))
931                     if ( ! rew_last(m, m->last))
932                         return(0);
933             } else if ( ! nc && 0 == cnt) {
934                 mdoc_argv_free(arg);
935                 mdoc_pmsg(mdoc, line, ppos,
936                          MANDOCERR_MACROEMPTY);
937                 mdoc_pmsg(m, line, ppos, MANDOCERR_MACROEMPTY);
938             }

936             if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
937                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
938                     return(0);
939             if ( ! nl)

```

```

939         return(1);
940         return(append_delims(mdoc, line, pos, buf));
941         return(append_delims(m, line, pos, buf));
942     }

943     /*
944     * Non-quote-enclosed punctuation. Set up our scope, if
945     * a word; rewind the scope, if a delimiter; then append
946     * the word.
947     */

949     d = ARGS_QWORD == ac ? DELIM_NONE : mdoc_isdelim(p);

951     if (DELIM_NONE != d) {
952         /*
953         * If we encounter closing punctuation, no word
954         * has been omitted, no scope is open, and we're
955         * allowed to have an empty element, then start
956         * a new scope. 'Ar', 'Fl', and 'Li', only do
957         * this once per invocation. There may be more
958         * of these (all of them?).
959         */
960         if (0 == cnt && (nc || MDOC_Li == tok) &&
961             DELIM_CLOSE == d && !scope) {
962             if ( ! mdoc_elem_alloc(mdoc, line,
963                                 ppos, tok, arg))
964                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
965                     return(0);
966             if (MDOC_Ar == tok || MDOC_Li == tok ||
967                 MDOC_Fl == tok)
968                 cnt++;
969             scope = 1;
970         }
971         /*
972         * Close out our scope, if one is open, before
973         * any punctuation.
974         */
975         if (scope && !rew_elem(mdoc, tok))
976             if (scope && !rew_elem(m, tok))
977                 return(0);
978         scope = 0;
979     } else if ( ! scope) {
980         if ( ! mdoc_elem_alloc(mdoc, line, ppos, tok, arg))
981             if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
982                 return(0);
983         scope = 1;
984     }

986     if (DELIM_NONE == d)
987         cnt++;

988     if ( ! dword(mdoc, line, la, p, d,
989                 MDOC_JOIN & mdoc_macros[tok].flags))
990         if ( ! dword(m, line, la, p, d))
991             return(0);

992     /*
993     * 'Fl' macros have their scope re-opened with each new
994     * word so that the '-' can be added to each one without
995     * having to parse out spaces.
996     */
997     if (scope && MDOC_Fl == tok) {
998         if ( ! rew_elem(mdoc, tok))
999             if ( ! rew_elem(m, tok))
1000                 return(0);
1001         scope = 0;

```

```

999     }
1000 }

1002 if (scope && ! rew_elem(mdoc, tok))
958   if (scope && ! rew_elem(m, tok))
1003     return(0);

1005 /*
1006  * If no elements have been collected and we're allowed to have
1007  * empties (nc), open a scope and close it out.  Otherwise,
1008  * raise a warning.
1009  */

1011 if (nc && 0 == cnt) {
1012   if ( ! mdoc_elem_alloc(mdoc, line, ppos, tok, arg))
968     if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
1013       return(0);
1014   if ( ! rew_last(mdoc, mdoc->last))
970     if ( ! rew_last(m, m->last))
1015     return(0);
1016 } else if ( ! nc && 0 == cnt) {
1017   mdoc_argv_free(arg);
1018   mdoc_pmsg(mdoc, line, ppos, MANDOCERR_MACROEMPTY);
974   mdoc_pmsg(m, line, ppos, MANDOCERR_MACROEMPTY);
1019 }

1021 if ( ! nl)
1022   return(1);
1023 return(append_delims(mdoc, line, pos, buf));
979 return(append_delims(m, line, pos, buf));
1024 }

1027 static int
1028 blk_full(MACRO_PROT_ARGS)
1029 {
1030   int          la, nl, nparsed;
1031   struct mdoc_arg *arg;
1032   struct mdoc_node *head; /* save of head macro */
1033   struct mdoc_node *body; /* save of body macro */
1034   struct mdoc_node *n;
1035   enum mdoc_type   mtt;
1036   enum mdoct       ntok;
1037   enum margserr    ac, lac;
1038   enum margverr    av;
1039   char             *p;

1041   nl = MDOC_NEWLINE & mdoc->flags;
997   nl = MDOC_NEWLINE & m->flags;

1043   /* Close out prior implicit scope. */

1045   if ( ! (MDOC_EXPLICIT & mdoc_macros[tok].flags)) {
1046     if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
1002     if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1047       return(0);
1048     if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
1004     if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1049       return(0);
1050   }

1052   /*
1053   * This routine accommodates implicitly- and explicitly-scoped
1054   * macro openings.  Implicit ones first close out prior scope
1055   * (seen above).  Delay opening the head until necessary to
1056   * allow leading punctuation to print.  Special consideration

```

```

1057   * for 'It -column', which has phrase-part syntax instead of
1058   * regular child nodes.
1059   */

1061   for (arg = NULL;; ) {
1062     la = *pos;
1063     av = mdoc_argv(mdoc, line, tok, &arg, pos, buf);
1019     av = mdoc_argv(m, line, tok, &arg, pos, buf);

1065     if (ARGV_WORD == av) {
1066       *pos = la;
1067       break;
1068     }

1070     if (ARGV_EOLN == av)
1071       break;
1072     if (ARGV_ARG == av)
1073       continue;

1075     mdoc_argv_free(arg);
1076     return(0);
1077   }

1079   if ( ! mdoc_block_alloc(mdoc, line, ppos, tok, arg))
1035   if ( ! mdoc_block_alloc(m, line, ppos, tok, arg))
1080     return(0);

1082   head = body = NULL;

1084   /*
1085   * Exception: Heads of 'It' macros in '-diag' lists are not
1086   * parsed, even though 'It' macros in general are parsed.
1087   */
1088   nparsed = MDOC_It == tok &&
1089             MDOC_Bl == mdoc->last->parent->tok &&
1090             LIST_diag == mdoc->last->parent->norm->Bl.type;
1045   MDOC_Bl == m->last->parent->tok &&
1046   LIST_diag == m->last->parent->norm->Bl.type;

1092   /*
1093   * The 'Nd' macro has all arguments in its body: it's a hybrid
1094   * of block partial-explicit and full-implicit.  Stupid.
1095   */

1097   if (MDOC_Nd == tok) {
1098     if ( ! mdoc_head_alloc(mdoc, line, ppos, tok))
1054     if ( ! mdoc_head_alloc(m, line, ppos, tok))
1099       return(0);
1100     head = mdoc->last;
1101     if ( ! rew_sub(MDOC_HEAD, mdoc, tok, line, ppos))
1056     head = m->last;
1057     if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1102       return(0);
1103     if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1059     if ( ! mdoc_body_alloc(m, line, ppos, tok))
1104       return(0);
1105     body = mdoc->last;
1061     body = m->last;
1106   }

1108   if (MDOC_Bk == tok)
1109     mdoc->flags |= MDOC_KEEP;

1111   ac = ARGS_ERROR;

1113   for ( ; ; ) {

```



```

1114     la = *pos;
1115     /* Initialise last-phrase-type with ARGS_PEND. */
1116     lac = ARGS_ERROR == ac ? ARGS_PEND : ac;
1117     ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
1070     ac = mdoc_args(m, line, pos, buf, tok, &p);

1119     if (ARGS_PUNCT == ac)
1120         break;

1122     if (ARGS_ERROR == ac)
1123         return(0);

1125     if (ARGS_EOLN == ac) {
1126         if (ARGS_PPHRASE != lac && ARGS_PHRASE != lac)
1127             break;
1128         /*
1129          * This is necessary: if the last token on a
1130          * line is a 'Ta' or tab, then we'll get
1131          * ARGS_EOLN, so we must be smart enough to
1132          * reopen our scope if the last parse was a
1133          * phrase or partial phrase.
1134          */
1135         if (!rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
1088         if (!rew_sub(MDOC_BODY, m, tok, line, ppos))
1136             return(0);
1137         if (!mdoc_body_alloc(mdoc, line, ppos, tok))
1090         if (!mdoc_body_alloc(m, line, ppos, tok))
1138             return(0);
1139         body = mdoc->last;
1092         body = m->last;
1140         break;
1141     }

1143     /*
1144     * Emit leading punctuation (i.e., punctuation before
1145     * the MDOC_HEAD) for non-phrase types.
1146     */

1148     if (NULL == head &&
1149         ARGS_PEND != ac &&
1150         ARGS_PHRASE != ac &&
1151         ARGS_PPHRASE != ac &&
1152         ARGS_QWORD != ac &&
1153         DELIM_OPEN == mdoc_isdelim(p)) {
1154         if (!dword(mdoc, line, la, p, DELIM_OPEN, 0))
1107         if (!dword(m, line, la, p, DELIM_OPEN))
1155             return(0);
1156         continue;
1157     }

1159     /* Open a head if one hasn't been opened. */

1161     if (NULL == head) {
1162         if (!mdoc_head_alloc(mdoc, line, ppos, tok))
1115         if (!mdoc_head_alloc(m, line, ppos, tok))
1163             return(0);
1164         head = mdoc->last;
1117         head = m->last;
1165     }

1167     if (ARGS_PHRASE == ac ||
1168         ARGS_PEND == ac ||
1169         ARGS_PPHRASE == ac) {
1170         /*
1171         * If we haven't opened a body yet, rewind the
1172         * head; if we have, rewind that instead.

```

```

1173     */

1175     mtt = body ? MDOC_BODY : MDOC_HEAD;
1176     if (!rew_sub(mtt, mdoc, tok, line, ppos))
1129     if (!rew_sub(mtt, m, tok, line, ppos))
1177         return(0);
1178
1179     /* Then allocate our body context. */

1181     if (!mdoc_body_alloc(mdoc, line, ppos, tok))
1134     if (!mdoc_body_alloc(m, line, ppos, tok))
1182         return(0);
1183     body = mdoc->last;
1136     body = m->last;

1185     /*
1186     * Process phrases: set whether we're in a
1187     * partial-phrase (this effects line handling)
1188     * then call down into the phrase parser.
1189     */

1191     if (ARGS_PPHRASE == ac)
1192         mdoc->flags |= MDOC_PPHRASE;
1145     m->flags |= MDOC_PPHRASE;
1193     if (ARGS_PEND == ac && ARGS_PPHRASE == lac)
1194         mdoc->flags |= MDOC_PPHRASE;
1147     m->flags |= MDOC_PPHRASE;

1196     if (!phrase(mdoc, line, la, buf))
1149     if (!phrase(m, line, la, buf))
1197         return(0);

1199     mdoc->flags &= ~MDOC_PPHRASE;
1152     m->flags &= ~MDOC_PPHRASE;
1200     continue;
1201 }

1203     ntok = nparsed || ARGS_QWORD == ac ?
1204         MDOC_MAX : lookup(tok, p);

1206     if (MDOC_MAX == ntok) {
1207         if (!dword(mdoc, line, la, p, DELIM_MAX,
1208             MDOC_JOIN & mdoc_macros[tok].flags))
1160         if (!dword(m, line, la, p, DELIM_MAX))
1209             return(0);
1210         continue;
1211     }

1213     if (!mdoc_macro(mdoc, ntok, line, la, pos, buf))
1165     if (!mdoc_macro(m, ntok, line, la, pos, buf))
1214         return(0);
1215     break;
1216 }

1218     if (NULL == head) {
1219         if (!mdoc_head_alloc(mdoc, line, ppos, tok))
1171         if (!mdoc_head_alloc(m, line, ppos, tok))
1220             return(0);
1221         head = mdoc->last;
1173         head = m->last;
1222     }

1223
1224     if (nl && !append_delims(mdoc, line, pos, buf))
1176     if (nl && !append_delims(m, line, pos, buf))
1225         return(0);

```

```

1227      /* If we've already opened our body, exit now. */
1229      if (NULL != body)
1230          goto out;

1232      /*
1233       * If there is an open (i.e., unvalidated) sub-block requiring
1234       * explicit close-out, postpone switching the current block from
1235       * head to body until the rew_sub() call closing out that
1236       * sub-block.
1237       */
1238      for (n = mdoc->last; n && n != head; n = n->parent) {
1190      for (n = m->last; n && n != head; n = n->parent) {
1239          if (MDOC_BLOCK == n->type &&
1240              MDOC_EXPLICIT & mdoc_macros[n->tok].flags &&
1241              !(MDOC_VALID & n->flags)) {
1242              n->pending = head;
1243              return(1);
1244          }
1245      }

1247      /* Close out scopes to remain in a consistent state. */

1249      if ( ! rew_sub(MDOC_HEAD, mdoc, tok, line, ppos))
1201      if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1250          return(0);
1251      if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1203      if ( ! mdoc_body_alloc(m, line, ppos, tok))
1252          return(0);

1254 out:
1255      if ( ! (MDOC_FREECOL & mdoc->flags))
1207      if ( ! (MDOC_FREECOL & m->flags))
1256          return(1);

1258      if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
1210      if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1259          return(0);
1260      if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
1212      if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1261          return(0);

1263      mdoc->flags &= ~MDOC_FREECOL;
1215      m->flags &= ~MDOC_FREECOL;
1264      return(1);
1265 }

1268 static int
1269 blk_part_imp(MACRO_PROT_ARGS)
1270 {
1271     int          la, nl;
1272     enum mdoct   ntok;
1273     enum margserr ac;
1274     char         *p;
1275     struct mdoc_node *blk; /* saved block context */
1276     struct mdoc_node *body; /* saved body context */
1277     struct mdoc_node *n;

1279     nl = MDOC_NEWLINE & mdoc->flags;
1231     nl = MDOC_NEWLINE & m->flags;

1281     /*
1282     * A macro that spans to the end of the line. This is generally
1283     * (but not necessarily) called as the first macro. The block
1284     * has a head as the immediate child, which is always empty,

```

```

1285     * followed by zero or more opening punctuation nodes, then the
1286     * body (which may be empty, depending on the macro), then zero
1287     * or more closing punctuation nodes.
1288     */

1290     if ( ! mdoc_block_alloc(mdoc, line, ppos, tok, NULL))
1242     if ( ! mdoc_block_alloc(m, line, ppos, tok, NULL))
1291         return(0);

1293     blk = mdoc->last;
1245     blk = m->last;

1295     if ( ! mdoc_head_alloc(mdoc, line, ppos, tok))
1247     if ( ! mdoc_head_alloc(m, line, ppos, tok))
1296         return(0);
1297     if ( ! rew_sub(MDOC_HEAD, mdoc, tok, line, ppos))
1249     if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1298         return(0);

1300     /*
1301     * Open the body scope "on-demand", that is, after we've
1302     * processed all our the leading delimiters (open parenthesis,
1303     * etc.).
1304     */

1306     for (body = NULL; ; ) {
1307         la = *pos;
1308         ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
1260         ac = mdoc_args(m, line, pos, buf, tok, &p);

1310         if (ARGS_ERROR == ac)
1311             return(0);
1312         if (ARGS_EOLN == ac)
1313             break;
1314         if (ARGS_PUNCT == ac)
1315             break;

1317         if (NULL == body && ARGS_QWORD != ac &&
1318             DELIM_OPEN == mdoc_isdelim(p)) {
1319             if ( ! dword(mdoc, line, la, p, DELIM_OPEN, 0))
1271             if ( ! dword(m, line, la, p, DELIM_OPEN))
1320                 return(0);
1321             continue;
1322         }

1324         if (NULL == body) {
1325             if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1277             if ( ! mdoc_body_alloc(m, line, ppos, tok))
1326                 return(0);
1327             body = mdoc->last;
1279             body = m->last;
1328         }

1330         ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1332         if (MDOC_MAX == ntok) {
1333             if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1334                 MDOC_JOIN & mdoc_macros[tok].flags))
1285             if ( ! dword(m, line, la, p, DELIM_MAX))
1335                 return(0);
1336             continue;
1337         }

1339         if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
1290         if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1340             return(0);

```

```

1341         break;
1342     }

1344     /* Clean-ups to leave in a consistent state. */

1346     if (NULL == body) {
1347         if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1298         if ( ! mdoc_body_alloc(m, line, ppos, tok))
1348             return(0);
1349         body = mdoc->last;
1300         body = m->last;
1350     }

1352     for (n = body->child; n && n->next; n = n->next)
1353         /* Do nothing. */ ;
1354
1355     /*
1356     * End of sentence spacing: if the last node is a text node and
1357     * has a trailing period, then mark it as being end-of-sentence.
1358     */

1360     if (n && MDOC_TEXT == n->type && n->string)
1361         if (mandoc_eos(n->string, strlen(n->string), 1))
1362             n->flags |= MDOC_EOS;

1364     /* Up-propagate the end-of-space flag. */

1366     if (n && (MDOC_EOS & n->flags)) {
1367         body->flags |= MDOC_EOS;
1368         body->parent->flags |= MDOC_EOS;
1369     }

1371     /*
1372     * If there is an open sub-block requiring explicit close-out,
1373     * postpone closing out the current block
1374     * until the rew_sub() call closing out the sub-block.
1375     */
1376     for (n = mdoc->last; n && n != body && n != blk->parent;
1377          n = n->parent) {
1378     for (n = m->last; n && n != body && n != blk->parent; n = n->parent) {
1379         if (MDOC_BLOCK == n->type &&
1380             MDOC_EXPLICIT & mdoc_macros[n->tok].flags &&
1381             ! (MDOC_VALID & n->flags)) {
1382             make_pending(n, tok, mdoc, line, ppos);
1383             if ( ! mdoc_endbody_alloc(mdoc, line, ppos,
1384                 make_pending(n, tok, m, line, ppos);
1385                 if ( ! mdoc_endbody_alloc(m, line, ppos,
1386                     tok, body, ENDBODY_NOSPACE))
1387                     return(0);
1388             }
1389         }
1390     }
1391     return(1);
1392     }

1393     /*
1394     * If we can't rewind to our body, then our scope has already
1395     * been closed by another macro (like 'Oc' closing 'Op'). This
1396     * is ugly behaviour nodding its head to OpenBSD's overwhelming
1397     * crufty use of 'Op' breakage.
1398     */
1399     if (n != body)
1400         mandoc_vmsg(MANDOCERR_SCOPENEST, mdoc->parse, line, ppos,
1401             mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
1402                 "%s broken", mdoc_macronames[tok]));

1403     if (n && ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
1404     if (n && ! rew_sub(MDOC_BODY, m, tok, line, ppos))

```

```

1400         return(0);

1402     /* Standard appending of delimiters. */

1404     if (nl && ! append_delims(mdoc, line, pos, buf))
1354     if (nl && ! append_delims(m, line, pos, buf))
1405         return(0);

1407     /* Rewind scope, if applicable. */

1409     if (n && ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
1359     if (n && ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1410         return(0);

1412     /* Move trailing .Ns out of scope. */

1414     for (n = body->child; n && n->next; n = n->next)
1415         /* Do nothing. */ ;
1416     if (n && MDOC_Ns == n->tok)
1417         mdoc_node_relink(mdoc, n);

1419     return(1);
1420 }

1423 static int
1424 blk_part_exp(MACRO_PROT_ARGS)
1425 {
1426     int                la, nl;
1427     enum margserr      ac;
1428     struct mdoc_node   *head; /* keep track of head */
1429     struct mdoc_node   *body; /* keep track of body */
1430     char               *p;
1431     enum mdoct         ntok;

1433     nl = MDOC_NEWLINE & mdoc->flags;
1376     nl = MDOC_NEWLINE & m->flags;

1435     /*
1436     * The opening of an explicit macro having zero or more leading
1437     * punctuation nodes; a head with optional single element (the
1438     * case of 'Eo'); and a body that may be empty.
1439     */

1441     if ( ! mdoc_block_alloc(mdoc, line, ppos, tok, NULL))
1384     if ( ! mdoc_block_alloc(m, line, ppos, tok, NULL))
1442         return(0);

1444     for (head = body = NULL; ; ) {
1445         la = *pos;
1446         ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
1389         ac = mdoc_args(m, line, pos, buf, tok, &p);

1448         if (ARGS_ERROR == ac)
1449             return(0);
1450         if (ARGS_PUNCT == ac)
1451             break;
1452         if (ARGS_EOLN == ac)
1453             break;

1455     /* Flush out leading punctuation. */

1457     if (NULL == head && ARGS_QWORD != ac &&
1458         DELIM_OPEN == mdoc_isdelim(p)) {
1459         assert(NULL == body);
1460         if ( ! dword(mdoc, line, la, p, DELIM_OPEN, 0))

```

```

1403     if ( ! dword(m, line, la, p, DELIM_OPEN))
1461         return(0);
1462     continue;
1463 }

1465     if (NULL == head) {
1466         assert(NULL == body);
1467         if ( ! mdoc_head_alloc(mdoc, line, ppos, tok))
1410             if ( ! mdoc_head_alloc(m, line, ppos, tok))
1468                 return(0);
1469         head = mdoc->last;
1412         head = m->last;
1470     }

1472     /*
1473     * 'Eo' gobbles any data into the head, but most other
1474     * macros just immediately close out and begin the body.
1475     */

1477     if (NULL == body) {
1478         assert(head);
1479         /* No check whether it's a macro! */
1480         if (MDOC_Eo == tok)
1481             if ( ! dword(mdoc, line, la, p, DELIM_MAX, 0))
1424                 if ( ! dword(m, line, la, p, DELIM_MAX))
1482                     return(0);

1484         if ( ! rew_sub(MDOC_HEAD, mdoc, tok, line, ppos))
1427             if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1485                 return(0);
1486         if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1429             if ( ! mdoc_body_alloc(m, line, ppos, tok))
1487                 return(0);
1488         body = mdoc->last;
1431         body = m->last;

1490         if (MDOC_Eo == tok)
1491             continue;
1492     }

1494     assert(NULL != head && NULL != body);

1496     ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1498     if (MDOC_MAX == ntok) {
1499         if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1500             MDOC_JOIN & mdoc_macros[tok].flags))
1442             if ( ! dword(m, line, la, p, DELIM_MAX))
1501                 return(0);
1502     }
1503     continue;

1505     if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
1447         if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1506             return(0);
1507     break;
1508 }

1510     /* Clean-up to leave in a consistent state. */

1512     if (NULL == head)
1513         if ( ! mdoc_head_alloc(mdoc, line, ppos, tok))
1455             if ( ! mdoc_head_alloc(m, line, ppos, tok))
1514                 return(0);

1516     if (NULL == body) {

```

```

1517         if ( ! rew_sub(MDOC_HEAD, mdoc, tok, line, ppos))
1459             if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1518                 return(0);
1519         if ( ! mdoc_body_alloc(mdoc, line, ppos, tok))
1461             if ( ! mdoc_body_alloc(m, line, ppos, tok))
1520                 return(0);
1521     }

1523     /* Standard appending of delimiters. */

1525     if ( ! nl)
1526         return(1);
1527     return(append_delims(mdoc, line, pos, buf));
1469     return(append_delims(m, line, pos, buf));
1528 }

1531 /* ARGSUSED */
1532 static int
1533 in_line_argn(MACRO_PROT_ARGS)
1534 {
1535     int             la, flushed, j, maxargs, nl;
1536     enum margserr   ac;
1537     enum margverr   av;
1538     struct mdoc_arg *arg;
1539     char            *p;
1540     enum mdoct      ntok;

1542     nl = MDOC_NEWLINE & mdoc->flags;
1484     nl = MDOC_NEWLINE & m->flags;

1544     /*
1545     * A line macro that has a fixed number of arguments (maxargs).
1546     * Only open the scope once the first non-leading-punctuation is
1547     * found (unless MDOC_IGNDELIM is noted, like in 'Pf'), then
1548     * keep it open until the maximum number of arguments are
1549     * exhausted.
1550     */

1552     switch (tok) {
1553     case (MDOC_Ap):
1554         /* FALLTHROUGH */
1555     case (MDOC_No):
1556         /* FALLTHROUGH */
1557     case (MDOC_Ns):
1558         /* FALLTHROUGH */
1559     case (MDOC_Ux):
1560         maxargs = 0;
1561         break;
1562     case (MDOC_Bx):
1563         /* FALLTHROUGH */
1564     case (MDOC_Xr):
1565         maxargs = 2;
1566         break;
1567     default:
1568         maxargs = 1;
1569         break;
1570     }

1572     for (arg = NULL; ; ) {
1573         la = *pos;
1574         av = mdoc_argv(mdoc, line, tok, &arg, pos, buf);
1516         av = mdoc_argv(m, line, tok, &arg, pos, buf);

1576         if (ARGV_WORD == av) {
1577             *pos = la;

```

```

1578         break;
1579     }
1581     if (ARGV_EOLN == av)
1582         break;
1583     if (ARGV_ARG == av)
1584         continue;
1586     mdoc_argv_free(arg);
1587     return(0);
1588 }
1590 for (flushed = j = 0; ; ) {
1591     la = *pos;
1592     ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
1593     ac = mdoc_args(m, line, pos, buf, tok, &p);
1594     if (ARGS_ERROR == ac)
1595         return(0);
1596     if (ARGS_PUNCT == ac)
1597         break;
1598     if (ARGS_EOLN == ac)
1599         break;
1601     if ( ! (MDOC_IGNDELIM & mdoc_macros[tok].flags) &&
1602           ARGS_QWORD != ac && 0 == j &&
1603           DELIM_OPEN == mdoc_isdelim(p)) {
1604         if ( ! dword(mdoc, line, la, p, DELIM_OPEN, 0))
1546         if ( ! dword(m, line, la, p, DELIM_OPEN))
1605             return(0);
1606         continue;
1607     } else if (0 == j)
1608         if ( ! mdoc_elem_alloc(mdoc, line, la, tok, arg))
1550         if ( ! mdoc_elem_alloc(m, line, la, tok, arg))
1609             return(0);
1611     if (j == maxargs && ! flushed) {
1612         if ( ! rew_elem(mdoc, tok))
1554         if ( ! rew_elem(m, tok))
1613             return(0);
1614         flushed = 1;
1615     }
1617     ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);
1619     if (MDOC_MAX != ntok) {
1620         if ( ! flushed && ! rew_elem(mdoc, tok))
1562         if ( ! flushed && ! rew_elem(m, tok))
1621             return(0);
1622         flushed = 1;
1623         if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
1565         if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1624             return(0);
1625         j++;
1626         break;
1627     }
1629     if ( ! (MDOC_IGNDELIM & mdoc_macros[tok].flags) &&
1630           ARGS_QWORD != ac &&
1631           ! flushed &&
1632           DELIM_NONE != mdoc_isdelim(p)) {
1633         if ( ! rew_elem(mdoc, tok))
1575         if ( ! rew_elem(m, tok))
1634             return(0);
1635         flushed = 1;
1636     }

```

```

1638         if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1639                   MDOC_JOIN & mdoc_macros[tok].flags))
1580         if ( ! dword(m, line, la, p, DELIM_MAX))
1640             return(0);
1641         j++;
1642     }
1644     if (0 == j && ! mdoc_elem_alloc(mdoc, line, la, tok, arg))
1585     if (0 == j && ! mdoc_elem_alloc(m, line, la, tok, arg))
1645         return(0);
1647     /* Close out in a consistent state. */
1649     if ( ! flushed && ! rew_elem(mdoc, tok))
1590     if ( ! flushed && ! rew_elem(m, tok))
1650         return(0);
1651     if ( ! nl)
1652         return(1);
1653     return(append_delims(mdoc, line, pos, buf));
1594     return(append_delims(m, line, pos, buf));
1654 }
1657 static int
1658 in_line_eoln(MACRO_PROT_ARGS)
1659 {
1660     int             la;
1661     enum margserr   ac;
1662     enum margverr   av;
1663     struct mdoc_arg *arg;
1664     char            *p;
1665     enum mdoct      ntok;
1667     assert( ! (MDOC_PARSED & mdoc_macros[tok].flags));
1669     if (tok == MDOC_Pp)
1670         rew_sub(MDOC_BLOCK, mdoc, MDOC_Nm, line, ppos);
1611     rew_sub(MDOC_BLOCK, m, MDOC_Nm, line, ppos);
1672     /* Parse macro arguments. */
1674     for (arg = NULL; ; ) {
1675         la = *pos;
1676         av = mdoc_argv(mdoc, line, tok, &arg, pos, buf);
1617         av = mdoc_argv(m, line, tok, &arg, pos, buf);
1678         if (ARGV_WORD == av) {
1679             *pos = la;
1680             break;
1681         }
1682         if (ARGV_EOLN == av)
1683             break;
1684         if (ARGV_ARG == av)
1685             continue;
1687         mdoc_argv_free(arg);
1688         return(0);
1689     }
1691     /* Open element scope. */
1693     if ( ! mdoc_elem_alloc(mdoc, line, ppos, tok, arg))
1634     if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
1694         return(0);

```

```

1696     /* Parse argument terms. */
1698     for (;;) {
1699         la = *pos;
1700         ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
1641         ac = mdoc_args(m, line, pos, buf, tok, &p);
1702
1703         if (ARGS_ERROR == ac)
1704             return(0);
1704         if (ARGS_EOLN == ac)
1705             break;
1707
1708         ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);
1709
1710         if (MDOC_MAX == ntok) {
1711             if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1712                 MDOC_JOIN & mdoc_macros[tok].flags))
1651             if ( ! dword(m, line, la, p, DELIM_MAX))
1712                 return(0);
1713             continue;
1714         }
1716         if ( ! rew_elem(mdoc, tok))
1656         if ( ! rew_elem(m, tok))
1717             return(0);
1718         return(mdoc_macro(mdoc, ntok, line, la, pos, buf));
1658         return(mdoc_macro(m, ntok, line, la, pos, buf));
1719     }
1721     /* Close out (no delimiters). */
1723     return(rew_elem(mdoc, tok));
1663     return(rew_elem(m, tok));
1724 }
1727 /* ARGSUSED */
1728 static int
1729 ctx_synopsis(MACRO_PROT_ARGS)
1730 {
1731     int         nl;
1733     nl = MDOC_NEWLINE & mdoc->flags;
1673     nl = MDOC_NEWLINE & m->flags;
1735     /* If we're not in the SYNOPSIS, go straight to in-line. */
1736     if ( ! (MDOC_SYNOPSIS & mdoc->flags))
1737         return(in_line(mdoc, tok, line, ppos, pos, buf));
1676     if ( ! (MDOC_SYNOPSIS & m->flags))
1677         return(in_line(m, tok, line, ppos, pos, buf));
1739     /* If we're a nested call, same place. */
1740     if ( ! nl)
1741         return(in_line(mdoc, tok, line, ppos, pos, buf));
1681         return(in_line(m, tok, line, ppos, pos, buf));
1743     /*
1744     * XXX: this will open a block scope; however, if later we end
1745     * up formatting the block scope, then child nodes will inherit
1746     * the formatting. Be careful.
1747     */
1748     if (MDOC_Nm == tok)
1749         return(blk_full(mdoc, tok, line, ppos, pos, buf));
1689         return(blk_full(m, tok, line, ppos, pos, buf));
1750     assert(MDOC_Vt == tok);
1751     return(blk_part_imp(mdoc, tok, line, ppos, pos, buf));

```

```

1691     return(blk_part_imp(m, tok, line, ppos, pos, buf));
1752 }
1755 /* ARGSUSED */
1756 static int
1757 obsolete(MACRO_PROT_ARGS)
1758 {
1760     mdoc_pmsg(mdoc, line, ppos, MANDOCERR_MACROOBS);
1700     mdoc_pmsg(m, line, ppos, MANDOCERR_MACROOBS);
1761     return(1);
1762 }
1765 /*
1766 * Phrases occur within 'Bl -column' entries, separated by 'Ta' or tabs.
1767 * They're unusual because they're basically free-form text until a
1768 * macro is encountered.
1769 */
1770 static int
1771 phrase(struct mdoc *mdoc, int line, int ppos, char *buf)
1711 phrase(struct mdoc *m, int line, int ppos, char *buf)
1772 {
1773     int             la, pos;
1774     enum margserrr ac;
1775     enum mdoct      ntok;
1776     char            *p;
1778     for (pos = ppos; ; ) {
1779         la = pos;
1781         ac = mdoc_zargs(mdoc, line, &pos, buf, &p);
1721         ac = mdoc_zargs(m, line, &pos, buf, &p);
1783         if (ARGS_ERROR == ac)
1784             return(0);
1785         if (ARGS_EOLN == ac)
1786             break;
1788         ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup_raw(p);
1790         if (MDOC_MAX == ntok) {
1791             if ( ! dword(mdoc, line, la, p, DELIM_MAX, 1))
1731             if ( ! dword(m, line, la, p, DELIM_MAX))
1792                 return(0);
1793             continue;
1794         }
1796         if ( ! mdoc_macro(mdoc, ntok, line, la, &pos, buf))
1736         if ( ! mdoc_macro(m, ntok, line, la, &pos, buf))
1797             return(0);
1798         return(append_delims(mdoc, line, &pos, buf));
1738         return(append_delims(m, line, &pos, buf));
1799     }
1801     return(1);
1802 }
1805 /* ARGSUSED */
1806 static int
1807 phrase_ta(MACRO_PROT_ARGS)
1808 {
1809     struct mdoc_node *n;
1810     int             la;

```

```

1811     enum mdoct      ntok;
1812     enum margserr   ac;
1813     char            *p;

1815     /* Make sure we are in a column list or ignore this macro. */
1816     n = mdoc->last;
1817     while (NULL != n && MDOC_B1 != n->tok)
1818         n = n->parent;
1819     if (NULL == n || LIST_column != n->norm->B1.type) {
1820         mdoc_pmsg(mdoc, line, ppos, MANDOCERR_STRAYTA);
1821         return(1);
1822     }
1823     /*
1824     * FIXME: this is overly restrictive: if the 'Ta' is unexpected,
1825     * it should simply error out with ARGSLOST.
1826     */

1824     /* Advance to the next column. */
1825     if ( ! rew_sub(MDOC_BODY, mdoc, MDOC_It, line, ppos))
1826         if ( ! rew_sub(MDOC_BODY, m, MDOC_It, line, ppos))
1827             return(0);
1828     if ( ! mdoc_body_alloc(mdoc, line, ppos, MDOC_It))
1829         if ( ! mdoc_body_alloc(m, line, ppos, MDOC_It))
1830             return(0);

1830     for (;;) {
1831         la = *pos;
1832         ac = mdoc_zargs(mdoc, line, pos, buf, &p);
1833         ac = mdoc_zargs(m, line, pos, buf, &p);

1834         if (ARGS_ERROR == ac)
1835             return(0);
1836         if (ARGS_EOLN == ac)
1837             break;

1839         ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup_raw(p);

1841         if (MDOC_MAX == ntok) {
1842             if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1843                 MDOC_JOIN & mdoc_macros[ntok].flags))
1844                 if ( ! dword(m, line, la, p, DELIM_MAX))
1845                     return(0);
1846             continue;
1847         }

1848         if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
1849             if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1850                 return(0);
1851         return(append_delims(mdoc, line, pos, buf));
1852         return(append_delims(m, line, pos, buf));
1853     }
1854 }

```

unchanged\_portion\_omitted

```

*****
33223 Fri Jul 25 15:00:14 2014
new/usr/src/cmd/mandoc/mdoc_man.c
Update to 1.12.3.
*****
1 /* $Id: mdoc_man.c,v 1.57 2013/12/25 22:00:45 schwarze Exp $ */
1 /* $Id: mdoc_man.c,v 1.9 2011/10/24 21:47:59 schwarze Exp $ */
2 /*
3  * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif

21 #include <assert.h>
22 #include <stdio.h>
23 #include <string.h>

25 #include "mandoc.h"
26 #include "out.h"
27 #include "man.h"
28 #include "mdoc.h"
29 #include "main.h"

31 #define DECL_ARGS const struct mdoc_meta *meta, \
32                 const struct mdoc_node *n
29 #define DECL_ARGS const struct mdoc_meta *m, \
30                 const struct mdoc_node *n, \
31                 struct mman *mm

33 struct mman {
34     int          need_space; /* next word needs prior ws */
35     int          need_nl; /* next word needs prior nl */
36 };

34 struct manact {
35     int          (*cond)(DECL_ARGS); /* DON'T run actions */
36     int          (*pre)(DECL_ARGS); /* pre-node action */
37     void         (*post)(DECL_ARGS); /* post-node action */
38     const char   *prefix; /* pre-node string constant */
39     const char   *suffix; /* post-node string constant */
40 };

42 static int      cond_body(DECL_ARGS);
43 static int      cond_head(DECL_ARGS);
44 static void     font_push(char);
45 static void     font_pop(void);
46 static void     mid_it(void);
47 static void     post__t(DECL_ARGS);
48 static void     post_bd(DECL_ARGS);
49 static void     post_bf(DECL_ARGS);
50 static void     post_bk(DECL_ARGS);
51 static void     post_bl(DECL_ARGS);

```

```

52 static void     post_dl(DECL_ARGS);
53 static void     post_enc(DECL_ARGS);
54 static void     post_eo(DECL_ARGS);
55 static void     post_fa(DECL_ARGS);
56 static void     post_fd(DECL_ARGS);
57 static void     post_fl(DECL_ARGS);
58 static void     post_fn(DECL_ARGS);
59 static void     post_fo(DECL_ARGS);
60 static void     post_font(DECL_ARGS);
61 static void     post_in(DECL_ARGS);
62 static void     post_it(DECL_ARGS);
63 static void     post_lb(DECL_ARGS);
64 static void     post_nm(DECL_ARGS);
65 static void     post_percent(DECL_ARGS);
66 static void     post_pf(DECL_ARGS);
67 static void     post_sect(DECL_ARGS);
68 static void     post_sp(DECL_ARGS);
69 static void     post_vt(DECL_ARGS);
70 static int      pre__t(DECL_ARGS);
71 static int      pre_an(DECL_ARGS);
72 static int      pre_ap(DECL_ARGS);
73 static int      pre_bd(DECL_ARGS);
74 static int      pre_bf(DECL_ARGS);
75 static int      pre_bk(DECL_ARGS);
76 static int      pre_bl(DECL_ARGS);
77 static int      pre_br(DECL_ARGS);
78 static int      pre_bx(DECL_ARGS);
79 static int      pre_dl(DECL_ARGS);
80 static int      pre_enc(DECL_ARGS);
81 static int      pre_em(DECL_ARGS);
82 static int      pre_fa(DECL_ARGS);
83 static int      pre_fd(DECL_ARGS);
84 static int      pre_fl(DECL_ARGS);
85 static int      pre_fn(DECL_ARGS);
86 static int      pre_fo(DECL_ARGS);
87 static int      pre_ft(DECL_ARGS);
88 static int      pre_in(DECL_ARGS);
89 static int      pre_it(DECL_ARGS);
90 static int      pre_lk(DECL_ARGS);
91 static int      pre_li(DECL_ARGS);
92 static int      pre_nm(DECL_ARGS);
93 static int      pre_no(DECL_ARGS);
94 static int      pre_ns(DECL_ARGS);
95 static int      pre_pp(DECL_ARGS);
96 static int      pre_rs(DECL_ARGS);
97 static int      pre_sm(DECL_ARGS);
98 static int      pre_sp(DECL_ARGS);
99 static int      pre_sect(DECL_ARGS);
100 static int      pre_sy(DECL_ARGS);
101 static void     pre_syn(const struct mdoc_node *);
102 static int      pre_vt(DECL_ARGS);
103 static int      pre_ux(DECL_ARGS);
104 static int      pre_xr(DECL_ARGS);
105 static void     print_word(const char *);
106 static void     print_line(const char *, int);
107 static void     print_block(const char *, int);
108 static void     print_offs(const char *);
109 static void     print_width(const char *,
110                          const struct mdoc_node *, size_t);
111 static void     print_count(int *);
70 static void     print_word(struct mman *, const char *);
112 static void     print_node(DECL_ARGS);

114 static const struct manact manacts[MDOC_MAX + 1] = {
115     { NULL, pre_ap, NULL, NULL, NULL }, /* Ap */
116     { NULL, NULL, NULL, NULL, NULL }, /* Dd */

```



```

117 { NULL, NULL, NULL, NULL, NULL }, /* Dt */
118 { NULL, NULL, NULL, NULL, NULL }, /* Os */
119 { NULL, pre_sect, post_sect, ".SH", NULL }, /* Sh */
120 { NULL, pre_sect, post_sect, ".SS", NULL }, /* Ss */
121 { NULL, pre_pp, NULL, NULL, NULL }, /* Pp */
122 { cond_body, pre_dl, post_dl, NULL, NULL }, /* D1 */
123 { cond_body, pre_dl, post_dl, NULL, NULL }, /* D1 */
124 { cond_body, pre_bd, post_bd, NULL, NULL }, /* Bd */
125 { NULL, NULL, NULL, NULL, NULL }, /* Ed */
126 { cond_body, pre_bl, post_bl, NULL, NULL }, /* Bl */
85 { NULL, NULL, NULL, NULL, NULL }, /* B1 */
127 { NULL, NULL, NULL, NULL, NULL }, /* E1 */
128 { NULL, pre_it, post_it, NULL, NULL }, /* It */
129 { NULL, pre_em, post_font, NULL, NULL }, /* Ad */
130 { NULL, pre_an, NULL, NULL, NULL }, /* An */
131 { NULL, pre_em, post_font, NULL, NULL }, /* Ar */
132 { NULL, pre_sy, post_font, NULL, NULL }, /* Cd */
133 { NULL, pre_sy, post_font, NULL, NULL }, /* Cm */
134 { NULL, pre_li, post_font, NULL, NULL }, /* Dv */
135 { NULL, pre_li, post_font, NULL, NULL }, /* Er */
136 { NULL, pre_li, post_font, NULL, NULL }, /* Ev */
87 { NULL, pre_it, NULL, NULL, NULL }, /* _It */
88 { NULL, pre_enc, post_enc, "\\fi", "\\fP" }, /* Ad */
89 { NULL, NULL, NULL, NULL, NULL }, /* _An */
90 { NULL, pre_enc, post_enc, "\\fi", "\\fP" }, /* Ar */
91 { NULL, pre_enc, post_enc, "\\fb", "\\fP" }, /* Cd */
92 { NULL, pre_enc, post_enc, "\\fb", "\\fP" }, /* Cm */
93 { NULL, pre_enc, post_enc, "\\fr", "\\fP" }, /* Dv */
94 { NULL, pre_enc, post_enc, "\\fr", "\\fP" }, /* Er */
95 { NULL, pre_enc, post_enc, "\\fr", "\\fP" }, /* Ev */
137 { NULL, pre_enc, post_enc, "The \\fb",
138 "\\fP\nutility exits 0 on success, and >0 if an error occurs."
139 }, /* Ex */
140 { NULL, pre_fa, post_fa, NULL, NULL }, /* Fa */
141 { NULL, pre_fd, post_fd, NULL, NULL }, /* Fd */
142 { NULL, pre_fl, post_fl, NULL, NULL }, /* Fl */
143 { NULL, pre_fn, post_fn, NULL, NULL }, /* Fn */
144 { NULL, pre_ft, post_font, NULL, NULL }, /* Ft */
145 { NULL, pre_sy, post_font, NULL, NULL }, /* Ic */
146 { NULL, pre_in, post_in, NULL, NULL }, /* In */
147 { NULL, pre_li, post_font, NULL, NULL }, /* Li */
99 { NULL, NULL, NULL, NULL, NULL }, /* _Fa */
100 { NULL, NULL, NULL, NULL, NULL }, /* _Fd */
101 { NULL, pre_enc, post_enc, "\\fb-", "\\fP" }, /* Fl */
102 { NULL, NULL, NULL, NULL, NULL }, /* _Fn */
103 { NULL, NULL, NULL, NULL, NULL }, /* _Ft */
104 { NULL, pre_enc, post_enc, "\\fb", "\\fP" }, /* Ic */
105 { NULL, NULL, NULL, NULL, NULL }, /* _In */
106 { NULL, pre_enc, post_enc, "\\fr", "\\fP" }, /* Li */
148 { cond_head, pre_enc, NULL, "\\-", NULL }, /* Nd */
149 { NULL, pre_nm, post_nm, NULL, NULL }, /* Nm */
150 { cond_body, pre_enc, post_enc, "[", "]" }, /* Op */
151 { NULL, NULL, NULL, NULL, NULL }, /* Ot */
152 { NULL, pre_em, post_font, NULL, NULL }, /* Pa */
111 { NULL, pre_enc, post_enc, "\\fi", "\\fP" }, /* Pa */
153 { NULL, pre_enc, post_enc, "The \\fb",
154 "\\fP\nfunction returns the value 0 if successful;\n"
155 "otherwise the value -1 is returned and the global\n"
156 "variable \\fierrno\\fP is set to indicate the error."
157 }, /* Rv */
158 { NULL, NULL, NULL, NULL, NULL }, /* St */
159 { NULL, pre_em, post_font, NULL, NULL }, /* Va */
160 { NULL, pre_vt, post_vt, NULL, NULL }, /* Vt */
118 { NULL, NULL, NULL, NULL, NULL }, /* _Va */
119 { NULL, NULL, NULL, NULL, NULL }, /* _Vt */
161 { NULL, pre_xr, NULL, NULL, NULL }, /* Xr */

```

```

162 { NULL, NULL, post_percent, NULL, NULL }, /* %A */
163 { NULL, pre_em, post_percent, NULL, NULL }, /* %B */
164 { NULL, NULL, post_percent, NULL, NULL }, /* %D */
165 { NULL, pre_em, post_percent, NULL, NULL }, /* %I */
166 { NULL, pre_em, post_percent, NULL, NULL }, /* %J */
167 { NULL, NULL, post_percent, NULL, NULL }, /* %N */
168 { NULL, NULL, post_percent, NULL, NULL }, /* %O */
169 { NULL, NULL, post_percent, NULL, NULL }, /* %P */
170 { NULL, NULL, post_percent, NULL, NULL }, /* %R */
171 { NULL, pre_t, post_t, NULL, NULL }, /* %T */
172 { NULL, NULL, post_percent, NULL, NULL }, /* %V */
121 { NULL, NULL, post_percent, NULL, NULL }, /* _%A */
122 { NULL, NULL, NULL, NULL, NULL }, /* _%B */
123 { NULL, NULL, post_percent, NULL, NULL }, /* _%D */
124 { NULL, NULL, NULL, NULL, NULL }, /* _%I */
125 { NULL, pre_enc, post_percent, "\\fi", "\\fP" }, /* %J */
126 { NULL, NULL, NULL, NULL, NULL }, /* _%N */
127 { NULL, NULL, NULL, NULL, NULL }, /* _%O */
128 { NULL, NULL, NULL, NULL, NULL }, /* _%P */
129 { NULL, NULL, NULL, NULL, NULL }, /* _%R */
130 { NULL, pre_enc, post_percent, "", "" }, /* %T */
131 { NULL, NULL, NULL, NULL, NULL }, /* _%V */
173 { NULL, NULL, NULL, NULL, NULL }, /* Ac */
174 { cond_body, pre_enc, post_enc, "<", ">" }, /* Ao */
175 { cond_body, pre_enc, post_enc, "<", ">" }, /* Aq */
176 { NULL, NULL, NULL, NULL, NULL }, /* At */
177 { NULL, NULL, NULL, NULL, NULL }, /* Bc */
178 { NULL, pre_bf, post_bf, NULL, NULL }, /* Bf */
137 { NULL, NULL, NULL, NULL, NULL }, /* _Bf */
179 { cond_body, pre_enc, post_enc, "[", "]" }, /* Bo */
180 { cond_body, pre_enc, post_enc, "[", "]" }, /* Bq */
181 { NULL, pre_ux, NULL, "BSD/OS", NULL }, /* Bsx */
182 { NULL, pre_bx, NULL, NULL, NULL }, /* Bx */
183 { NULL, NULL, NULL, NULL, NULL }, /* Db */
184 { NULL, NULL, NULL, NULL, NULL }, /* Dc */
185 { cond_body, pre_enc, post_enc, "\\(lq", "\\(rq" }, /* Do */
186 { cond_body, pre_enc, post_enc, "\\(lq", "\\(rq" }, /* Dq */
187 { NULL, NULL, NULL, NULL, NULL }, /* Ec */
188 { NULL, NULL, NULL, NULL, NULL }, /* Ef */
189 { NULL, pre_em, post_font, NULL, NULL }, /* Em */
190 { NULL, NULL, post_eo, NULL, NULL }, /* Eo */
144 { cond_body, pre_enc, post_enc, "'", "'" }, /* Do */
145 { cond_body, pre_enc, post_enc, "'", "'" }, /* Dq */
146 { NULL, NULL, NULL, NULL, NULL }, /* Ec */
147 { NULL, NULL, NULL, NULL, NULL }, /* Ef */
148 { NULL, pre_enc, post_enc, "\\fi", "\\fP" }, /* Em */
149 { NULL, NULL, NULL, NULL, NULL }, /* Eo */
191 { NULL, pre_ux, NULL, "FreeBSD", NULL }, /* Fx */
192 { NULL, pre_sy, post_font, NULL, NULL }, /* Ms */
193 { NULL, pre_no, NULL, NULL, NULL }, /* No */
151 { NULL, pre_enc, post_enc, "\\fb", "\\fP" }, /* Ms */
152 { NULL, NULL, NULL, NULL, NULL }, /* No */
194 { NULL, pre_ns, NULL, NULL, NULL }, /* Ns */
195 { NULL, pre_ux, NULL, "NetBSD", NULL }, /* Nx */
196 { NULL, pre_ux, NULL, "OpenBSD", NULL }, /* Ox */
197 { NULL, NULL, NULL, NULL, NULL }, /* Pc */
198 { NULL, NULL, post_pf, NULL, NULL }, /* Pf */
199 { cond_body, pre_enc, post_enc, "(", ")" }, /* Po */
200 { cond_body, pre_enc, post_enc, "(", ")" }, /* Pq */
201 { NULL, NULL, NULL, NULL, NULL }, /* Qc */
202 { cond_body, pre_enc, post_enc, "\\(oq", "\\(cq" }, /* Ql */
161 { cond_body, pre_enc, post_enc, "'", "'" }, /* Ql */
203 { cond_body, pre_enc, post_enc, "\"", "\"" }, /* Qo */
204 { cond_body, pre_enc, post_enc, "\"", "\"" }, /* Qq */
205 { NULL, NULL, NULL, NULL, NULL }, /* Re */
206 { cond_body, pre_rs, NULL, NULL, NULL }, /* Rs */

```

```

165 { cond_body, pre_pp, NULL, NULL, NULL }, /* Rs */
166 { NULL, NULL, NULL, NULL, NULL }, /* Sc */
167 { cond_body, pre_enc, post_enc, "\\(cq", "\\(cq" }, /* So */
168 { cond_body, pre_enc, post_enc, "\\(cq", "\\(cq" }, /* Sq */
169 { NULL, pre_sm, NULL, NULL, NULL }, /* Sm */
170 { NULL, pre_em, post_font, NULL, NULL }, /* Sx */
171 { NULL, pre_sy, post_font, NULL, NULL }, /* Sy */
172 { NULL, pre_li, post_font, NULL, NULL }, /* Tn */
173 { cond_body, pre_enc, post_enc, "\"", "\"" }, /* So */
174 { cond_body, pre_enc, post_enc, "\"", "\"" }, /* Sq */
175 { NULL, NULL, NULL, NULL, NULL }, /* _Sm */
176 { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Sx */
177 { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Sy */
178 { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Tn */
179 { NULL, pre_ux, NULL, "UNIX", NULL }, /* Ux */
180 { NULL, NULL, NULL, NULL, NULL }, /* Xc */
181 { NULL, NULL, NULL, NULL, NULL }, /* Xo */
182 { NULL, pre_fo, post_fo, NULL, NULL }, /* Fo */
183 { NULL, NULL, NULL, NULL, NULL }, /* Fc */
184 { NULL, NULL, NULL, NULL, NULL }, /* _Xc */
185 { NULL, NULL, NULL, NULL, NULL }, /* _Xo */
186 { NULL, NULL, NULL, NULL, NULL }, /* _Fo */
187 { NULL, NULL, NULL, NULL, NULL }, /* _Fc */
188 { cond_body, pre_enc, post_enc, "[", "]" }, /* Oo */
189 { NULL, NULL, NULL, NULL, NULL }, /* Oc */
190 { NULL, pre_bk, post_bk, NULL, NULL }, /* Bk */
191 { NULL, NULL, NULL, NULL, NULL }, /* Ek */
192 { NULL, NULL, NULL, NULL, NULL }, /* _Bk */
193 { NULL, NULL, NULL, NULL, NULL }, /* _Ek */
194 { NULL, pre_ux, NULL, "is currently in beta test.", NULL }, /* Bt */
195 { NULL, NULL, NULL, NULL, NULL }, /* Hf */
196 { NULL, NULL, NULL, NULL, NULL }, /* Fr */
197 { NULL, pre_ux, NULL, "currently under development.", NULL }, /* Ud */
198 { NULL, NULL, post_lb, NULL, NULL }, /* Lb */
199 { NULL, NULL, NULL, NULL, NULL }, /* _Lb */
200 { NULL, pre_pp, NULL, NULL, NULL }, /* Lp */
201 { NULL, pre_lk, NULL, NULL, NULL }, /* Lk */
202 { NULL, pre_em, post_font, NULL, NULL }, /* Mt */
203 { NULL, NULL, NULL, NULL, NULL }, /* _Lk */
204 { NULL, NULL, NULL, NULL, NULL }, /* _Mt */
205 { cond_body, pre_enc, post_enc, "{", "}" }, /* Brq */
206 { cond_body, pre_enc, post_enc, "{", "}" }, /* Bro */
207 { NULL, NULL, NULL, NULL, NULL }, /* Brc */
208 { NULL, NULL, post_percent, NULL, NULL }, /* %C */
209 { NULL, NULL, NULL, NULL, NULL }, /* Es */
210 { NULL, NULL, NULL, NULL, NULL }, /* En */
211 { NULL, NULL, NULL, NULL, NULL }, /* _%C */
212 { NULL, NULL, NULL, NULL, NULL }, /* _Es */
213 { NULL, NULL, NULL, NULL, NULL }, /* _En */
214 { NULL, pre_ux, NULL, "DragonFly", NULL }, /* Dx */
215 { NULL, NULL, post_percent, NULL, NULL }, /* %Q */
216 { NULL, NULL, NULL, NULL, NULL }, /* %Q */
217 { NULL, pre_br, NULL, NULL, NULL }, /* br */
218 { NULL, pre_sp, post_sp, NULL, NULL }, /* sp */
219 { NULL, NULL, post_percent, NULL, NULL }, /* %U */
220 { NULL, NULL, NULL, NULL, NULL }, /* Ta */
221 { NULL, NULL, NULL, NULL, NULL }, /* %U */
222 { NULL, NULL, NULL, NULL, NULL }, /* _Ta */
223 { NULL, NULL, NULL, NULL, NULL }, /* ROOT */
224 };

225 static int outflags;
226 #define MMAN_spc (1 << 0) /* blank character before next word */
227 #define MMAN_spc_force (1 << 1) /* even before trailing punctuation */
228 #define MMAN_nl (1 << 2) /* break man(7) code line */
229 #define MMAN_br (1 << 3) /* break output line */

```

```

230 #define MMAN_sp (1 << 4) /* insert a blank output line */
231 #define MMAN_PP (1 << 5) /* reset indentation etc. */
232 #define MMAN_Sm (1 << 6) /* horizontal spacing mode */
233 #define MMAN_Bk (1 << 7) /* word keep mode */
234 #define MMAN_Bk_susp (1 << 8) /* suspend this (after a macro) */
235 #define MMAN_An_split (1 << 9) /* author mode is "split" */
236 #define MMAN_An_nosplit (1 << 10) /* author mode is "nosplit" */
237 #define MMAN_PD (1 << 11) /* inter-paragraph spacing disabled */
238 #define MMAN_nbrword (1 << 12) /* do not break the next word */

239 #define BL_STACK_MAX 32

240 static size_t Bl_stack[BL_STACK_MAX]; /* offsets [chars] */
241 static int Bl_stack_post[BL_STACK_MAX]; /* add final .RE */
242 static int Bl_stack_len; /* number of nested Bl blocks */
243 static int TPremain; /* characters before tag is full */

244 static struct {
245     char *head;
246     char *tail;
247     size_t size;
248 } fontqueue;

249 static void
250 font_push(char newfont)
251 {
252     if (fontqueue.head + fontqueue.size <= ++fontqueue.tail) {
253         fontqueue.size += 8;
254         fontqueue.head = mandoc_realloc(fontqueue.head,
255                                         fontqueue.size);
256     }
257     *fontqueue.tail = newfont;
258     print_word("");
259     printf("\\f");
260     putchar(newfont);
261     outflags &= ~MMAN_spc;
262 }

263 static void
264 font_pop(void)
265 {
266     if (fontqueue.tail > fontqueue.head)
267         fontqueue.tail--;
268     outflags &= ~MMAN_spc;
269     print_word("");
270     printf("\\f");
271     putchar(*fontqueue.tail);
272 }

273 static void
274 print_word(const char *s)
275 {
276     if ((MMAN_PP | MMAN_sp | MMAN_br | MMAN_nl) & outflags) {
277         if (mm->need_nl) {
278             /*
279              * If we need a newline, print it now and start afresh.
280              */
281             if (MMAN_PP & outflags) {
282                 if (MMAN_sp & outflags) {
283                     if (MMAN_PD & outflags) {
284                         printf("\n.PD");
285                         outflags &= ~MMAN_PD;
286                     }
287                 }
288             }
289         }
290     }

```

```

315     }
316     } else if ( ! (MMAN_PD & outflags)) {
317         printf("\n.PD 0");
318         outflags |= MMAN_PD;
319     }
320     printf("\n.PP\n");
321 } else if (MMAN_sp & outflags)
322     printf("\n.sp\n");
323 else if (MMAN_br & outflags)
324     printf("\n.br\n");
325 else if (MMAN_nl & outflags)
326     putchar('\n');
327 outflags &= ~(MMAN_PP|MMAN_sp|MMAN_br|MMAN_nl|MMAN_spc);
328 if (1 == TPremain)
329     printf(".br\n");
330 TPremain = 0;
331 } else if (MMAN_spc & outflags) {
332     mm->need_space = 0;
333     mm->need_nl = 0;
334 } else if (mm->need_space && '\0' != s[0])
335     /*
336     * If we need a space, only print it if
337     * (1) it is forced by 'No' or
338     * (2) what follows is not terminating punctuation or
339     * (3) what follows is longer than one character.
340     * If we need a space, only print it before
341     * (1) a nonzero length word;
342     * (2) a word that is non-punctuation; and
343     * (3) if punctuation, non-terminating punctation.
344     */
345 if (MMAN_spc_force & outflags || '\0' == s[0] ||
346     NULL == strchr(".,:;]?!", s[0]) || '\0' != s[1]) {
347     if (MMAN_Bk & outflags &&
348         ! (MMAN_Bk_susp & outflags))
349         putchar('\ ');
350 if (NULL == strchr(".,:;]?!", s[0]) || '\0' != s[1])
351     putchar(' ');
352 if (TPremain)
353     TPremain--;
354 }
355 }
356
357 /*
358 * Reassign needing space if we're not following opening
359 * punctuation.
360 */
361 if (MMAN_Sm & outflags && ('\0' == s[0] ||
362     ((' ' != s[0] && '[' != s[0]) || '\0' != s[1])))
363     outflags |= MMAN_spc;
364 else
365     outflags &= ~MMAN_spc;
366 outflags &= ~(MMAN_spc_force | MMAN_Bk_susp);
367 mm->need_space =
368     ((' ' != s[0] && '[' != s[0]) || '\0' != s[1];
369
370 for ( ; *s; s++) {
371     switch (*s) {
372     case (ASCII_NBRSP):
373         printf("\ \ ");
374         printf("\ \~");
375         break;
376     case (ASCII_HYPH):
377         putchar('-');
378         break;
379     case (' '):
380         if (MMAN_nbrword & outflags) {

```

```

370         printf("\ \ ");
371         break;
372     }
373     /* FALLTHROUGH */
374     default:
375         putchar((unsigned char)*s);
376         break;
377     }
378     if (TPremain)
379         TPremain--;
380 }
381 outflags &= ~MMAN_nbrword;
382 }
383
384 static void
385 print_line(const char *s, int newflags)
386 {
387
388     outflags &= ~MMAN_br;
389     outflags |= MMAN_nl;
390     print_word(s);
391     outflags |= newflags;
392 }
393
394 static void
395 print_block(const char *s, int newflags)
396 {
397
398     outflags &= ~MMAN_PP;
399     if (MMAN_sp & outflags) {
400         outflags &= ~(MMAN_sp | MMAN_br);
401         if (MMAN_PD & outflags) {
402             print_line(".PD", 0);
403             outflags &= ~MMAN_PD;
404         }
405     } else if ( ! (MMAN_PD & outflags))
406         print_line(".PD 0", MMAN_PD);
407     outflags |= MMAN_nl;
408     print_word(s);
409     outflags |= MMAN_Bk_susp | newflags;
410 }
411
412 static void
413 print_offs(const char *v)
414 {
415     char          buf[24];
416     struct roffsu su;
417     size_t        sz;
418
419     print_line(".RS", MMAN_Bk_susp);
420
421     /* Convert v into a number (of characters). */
422     if (NULL == v || '\0' == *v || 0 == strcmp(v, "left"))
423         sz = 0;
424     else if (0 == strcmp(v, "indent"))
425         sz = 6;
426     else if (0 == strcmp(v, "indent-two"))
427         sz = 12;
428     else if (a2roffsu(v, &su, SCALE_MAX)) {
429         if (SCALE_EN == su.unit)
430             sz = su.scale;
431     } else {
432         /*
433          * XXX
434          * If we are inside an enclosing list,
435          * there is no easy way to add the two

```

```

436     * indentations because they are provided
437     * in terms of different units.
438     */
439     print_word(v);
440     outflags |= MMAN_nl;
441     return;
442 }
443 } else
444     sz = strlen(v);
445
446 /*
447  * We are inside an enclosing list.
448  * Add the two indentations.
449  */
450 if (Bl_stack_len)
451     sz += Bl_stack[Bl_stack_len - 1];
452
453 snprintf(buf, sizeof(buf), "%zun", sz);
454 print_word(buf);
455 outflags |= MMAN_nl;
456 }
457
458 /*
459  * Set up the indentation for a list item; used from pre_it().
460  */
461 void
462 print_width(const char *v, const struct mdoc_node *child, size_t defsz)
463 {
464     char          buf[24];
465     struct roffsu su;
466     size_t        sz, chsz;
467     int           numeric, remain;
468
469     numeric = 1;
470     remain = 0;
471
472     /* Convert v into a number (of characters). */
473     if (NULL == v)
474         sz = defsz;
475     else if (a2roffsu(v, &su, SCALE_MAX)) {
476         if (SCALE_EN == su.unit)
477             sz = su.scale;
478         else {
479             sz = 0;
480             numeric = 0;
481         }
482     } else
483         sz = strlen(v);
484
485     /* XXX Rough estimation, might have multiple parts. */
486     chsz = (NULL != child && MDOC_TEXT == child->type) ?
487         strlen(child->string) : 0;
488
489     /* Maybe we are inside an enclosing list? */
490     mid_it();
491
492     /*
493      * Save our own indentation,
494      * such that child lists can use it.
495      */
496     Bl_stack[Bl_stack_len++] = sz + 2;
497
498     /* Set up the current list. */
499     if (defsz && chsz > sz)
500         print_block(".HP", 0);
501     else {

```

```

502         print_block(".TP", 0);
503         remain = sz + 2;
504     }
505     if (numeric) {
506         snprintf(buf, sizeof(buf), "%zun", sz + 2);
507         print_word(buf);
508     } else
509         print_word(v);
510     TPremain = remain;
511 }
512
513 void
514 print_count(int *count)
515 {
516     char          buf[12];
517
518     snprintf(buf, sizeof(buf), "%d.", ++*count);
519     print_word(buf);
520 }
521
522 void
523 man_man(void *arg, const struct man *man)
524 {
525     /*
526      * Dump the keep buffer.
527      * We're guaranteed by now that this exists (is non-NULL).
528      * Flush stdout afterward, just in case.
529      */
530     fputs(mparse_getkeep(man_mparse(man)), stdout);
531     fflush(stdout);
532 }
533
534 void
535 man_mdoc(void *arg, const struct mdoc *mdoc)
536 {
537     const struct mdoc_meta *meta;
538     const struct mdoc_meta *m;
539     const struct mdoc_node *n;
540     struct mman            mm;
541
542     meta = mdoc_meta(mdoc);
543     m = mdoc_meta(mdoc);
544     n = mdoc_node(mdoc);
545
546     printf(".TH \"%s\" \"%s\" \"%s\" \"%s\" \"%s\" \n",
547           meta->title, meta->msec, meta->date,
548           meta->os, meta->vol);
549     printf(".TH \"%s\" \"%s\" \"%s\" \"%s\" \"%s\" \n",
550           m->title, m->msec, m->date, m->os, m->vol);
551
552     /* Disable hyphenation and if nroff, disable justification. */
553     printf(".nh\n.if n .ad l");
554     memset(&mm, 0, sizeof(struct mman));
555
556     outflags = MMAN_nl | MMAN_Sm;
557     if (0 == fontqueue.size) {
558         fontqueue.size = 8;
559         fontqueue.head = fontqueue.tail = mandoc_malloc(8);
560         *fontqueue.tail = 'R';
561     }
562     print_node(meta, n);
563     mm.need_nl = 1;
564     print_node(m, n, &mm);
565     putchar('\n');
566 }

```

```

561 static void
562 print_node(DECL_ARGS)
563 {
564     const struct mdoc_node *sub;
565     const struct mdoc_node *prev, *sub;
566     const struct manact *act;
567     int cond, do_sub;

568     /*
569     * Break the line if we were parsed subsequent the current node.
570     * This makes the page structure be more consistent.
571     */
572     if (MMAN_spc & outflags && MDOC_LINE & n->flags)
573         outflags |= MMAN_nl;
574     prev = n->prev ? n->prev : n->parent;
575     if (prev && prev->line < n->line)
576         mm->need_nl = 1;

577     act = NULL;
578     cond = 0;
579     do_sub = 1;

580     if (MDOC_TEXT == n->type) {
581         /*
582         * Make sure that we don't happen to start with a
583         * control character at the start of a line.
584         */
585         if (MMAN_nl & outflags && ('.' == *n->string ||
586             if (mm->need_nl && ('.' == *n->string ||
587                 '\'' == *n->string)) {
588             print_word("");
589             printf("\\&");
590             outflags &= ~MMAN_spc;
591             print_word(mm, "\\&");
592             mm->need_space = 0;
593         }
594         print_word(n->string);
595         print_word(mm, n->string);
596     } else {
597         /*
598         * Conditionally run the pre-node action handler for a
599         * node.
600         */
601         act = manacts + n->tok;
602         cond = NULL == act->cond || (*act->cond)(meta, n);
603         cond = NULL == act->cond || (*act->cond)(m, n, mm);
604         if (cond && act->pre)
605             do_sub = (*act->pre)(meta, n);
606             do_sub = (*act->pre)(m, n, mm);
607     }

608     /*
609     * Conditionally run all child nodes.
610     * Note that this iterates over children instead of using
611     * recursion. This prevents unnecessary depth in the stack.
612     */
613     if (do_sub)
614         for (sub = n->child; sub; sub = sub->next)
615             print_node(meta, sub);
616             print_node(m, sub, mm);

617     /*
618     * Lastly, conditionally run the post-node handler.
619     */
620     if (cond && act->post)

```

```

621         (*act->post)(meta, n);
622         (*act->post)(m, n, mm);
623     }
624     unchanged_portion_omitted

625 /*
626 * Output a font encoding before a node, e.g., \fR.
627 * This obviously has no trailing space.
628 */
629 static int
630 pre_enc(DECL_ARGS)
631 {
632     const char *prefix;

633     prefix = manacts[n->tok].prefix;
634     if (NULL == prefix)
635         return(1);
636     print_word(prefix);
637     outflags &= ~MMAN_spc;
638     print_word(mm, prefix);
639     mm->need_space = 0;
640     return(1);
641 }

642 /*
643 * Output a font encoding subsequent a node, e.g., \fP.
644 */
645 static void
646 post_enc(DECL_ARGS)
647 {
648     const char *suffix;

649     suffix = manacts[n->tok].suffix;
650     if (NULL == suffix)
651         return;
652     outflags &= ~MMAN_spc;
653     print_word(suffix);
654     mm->need_space = 0;
655     print_word(mm, suffix);
656 }

657 /*
658 * Used in listings (percent = %A, e.g.).
659 * FIXME: this is incomplete.
660 * It doesn't print a nice ", and" for lists.
661 */
662 static void
663 post_font(DECL_ARGS)
664 {
665     font_pop();
666 }

667 static void
668 post_percent(DECL_ARGS)
669 {
670     if (pre_em == manacts[n->tok].pre)
671         font_pop();
672     if (n->next) {
673         print_word(",");
674         if (n->prev && n->prev->tok == n->tok &&
675             n->next->tok == n->tok)
676             print_word("and");
677     } else {
678         print_word(".");
679     }

```

```

677         outflags |= MMAN_nl;
678     }
679 }

681 static int
682 pre_t(DECL_ARGS)
683 {
684     if (n->parent && MDOC_Rs == n->parent->tok &&
685         n->parent->norm->Rs.quote_T) {
686         print_word("");
687         putchar('\n');
688         outflags &= ~MMAN_spc;
689     } else
690         font_push('I');
691     return(1);
692 }

695 static void
696 post_t(DECL_ARGS)
697 {
698     if (n->parent && MDOC_Rs == n->parent->tok &&
699         n->parent->norm->Rs.quote_T) {
700         outflags &= ~MMAN_spc;
701         print_word("");
702         putchar('\n');
703     } else
704         font_pop();
705     post_percent(meta, n);
706 }

709 /*
710  * Print before a section header.
711  */
712 static int
713 pre_sect(DECL_ARGS)
714 {
715     if (MDOC_HEAD == n->type) {
716         outflags |= MMAN_sp;
717         print_block(manacts[n->tok].prefix, 0);
718         print_word("");
719         putchar('\n');
720         outflags &= ~MMAN_spc;
721     }
722     if (MDOC_HEAD != n->type)
723         return(1);
724     mm->need_nl = 1;
725     print_word(mm, manacts[n->tok].prefix);
726     print_word(mm, "\"");
727     mm->need_space = 0;
728     return(1);
729 }

726 /*
727  * Print subsequent a section header.
728  */
729 static void
730 post_sect(DECL_ARGS)

```

```

731 {
732     if (MDOC_HEAD != n->type)
733         return;
734     outflags &= ~MMAN_spc;
735     print_word("");
736     putchar('\n');
737     outflags |= MMAN_nl;
738     if (MDOC_Sh == n->tok && SEC_AUTHORS == n->sec)
739         outflags &= ~(MMAN_An_split | MMAN_An_nosplit);
740     mm->need_space = 0;
741     print_word(mm, "\"");
742     mm->need_nl = 1;
743 }

743 /* See mdoc_term.c, synopsis_pre() for comments. */
744 static void
745 pre_syn(const struct mdoc_node *n)
746 {
747     if (NULL == n->prev || ! (MDOC_SYNPRETTY & n->flags))
748         return;
749     if (n->prev->tok == n->tok &&
750         MDOC_Ft != n->tok &&
751         MDOC_Fo != n->tok &&
752         MDOC_Fn != n->tok) {
753         outflags |= MMAN_br;
754         return;
755     }
756     switch (n->prev->tok) {
757     case (MDOC_Fd):
758         /* FALLTHROUGH */
759     case (MDOC_Fn):
760         /* FALLTHROUGH */
761     case (MDOC_Fo):
762         /* FALLTHROUGH */
763     case (MDOC_In):
764         /* FALLTHROUGH */
765     case (MDOC_Vt):
766         outflags |= MMAN_sp;
767         break;
768     case (MDOC_Ft):
769         if (MDOC_Fn != n->tok && MDOC_Fo != n->tok) {
770             outflags |= MMAN_sp;
771             break;
772         }
773         /* FALLTHROUGH */
774     default:
775         outflags |= MMAN_br;
776         break;
777     }
778 }

783 static int
784 pre_an(DECL_ARGS)
785 {
786     switch (n->norm->An.auth) {
787     case (AUTH_split):
788         outflags &= ~MMAN_An_nosplit;
789         outflags |= MMAN_An_split;
790         return(0);
791     case (AUTH_nosplit):
792         outflags &= ~MMAN_An_split;
793     }

```

```

794         outflags |= MMAN_An_nosplit;
795         return(0);
796     default:
797         if (MMAN_An_split & outflags)
798             outflags |= MMAN_br;
799         else if (SEC_AUTHORS == n->sec &&
800                ! (MMAN_An_nosplit & outflags))
801             outflags |= MMAN_An_split;
802         return(1);
803     }
804 }

806 static int
807 pre_ap(DECL_ARGS)
808 {

810     outflags &= ~MMAN_spc;
811     print_word("");
812     outflags &= ~MMAN_spc;
813     mm->need_space = 0;
814     print_word(mm, "");
815     mm->need_space = 0;
816     return(0);
817 }

816 static int
817 pre_bd(DECL_ARGS)
818 {

820     outflags &= ~(MMAN_PP | MMAN_sp | MMAN_br);

822     if (DISP_unfilled == n->norm->Bd.type ||
823         DISP_literal == n->norm->Bd.type)
824         print_line(".nf", 0);
825     if (0 == n->norm->Bd.comp && NULL != n->parent->prev)
826         outflags |= MMAN_sp;
827     print_offs(n->norm->Bd.off);
828     DISP_literal == n->norm->Bd.type) {
829         mm->need_nl = 1;
830         print_word(mm, ".nf");
831     }
832     mm->need_nl = 1;
833     return(1);
834 }

831 static void
832 post_bd(DECL_ARGS)
833 {

835     /* Close out this display. */
836     print_line(".RE", MMAN_nl);
837     if (DISP_unfilled == n->norm->Bd.type ||
838         DISP_literal == n->norm->Bd.type)
839         print_line(".fi", MMAN_nl);

841     /* Maybe we are inside an enclosing list? */
842     if (NULL != n->parent->next)
843         mid_it();
844 }

846 static int
847 pre_bf(DECL_ARGS)
848 {

850     switch (n->type) {
851     case (MDOC_BLOCK):

```

```

852         return(1);
853     case (MDOC_BODY):
854         break;
855     default:
856         return(0);
857     DISP_literal == n->norm->Bd.type) {
858         mm->need_nl = 1;
859         print_word(mm, ".fi");
860     }
861     switch (n->norm->Bf.font) {
862     case (FONT_Em):
863         font_push('I');
864         break;
865     case (FONT_Sy):
866         font_push('B');
867         break;
868     default:
869         font_push('R');
870         break;
871     }
872     return(1);
873     mm->need_nl = 1;
874 }

872 static void
873 post_bf(DECL_ARGS)
874 {

876     if (MDOC_BODY == n->type)
877         font_pop();
878 }

880 static int
881 pre_bk(DECL_ARGS)
882 {

884     switch (n->type) {
885     case (MDOC_BLOCK):
886         return(1);
887     case (MDOC_BODY):
888         outflags |= MMAN_Bk;
889         return(1);
890     default:
891         return(0);
892     }
893 }

895 static void
896 post_bk(DECL_ARGS)
897 {

899     if (MDOC_BODY == n->type)
900         outflags &= ~MMAN_Bk;
901 }

903 static int
904 pre_bl(DECL_ARGS)
905 {
906     size_t        icol;

908     /*
909     * print_offs() will increase the -offset to account for
910     * a possible enclosing .It, but any enclosed .It blocks
911     * just nest and do not add up their indentation.
912     */
913     if (n->norm->Bl.off) {

```

```

914         print_offs(n->norm->Bl.off);
915         Bl_stack[Bl_stack_len++] = 0;
916     }

918     switch (n->norm->Bl.type) {
919     case (LIST_enum):
920         n->norm->Bl.count = 0;
921         return(1);
922     case (LIST_column):
923         break;
924     default:
925         return(1);
926     }

928     print_line(".TS", MMAN_nl);
929     for (icol = 0; icol < n->norm->Bl.ncols; icol++)
930         print_word("l");
931     print_word(".");
932     outflags |= MMAN_nl;
933     return(1);
934 }

936 static void
937 post_bl(DECL_ARGS)
938 {

940     switch (n->norm->Bl.type) {
941     case (LIST_column):
942         print_line(".TE", 0);
943         break;
944     case (LIST_enum):
945         n->norm->Bl.count = 0;
946         break;
947     default:
948         break;
949     }

951     if (n->norm->Bl.off) {
952         print_line(".RE", MMAN_nl);
953         assert(Bl_stack_len);
954         Bl_stack_len--;
955         assert(0 == Bl_stack[Bl_stack_len]);
956     } else {
957         outflags |= MMAN_PP | MMAN_nl;
958         outflags &= ~(MMAN_sp | MMAN_br);
959     }

961     /* Maybe we are inside an enclosing list? */
962     if (NULL != n->parent->next)
963         mid_it();

965 }

967 static int
968 pre_br(DECL_ARGS)
969 {

971     outflags |= MMAN_br;
972     mm->need_nl = 1;
973     print_word(mm, ".br");
974     mm->need_nl = 1;
975     return(0);
976 }

977 static int
978 pre_bx(DECL_ARGS)

```

```

977 {

979     n = n->child;
980     if (n) {
981         print_word(n->string);
982         outflags &= ~MMAN_sp;
983         print_word(mm, n->string);
984         mm->need_space = 0;
985         n = n->next;
986     }
987     print_word("BSD");
988     print_word(mm, "BSD");
989     if (NULL == n)
990         return(0);
991     outflags &= ~MMAN_sp;
992     print_word("-");
993     outflags &= ~MMAN_sp;
994     print_word(n->string);
995     mm->need_space = 0;
996     print_word(mm, "-");
997     mm->need_space = 0;
998     print_word(mm, n->string);
999     return(0);
1000 }

1001 static int
1002 pre_dl(DECL_ARGS)
1003 {

1005     print_offs("6n");
1006     mm->need_nl = 1;
1007     print_word(mm, ".RS 6n");
1008     mm->need_nl = 1;
1009     return(1);
1010 }

1011 static void
1012 post_dl(DECL_ARGS)
1013 {

1015     print_line(".RE", MMAN_nl);

1017     /* Maybe we are inside an enclosing list? */
1018     if (NULL != n->parent->next)
1019         mid_it();
1020     mm->need_nl = 1;
1021     print_word(mm, ".RE");
1022     mm->need_nl = 1;
1023 }

1024 static int
1025 pre_em(DECL_ARGS)
1026 {

1028     font_push('I');
1029     return(1);
1030 }

1031 static void
1032 post_eo(DECL_ARGS)
1033 {

1035     if (MDOC_HEAD == n->type || MDOC_BODY == n->type)
1036         outflags &= ~MMAN_sp;
1037 }

```



```

1030 static int
1031 pre_fa(DECL_ARGS)
1032 {
1033     int    am_Fa;

1034     am_Fa = MDOC_Fa == n->tok;

1035     if (am_Fa)
1036         n = n->child;

1037     while (NULL != n) {
1038         font_push('I');
1039         if (am_Fa || MDOC_SYNPRETTY & n->flags)
1040             outflags |= MMAN_nbrword;
1041         print_node(meta, n);
1042         font_pop();
1043         if (NULL != (n = n->next))
1044             print_word(",");
1045     }
1046     return(0);
1047 }

1048 static void
1049 post_fa(DECL_ARGS)
1050 {
1051     if (NULL != n->next && MDOC_Fa == n->next->tok)
1052         print_word(",");
1053 }

1054 static int
1055 pre_fd(DECL_ARGS)
1056 {
1057     pre_syn(n);
1058     font_push('B');
1059     return(1);
1060 }

1061 static void
1062 post_fd(DECL_ARGS)
1063 {
1064     font_pop();
1065     outflags |= MMAN_br;
1066 }

1067 static int
1068 pre_fl(DECL_ARGS)
1069 {
1070     font_push('B');
1071     print_word("\\-");
1072     outflags &= ~MMAN_spc;
1073     return(1);
1074 }

1075 static void
1076 post_fl(DECL_ARGS)
1077 {
1078     font_pop();
1079     if (0 == n->nchild && NULL != n->next &&
1080         n->next->line == n->line)
1081         outflags &= ~MMAN_spc;
1082 }

```

```

1097 static int
1098 pre_fn(DECL_ARGS)
1099 {
1100     pre_syn(n);

1101     n = n->child;
1102     if (NULL == n)
1103         return(0);

1104     if (MDOC_SYNPRETTY & n->flags)
1105         print_block(".HP 4n", MMAN_nl);

1106     font_push('B');
1107     print_node(meta, n);
1108     font_pop();
1109     outflags &= ~MMAN_spc;
1110     print_word("(");
1111     outflags &= ~MMAN_spc;

1112     n = n->next;
1113     if (NULL != n)
1114         pre_fa(meta, n);
1115     return(0);
1116 }

1117 static void
1118 post_fn(DECL_ARGS)
1119 {
1120     print_word(")");
1121     if (MDOC_SYNPRETTY & n->flags) {
1122         print_word(";");
1123         outflags |= MMAN_PP;
1124     }
1125 }

1126 static int
1127 pre_fo(DECL_ARGS)
1128 {
1129     switch (n->type) {
1130     case (MDOC_BLOCK):
1131         pre_syn(n);
1132         break;
1133     case (MDOC_HEAD):
1134         if (MDOC_SYNPRETTY & n->flags)
1135             print_block(".HP 4n", MMAN_nl);
1136         font_push('B');
1137         break;
1138     case (MDOC_BODY):
1139         outflags &= ~MMAN_spc;
1140         print_word("(");
1141         outflags &= ~MMAN_spc;
1142         break;
1143     default:
1144         break;
1145     }
1146     return(1);
1147 }

1148 static void
1149 post_fo(DECL_ARGS)
1150 {
1151     print_word(")");
1152     if (MDOC_SYNPRETTY & n->flags)
1153         print_block(".HP 4n", MMAN_nl);
1154     font_pop();
1155     outflags &= ~MMAN_spc;
1156 }

1157 static void
1158 post_fl(DECL_ARGS)
1159 {
1160     font_pop();

```

```

1162     switch (n->type) {
1163     case (MDOC_HEAD):
1164         font_pop();
1165         break;
1166     case (MDOC_BODY):
1167         post_fn(meta, n);
1168         break;
1169     default:
1170         break;
1171     }
1172 }

1174 static int
1175 pre_ft(DECL_ARGS)
1176 {

1178     pre_syn(n);
1179     font_push('I');
1180     return(1);
1181 }

1183 static int
1184 pre_in(DECL_ARGS)
1185 {

1187     if (MDOC_SYNPretty & n->flags) {
1188         pre_syn(n);
1189         font_push('B');
1190         print_word("#include <");
1191         outflags &= ~MMAN_spc;
1192     } else {
1193         print_word("<");
1194         outflags &= ~MMAN_spc;
1195         font_push('I');
1196     }
1197     return(1);
1198 }

1200 static void
1201 post_in(DECL_ARGS)
1202 {

1204     if (MDOC_SYNPretty & n->flags) {
1205         outflags &= ~MMAN_spc;
1206         print_word(">");
1207         font_pop();
1208         outflags |= MMAN_br;
1209     } else {
1210         font_pop();
1211         outflags &= ~MMAN_spc;
1212         print_word(">");
1213     }
1214 }

1216 static int
1217 pre_it(DECL_ARGS)
1218 {
1219     const struct mdoc_node *bln;

1221     switch (n->type) {
1222     case (MDOC_HEAD):
1223         outflags |= MMAN_PP | MMAN_nl;
1224         bln = n->parent->parent;
1225         if (0 == bln->norm->Bl.comp ||
1226             (NULL == n->parent->prev &&
1227              NULL == bln->parent->prev))

```

```

1228         outflags |= MMAN_spc;
1229         outflags &= ~MMAN_br;
1230         if (MDOC_HEAD == n->type) {
1231             mm->need_nl = 1;
1232             print_word(mm, ".TP");
1233             bln = n->parent->parent->prev;
1234             switch (bln->norm->Bl.type) {
1235             case (LIST_item):
1236                 return(0);
1237             case (LIST_inset):
1238                 /* FALLTHROUGH */
1239             case (LIST_diag):
1240                 /* FALLTHROUGH */
1241             case (LIST_ohang):
1242                 if (bln->norm->Bl.type == LIST_diag)
1243                     print_line(".B \"", 0);
1244                 else
1245                     print_line(".R \"", 0);
1246                 outflags &= ~MMAN_spc;
1247                 return(1);
1248             case (LIST_bullet):
1249                 /* FALLTHROUGH */
1250             case (LIST_dash):
1251                 /* FALLTHROUGH */
1252             case (LIST_hyphen):
1253                 print_width(bln->norm->Bl.width, NULL, 0);
1254                 TPremain = 0;
1255                 outflags |= MMAN_nl;
1256                 font_push('B');
1257                 if (LIST_bullet == bln->norm->Bl.type)
1258                     print_word("o");
1259                 else
1260                     print_word("-");
1261                 font_pop();
1262                 print_word(mm, "4n");
1263                 mm->need_nl = 1;
1264                 print_word(mm, "\\fBo\\fP");
1265                 break;
1266             case (LIST_enum):
1267                 print_width(bln->norm->Bl.width, NULL, 0);
1268                 TPremain = 0;
1269                 outflags |= MMAN_nl;
1270                 print_count(&bln->norm->Bl.count);
1271                 break;
1272             case (LIST_hang):
1273                 print_width(bln->norm->Bl.width, n->child, 6);
1274                 TPremain = 0;
1275                 break;
1276             case (LIST_tag):
1277                 print_width(bln->norm->Bl.width, n->child, 0);
1278                 putchar('n');
1279                 outflags &= ~MMAN_spc;
1280                 return(1);
1281             default:
1282                 return(1);
1283             }
1284         }
1285         outflags |= MMAN_nl;
1286     default:
1287         if (bln->norm->Bl.width)
1288             print_word(mm, bln->norm->Bl.width);
1289     }
1290     break;
1291 }
1292 return(1);
1293 }
1294 /*

```

```

1285 * This function is called after closing out an indented block.
1286 * If we are inside an enclosing list, restore its indentation.
1287 */
1288 static void
1289 mid_it(void)
1290 {
1291     char        buf[24];

1293     /* Nothing to do outside a list. */
1294     if (0 == Bl_stack_len || 0 == Bl_stack[Bl_stack_len - 1])
1295         return;

1297     /* The indentation has already been set up. */
1298     if (Bl_stack_post[Bl_stack_len - 1])
1299         return;

1301     /* Restore the indentation of the enclosing list. */
1302     print_line(".RS", MMAN_Bk_susp);
1303     snprintf(buf, sizeof(buf), "%zun", Bl_stack[Bl_stack_len - 1]);
1304     print_word(buf);

1306     /* Remember to close out this .RS block later. */
1307     Bl_stack_post[Bl_stack_len - 1] = 1;
1308 }

1310 static void
1311 post_it(DECL_ARGS)
1312 {
1313     const struct mdoc_node *bln;

1315     bln = n->parent->parent;

1317     switch (n->type) {
1318     case (MDOC_HEAD):
1319         switch (bln->norm->Bl.type) {
1320         case (LIST_diag):
1321             outflags &= ~MMAN_spc;
1322             print_word("\\ ");
1323             break;
1324         case (LIST_ohang):
1325             outflags |= MMAN_br;
1326             break;
1327         default:
1328             break;
1329         }
1330         mm->need_nl = 1;
1331     case (MDOC_BODY):
1332         switch (bln->norm->Bl.type) {
1333         case (LIST_bullet):
1334             /* FALLTHROUGH */
1335         case (LIST_dash):
1336             /* FALLTHROUGH */
1337         case (LIST_hyphen):
1338             /* FALLTHROUGH */
1339         case (LIST_enum):
1340             /* FALLTHROUGH */
1341         case (LIST_hang):
1342             /* FALLTHROUGH */
1343         case (LIST_tag):
1344             assert(Bl_stack_len);
1345             Bl_stack[--Bl_stack_len] = 0;

1347             /*
1348              * Our indentation had to be restored
1349              * after a child display or child list.

```

```

1350         * Close out that indentation block now.
1351         */
1352         if (Bl_stack_post[Bl_stack_len]) {
1353             print_line(".RE", MMAN_nl);
1354             Bl_stack_post[Bl_stack_len] = 0;
1355         }
1356         break;
1357     case (LIST_column):
1358         if (NULL != n->next) {
1359             putchar('\t');
1360             outflags &= ~MMAN_spc;
1361         }
1362         break;
1363     default:
1364         break;
1365     }
1366     break;
1367 default:
1368     break;
1369 }
1370 }

1372 static void
1373 post_lb(DECL_ARGS)
1374 {

1376     if (SEC_LIBRARY == n->sec)
1377         outflags |= MMAN_br;
1378 }

1380 static int
1381 pre_lk(DECL_ARGS)
1382 {
1383     const struct mdoc_node *link, *descr;

1385     if (NULL == (link = n->child))
1386         return(0);

1388     if (NULL != (descr = link->next)) {
1389         font_push('I');
1390         while (NULL != descr) {
1391             print_word(descr->string);
1392             descr = descr->next;
1393         }
1394         print_word(":");
1395         font_pop();
1396     }

1398     font_push('B');
1399     print_word(link->string);
1400     font_pop();
1401     return(0);
1402 }

1404 static int
1405 pre_li(DECL_ARGS)
1406 {

1408     font_push('R');
1409     return(1);
1410 }

1412 static int
1413 pre_nm(DECL_ARGS)
1414 {
1415     char        *name;

```

```

1417     if (MDOC_BLOCK == n->type) {
1418         outflags |= MMAN_Bk;
1419         pre_syn(n);
1420     }
1421     if (MDOC_ELEM != n->type && MDOC_HEAD != n->type)
1422         return(1);
1423     name = n->child ? n->child->string : meta->name;
1424     if (NULL == name)
1425         return(0);
1426     if (MDOC_HEAD == n->type) {
1427         if (NULL == n->parent->prev)
1428             outflags |= MMAN_sp;
1429         print_block(".HP", 0);
1430         printf(" %zun", strlen(name) + 1);
1431         outflags |= MMAN_nl;
1432     }
1433     font_push('B');
1434     print_word(mm, "\\fB");
1435     mm->need_space = 0;
1436     if (NULL == n->child)
1437         print_word(meta->name);
1438     print_word(mm, m->name);
1439     return(1);
1440 }
1441
1442 static void
1443 post_nm(DECL_ARGS)
1444 {
1445     switch (n->type) {
1446     case (MDOC_BLOCK):
1447         outflags &= ~MMAN_Bk;
1448         break;
1449     case (MDOC_HEAD):
1450         /* FALLTHROUGH */
1451     case (MDOC_ELEM):
1452         font_pop();
1453         break;
1454     default:
1455         break;
1456     }
1457     if (MDOC_ELEM != n->type && MDOC_HEAD != n->type)
1458         return;
1459     mm->need_space = 0;
1460     print_word(mm, "\\fP");
1461 }
1462
1463 static int
1464 pre_no(DECL_ARGS)
1465 {
1466     outflags |= MMAN_spc_force;
1467     return(1);
1468 }
1469
1470 static int
1471 pre_ns(DECL_ARGS)
1472 {
1473     outflags &= ~MMAN_spc;
1474     mm->need_space = 0;
1475     return(0);
1476 }
1477
1478 static void

```

```

1474 post_pf(DECL_ARGS)
1475 {
1476     outflags &= ~MMAN_spc;
1477     mm->need_space = 0;
1478 }
1479
1480 static int
1481 pre_pp(DECL_ARGS)
1482 {
1483     if (MDOC_It != n->parent->tok)
1484         outflags |= MMAN_PP;
1485     outflags |= MMAN_sp | MMAN_nl;
1486     outflags &= ~MMAN_br;
1487     return(0);
1488 }
1489
1490 static int
1491 pre_rs(DECL_ARGS)
1492 {
1493     if (SEC_SEE_ALSO == n->sec) {
1494         outflags |= MMAN_PP | MMAN_sp | MMAN_nl;
1495         outflags &= ~MMAN_br;
1496     }
1497     mm->need_nl = 1;
1498     if (MDOC_It == n->parent->tok)
1499         print_word(mm, ".sp");
1500     else
1501         print_word(mm, ".PP");
1502     mm->need_nl = 1;
1503     return(1);
1504 }
1505
1506 static int
1507 pre_sm(DECL_ARGS)
1508 {
1509     assert(n->child && MDOC_TEXT == n->child->type);
1510     if (0 == strcmp("on", n->child->string))
1511         outflags |= MMAN_Sm | MMAN_spc;
1512     else
1513         outflags &= ~MMAN_Sm;
1514     return(0);
1515 }
1516
1517 static int
1518 pre_sp(DECL_ARGS)
1519 {
1520     if (MMAN_PP & outflags) {
1521         outflags &= ~MMAN_PP;
1522         print_line(".PP", 0);
1523     } else
1524         print_line(".sp", 0);
1525     mm->need_nl = 1;
1526     print_word(mm, ".sp");
1527     return(1);
1528 }
1529
1530 static void
1531 post_sp(DECL_ARGS)
1532 {
1533     outflags |= MMAN_nl;

```

```

605     mm->need_nl = 1;
1531 }

1533 static int
1534 pre_sy(DECL_ARGS)
1535 {
1537     font_push('B');
1538     return(1);
1539 }

1541 static int
1542 pre_vt(DECL_ARGS)
1543 {
1545     if (MDOC_SYNPRETTY & n->flags) {
1546         switch (n->type) {
1547             case (MDOC_BLOCK):
1548                 pre_syn(n);
1549                 return(1);
1550             case (MDOC_BODY):
1551                 break;
1552             default:
1553                 return(0);
1554         }
1555     }
1556     font_push('I');
1557     return(1);
1558 }

1560 static void
1561 post_vt(DECL_ARGS)
1562 {
1564     if (MDOC_SYNPRETTY & n->flags && MDOC_BODY != n->type)
1565         return;
1566     font_pop();
1567 }

1569 static int
1570 pre_xr(DECL_ARGS)
1571 {
1573     n = n->child;
1574     if (NULL == n)
1575         return(0);
1576     print_node(meta, n);
1577     print_node(m, n, mm);
1578     n = n->next;
1579     if (NULL == n)
1580         return(0);
1581     outflags &= ~MMAN_spc;
1582     print_word("(");
1583     print_node(meta, n);
1584     print_word(")");
1585     mm->need_space = 0;
1586     print_word(mm, "(");
1587     print_node(m, n, mm);
1588     print_word(mm, ")");
1589     return(0);
1590 }

1587 static int
1588 pre_ux(DECL_ARGS)
1589 {

```

```

1591     print_word(manacts[n->tok].prefix);
1592     print_word(mm, manacts[n->tok].prefix);
1593     if (NULL == n->child)
1594         return(0);
1595     outflags &= ~MMAN_spc;
1596     print_word("\\ ");
1597     outflags &= ~MMAN_spc;
1598     mm->need_space = 0;
1599     print_word(mm, "\\~");
1600     mm->need_space = 0;
1601     return(1);
1602 }

```

unchanged\_portion\_omitted

```

*****
45416 Fri Jul 25 15:00:14 2014
new/usr/src/cmd/mandoc/mdoc_term.c
Update to 1.12.3.
*****
1 /* $Id: mdoc_term.c,v 1.258 2013/12/25 21:24:12 schwarze Exp $ */
1 /* $Id: mdoc_term.c,v 1.238 2011/11/13 13:15:14 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
5  * Copyright (c) 2013 Franco Fichtner <franco@lastsummer.de>
4  * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
6  *
7  * Permission to use, copy, modify, and distribute this software for any
8  * purpose with or without fee is hereby granted, provided that the above
9  * copyright notice and this permission notice appear in all copies.
10 *
11 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
12 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
13 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
14 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
15 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
16 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
17 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
18 */
19 #ifdef HAVE_CONFIG_H
20 #include "config.h"
21 #endif

23 #include <sys/types.h>

25 #include <assert.h>
26 #include <ctype.h>
27 #include <stdint.h>
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <string.h>

32 #include "mandoc.h"
33 #include "out.h"
34 #include "term.h"
35 #include "mdoc.h"
36 #include "main.h"

38 struct termpair {
39     struct termpair *ppair;
40     int count;
41 };

43 #define DECL_ARGS struct termp *p, \
44                 struct termpair *pair, \
45                 const struct mdoc_meta *meta, \
46                 struct mdoc_node *n
44                 const struct mdoc_meta *m, \
45                 const struct mdoc_node *n

48 struct termact {
49     int (*pre)(DECL_ARGS);
50     void (*post)(DECL_ARGS);
51 };

53 static size_t a2width(const struct termp *, const char *);
54 static size_t a2height(const struct termp *, const char *);
55 static size_t a2offs(const struct termp *, const char *);

57 static void print_bvspace(struct termp *,

```

```

58     const struct mdoc_node *,
59     const struct mdoc_node *);
60 static void print_mdoc_node(DECL_ARGS);
61 static void print_mdoc_nodelist(DECL_ARGS);
62 static void print_mdoc_head(struct termp *, const void *);
63 static void print_mdoc_foot(struct termp *, const void *);
64 static void synopsis_pre(struct termp *,
65     const struct mdoc_node *);

67 static void term__post(DECL_ARGS);
68 static void term__t_post(DECL_ARGS);
69 static void term_an_post(DECL_ARGS);
70 static void term_bd_post(DECL_ARGS);
71 static void term_bk_post(DECL_ARGS);
72 static void term_bl_post(DECL_ARGS);
73 static void term_fd_post(DECL_ARGS);
72 static void term_d1_post(DECL_ARGS);
74 static void term_fo_post(DECL_ARGS);
75 static void term_in_post(DECL_ARGS);
76 static void term_it_post(DECL_ARGS);
77 static void term_lb_post(DECL_ARGS);
78 static void term_nm_post(DECL_ARGS);
79 static void term_pf_post(DECL_ARGS);
80 static void term_quote_post(DECL_ARGS);
81 static void term_sh_post(DECL_ARGS);
82 static void term_ss_post(DECL_ARGS);

84 static int term__a_pre(DECL_ARGS);
85 static int term__t_pre(DECL_ARGS);
86 static int term_an_pre(DECL_ARGS);
87 static int term_ap_pre(DECL_ARGS);
88 static int term_bd_pre(DECL_ARGS);
89 static int term_bf_pre(DECL_ARGS);
90 static int term_bk_pre(DECL_ARGS);
91 static int term_bl_pre(DECL_ARGS);
92 static int term_bold_pre(DECL_ARGS);
93 static int term_bt_pre(DECL_ARGS);
94 static int term_bx_pre(DECL_ARGS);
95 static int term_cd_pre(DECL_ARGS);
96 static int term_dl_pre(DECL_ARGS);
97 static int term_ex_pre(DECL_ARGS);
98 static int term_fa_pre(DECL_ARGS);
99 static int term_fd_pre(DECL_ARGS);
100 static int term_fl_pre(DECL_ARGS);
101 static int term_fn_pre(DECL_ARGS);
102 static int term_fo_pre(DECL_ARGS);
103 static int term_ft_pre(DECL_ARGS);
103 static int term_igndelim_pre(DECL_ARGS);
104 static int term_in_pre(DECL_ARGS);
105 static int term_it_pre(DECL_ARGS);
106 static int term_li_pre(DECL_ARGS);
107 static int term_lk_pre(DECL_ARGS);
108 static int term_nd_pre(DECL_ARGS);
109 static int term_nm_pre(DECL_ARGS);
110 static int term_ns_pre(DECL_ARGS);
111 static int term_quote_pre(DECL_ARGS);
112 static int term_rs_pre(DECL_ARGS);
113 static int term_rv_pre(DECL_ARGS);
114 static int term_sh_pre(DECL_ARGS);
115 static int term_sm_pre(DECL_ARGS);
116 static int term_sp_pre(DECL_ARGS);
117 static int term_ss_pre(DECL_ARGS);
118 static int term_under_pre(DECL_ARGS);
119 static int term_ud_pre(DECL_ARGS);
120 static int term_vt_pre(DECL_ARGS);
121 static int term_xr_pre(DECL_ARGS);

```

```

122 static int      term_p_xx_pre(DECL_ARGS);

124 static const struct termact termacts[MDOC_MAX] = {
125     term_p_ap_pre, NULL }, /* Ap */
126     NULL, NULL }, /* Dd */
127     NULL, NULL }, /* Dt */
128     NULL, NULL }, /* Os */
129     term_p_sh_pre, term_p_sh_post }, /* Sh */
130     term_p_ss_pre, term_p_ss_post }, /* Ss */
131     term_p_sp_pre, NULL }, /* Pp */
132     term_p_d1_pre, term_p_b1_post }, /* D1 */
133     term_p_d1_pre, term_p_b1_post }, /* D1 */
132     term_p_d1_pre, term_p_d1_post }, /* D1 */
133     term_p_d1_pre, term_p_d1_post }, /* D1 */
134     term_p_bd_pre, term_p_bd_post }, /* Bd */
135     NULL, NULL }, /* Ed */
136     term_p_b1_pre, term_p_b1_post }, /* B1 */
137     NULL, NULL }, /* E1 */
138     term_p_it_pre, term_p_it_post }, /* It */
139     term_p_under_pre, NULL }, /* Ad */
140     term_p_an_pre, term_p_an_post }, /* An */
141     term_p_under_pre, NULL }, /* Ar */
142     term_p_cd_pre, NULL }, /* Cd */
143     term_p_bold_pre, NULL }, /* Cm */
144     NULL, NULL }, /* Dv */
145     NULL, NULL }, /* Er */
146     NULL, NULL }, /* Ev */
147     term_p_ex_pre, NULL }, /* Ex */
148     term_p_fa_pre, NULL }, /* Fa */
149     term_p_fd_pre, term_p_fd_post }, /* Fd */
149     term_p_fd_pre, NULL }, /* Fd */
150     term_p_fl_pre, NULL }, /* Fl */
151     term_p_fn_pre, NULL }, /* Fn */
152     term_p_ft_pre, NULL }, /* Ft */
153     term_p_bold_pre, NULL }, /* Ic */
154     term_p_in_pre, term_p_in_post }, /* In */
155     term_p_li_pre, NULL }, /* Li */
156     term_p_nd_pre, NULL }, /* Nd */
157     term_p_nm_pre, term_p_nm_post }, /* Nm */
158     term_p_quote_pre, term_p_quote_post }, /* Op */
159     NULL, NULL }, /* Ot */
160     term_p_under_pre, NULL }, /* Pa */
161     term_p_rv_pre, NULL }, /* Rv */
162     NULL, NULL }, /* St */
163     term_p_under_pre, NULL }, /* Va */
164     term_p_vt_pre, NULL }, /* Vt */
165     term_p_xr_pre, NULL }, /* Xr */
166     term_p_a_pre, term_p_post }, /* %A */
167     term_p_under_pre, term_p_post }, /* %B */
168     NULL, term_p_post }, /* %D */
169     term_p_under_pre, term_p_post }, /* %I */
170     term_p_under_pre, term_p_post }, /* %J */
171     NULL, term_p_post }, /* %N */
172     NULL, term_p_post }, /* %O */
173     NULL, term_p_post }, /* %P */
174     NULL, term_p_post }, /* %R */
175     term_p_t_pre, term_p_t_post }, /* %T */
176     NULL, term_p_post }, /* %V */
177     NULL, NULL }, /* Ac */
178     term_p_quote_pre, term_p_quote_post }, /* Ao */
179     term_p_quote_pre, term_p_quote_post }, /* Aq */
180     NULL, NULL }, /* At */
181     NULL, NULL }, /* Bc */
182     term_p_bf_pre, NULL }, /* Bf */
183     term_p_quote_pre, term_p_quote_post }, /* Bo */
184     term_p_quote_pre, term_p_quote_post }, /* Bq */

```

```

185     term_p_xx_pre, NULL }, /* Bsx */
186     term_p_bx_pre, NULL }, /* Bx */
187     NULL, NULL }, /* Db */
188     NULL, NULL }, /* Dc */
189     term_p_quote_pre, term_p_quote_post }, /* Do */
190     term_p_quote_pre, term_p_quote_post }, /* Dq */
191     NULL, NULL }, /* Ec */ /* FIXME: no space */
192     NULL, NULL }, /* Ef */
193     term_p_under_pre, NULL }, /* Em */
194     term_p_quote_pre, term_p_quote_post }, /* Eo */
195     term_p_xx_pre, NULL }, /* Fx */
196     term_p_bold_pre, NULL }, /* Ms */
197     NULL, NULL }, /* No */
197     term_p_igndelim_pre, NULL }, /* No */
198     term_p_ns_pre, NULL }, /* Ns */
199     term_p_xx_pre, NULL }, /* Nx */
200     term_p_xx_pre, NULL }, /* Ox */
201     NULL, NULL }, /* Pc */
202     NULL, term_p_pf_post }, /* Pf */
202     term_p_igndelim_pre, term_p_pf_post }, /* Pf */
203     term_p_quote_pre, term_p_quote_post }, /* Po */
204     term_p_quote_pre, term_p_quote_post }, /* Pq */
205     NULL, NULL }, /* Qc */
206     term_p_quote_pre, term_p_quote_post }, /* Ql */
207     term_p_quote_pre, term_p_quote_post }, /* Qo */
208     term_p_quote_pre, term_p_quote_post }, /* Qq */
209     NULL, NULL }, /* Re */
210     term_p_rs_pre, NULL }, /* Rs */
211     NULL, NULL }, /* Sc */
212     term_p_quote_pre, term_p_quote_post }, /* So */
213     term_p_quote_pre, term_p_quote_post }, /* Sq */
214     term_p_sm_pre, NULL }, /* Sm */
215     term_p_under_pre, NULL }, /* Sx */
216     term_p_bold_pre, NULL }, /* Sy */
217     NULL, NULL }, /* Tn */
218     term_p_xx_pre, NULL }, /* Ux */
219     NULL, NULL }, /* Xc */
220     NULL, NULL }, /* Xo */
221     term_p_fo_pre, term_p_fo_post }, /* Fo */
222     NULL, NULL }, /* Fc */
223     term_p_quote_pre, term_p_quote_post }, /* Oo */
224     NULL, NULL }, /* Oc */
225     term_p_bk_pre, term_p_bk_post }, /* Bk */
226     NULL, NULL }, /* Ek */
227     term_p_bt_pre, NULL }, /* Bt */
228     NULL, NULL }, /* Hf */
229     NULL, NULL }, /* Fr */
230     term_p_ud_pre, NULL }, /* Ud */
231     NULL, term_p_lb_post }, /* Lb */
232     term_p_sp_pre, NULL }, /* Lp */
233     term_p_lk_pre, NULL }, /* Lk */
234     term_p_under_pre, NULL }, /* Mt */
235     term_p_quote_pre, term_p_quote_post }, /* Brq */
236     term_p_quote_pre, term_p_quote_post }, /* Bro */
237     NULL, NULL }, /* Brc */
238     NULL, term_p_post }, /* %C */
239     NULL, NULL }, /* Es */ /* TODO */
240     NULL, NULL }, /* En */ /* TODO */
241     term_p_xx_pre, NULL }, /* Dx */
242     NULL, term_p_post }, /* %Q */
243     term_p_sp_pre, NULL }, /* br */
244     term_p_sp_pre, NULL }, /* sp */
245     NULL, term_p_post }, /* %U */
245     term_p_under_pre, term_p_post }, /* %U */
246     NULL, NULL }, /* Ta */
247 };

```

```

250 void
251 terminal_mdoc(void *arg, const struct mdoc *mdoc)
252 {
253     const struct mdoc_node *n;
254     const struct mdoc_meta *meta;
255     const struct mdoc_meta *m;
256     struct term *p;
257
258     p = (struct term *)arg;
259
260     if (0 == p->defindent)
261         p->defindent = 5;
262
263     p->overstep = 0;
264     p->maxrmargin = p->defrmargin;
265     p->tabwidth = term_len(p, 5);
266
267     if (NULL == p->syntab)
268         p->syntab = mchars_alloc();
269
270     n = mdoc_node(mdoc);
271     meta = mdoc_meta(mdoc);
272     m = mdoc_meta(mdoc);
273
274     term_begin(p, print_mdoc_head, print_mdoc_foot, meta);
275     term_begin(p, print_mdoc_head, print_mdoc_foot, m);
276
277     if (n->child)
278         print_mdoc_nodelist(p, NULL, meta, n->child);
279     print_mdoc_nodelist(p, NULL, m, n->child);
280
281     term_end(p);
282 }
283
284 static void
285 print_mdoc_nodelist(DECL_ARGS)
286 {
287     print_mdoc_node(p, pair, meta, n);
288     print_mdoc_node(p, pair, m, n);
289     if (n->next)
290         print_mdoc_nodelist(p, pair, meta, n->next);
291     print_mdoc_nodelist(p, pair, m, n->next);
292 }
293
294 /* ARGSUSED */
295 static void
296 print_mdoc_node(DECL_ARGS)
297 {
298     int chld;
299     const void *font;
300     struct termpair npair;
301     size_t offset, rmargin;
302
303     chld = 1;
304     offset = p->offset;
305     rmargin = p->rmargin;
306     n->prev_font = term_fontq(p);
307     font = term_fontq(p);
308
309     memset(&npair, 0, sizeof(struct termpair));
310     npair.ppair = pair;

```

```

307     /*
308     * Keeps only work until the end of a line. If a keep was
309     * invoked in a prior line, revert it to PREKEEP.
310     *
311     * Also let SYNPRETTY sections behave as if they were wrapped
312     * in a 'Ek' block.
313     */
314
315     if (TERMP_KEEP & p->flags) {
316         if (n->prev ? (n->prev->lastline != n->line) :
317             (n->parent && n->parent->line != n->line)) {
318             if (TERMP_KEEP & p->flags || MDOC_SYNPRETTY & n->flags) {
319                 if (n->prev && n->prev->line != n->line) {
320                     p->flags &= ~TERMP_KEEP;
321                     p->flags |= TERMP_PREKEEP;
322                 } else if (NULL == n->prev) {
323                     if (n->parent && n->parent->line != n->line) {
324                         p->flags &= ~TERMP_KEEP;
325                         p->flags |= TERMP_PREKEEP;
326                     }
327                 }
328             }
329         }
330
331         /*
332         * Since SYNPRETTY sections aren't "turned off" with 'Ek',
333         * we have to intuit whether we should disable formatting.
334         */
335
336         if (! (MDOC_SYNPRETTY & n->flags) &&
337             ((n->prev && MDOC_SYNPRETTY & n->prev->flags) ||
338              (n->parent && MDOC_SYNPRETTY & n->parent->flags)))
339             p->flags &= ~(TERMP_KEEP | TERMP_PREKEEP);
340
341     /*
342     * After the keep flags have been set up, we may now
343     * produce output. Note that some pre-handlers do so.
344     */
345
346     switch (n->type) {
347     case (MDOC_TEXT):
348         if (' ' == *n->string && MDOC_LINE & n->flags)
349             term_newln(p);
350         if (MDOC_DELIMC & n->flags)
351             p->flags |= TERMP_NOSPACE;
352         term_word(p, n->string);
353         if (MDOC_DELIMO & n->flags)
354             p->flags |= TERMP_NOSPACE;
355         break;
356     case (MDOC_EQN):
357         term_eqn(p, n->eqn);
358         break;
359     case (MDOC_TBL):
360         term_tbl(p, n->span);
361         break;
362     default:
363         if (termacts[n->tok].pre && ENDBODY_NOT == n->end)
364             chld = (*termacts[n->tok].pre)
365                 (p, &npair, meta, n);
366         break;
367     }
368
369     if (chld && n->child)
370         print_mdoc_nodelist(p, &npair, meta, n->child);
371     print_mdoc_nodelist(p, &npair, m, n->child);

```



```

351 term_fontpopq(p,
352 (ENDBODY_NOT == n->end ? n : n->pending)->prev_font);
359 term_fontpopq(p, font);

354 switch (n->type) {
355 case (MDOC_TEXT):
356     break;
357 case (MDOC_TBL):
358     break;
359 case (MDOC_EQN):
360     break;
361 default:
362     if (! termacts[n->tok].post || MDOC_ENDED & n->flags)
363         break;
364 (void)(*termacts[n->tok].post)(p, &npair, meta, n);
381 (void)(*termacts[n->tok].post)(p, &npair, m, n);

366 /*
367  * Explicit end tokens not only call the post
368  * handler, but also tell the respective block
369  * that it must not call the post handler again.
370  */
371 if (ENDBODY_NOT != n->end)
372     n->pending->flags |= MDOC_ENDED;

374 /*
375  * End of line terminating an implicit block
376  * while an explicit block is still open.
377  * Continue the explicit block without spacing.
378  */
379 if (ENDBODY_NOSPACE == n->end)
380     p->flags |= TERMP_NOSPACE;
381 break;
382 }

384 if (MDOC_EOS & n->flags)
385     p->flags |= TERMP_SENTENCE;

387 p->offset = offset;
388 p->rmargin = rmargin;
389 }

392 static void
393 print_mdofoot(struct term *p, const void *arg)
394 {
395     const struct mdoc_meta *meta;
412     const struct mdoc_meta *m;

397     meta = (const struct mdoc_meta *)arg;
414     m = (const struct mdoc_meta *)arg;

399     term_fontrepl(p, TERMPFONT_NONE);

401     /*
402     * Output the footer in new-groff style, that is, three columns
403     * with the middle being the manual date and flanking columns
404     * being the operating system:
405     *
406     * SYSTEM                DATE                SYSTEM
407     */

409     term_vspace(p);

411     p->offset = 0;

```

```

412     p->rmargin = (p->maxrmargin -
413                 term_strlen(p, meta->date) + term_len(p, 1)) / 2;
414     p->trailspace = 1;
430     term_strlen(p, m->date) + term_len(p, 1)) / 2;
415     p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;

417     term_word(p, meta->os);
433     term_word(p, m->os);
418     term_flushln(p);

420     p->offset = p->rmargin;
421     p->rmargin = p->maxrmargin - term_strlen(p, meta->os);
437     p->rmargin = p->maxrmargin - term_strlen(p, m->os);
422     p->flags |= TERMP_NOSPACE;

424     term_word(p, meta->date);
440     term_word(p, m->date);
425     term_flushln(p);

427     p->offset = p->rmargin;
428     p->rmargin = p->maxrmargin;
429     p->trailspace = 0;
430     p->flags &= ~TERMP_NOBREAK;
431     p->flags |= TERMP_NOSPACE;

433     term_word(p, meta->os);
448     term_word(p, m->os);
434     term_flushln(p);

436     p->offset = 0;
437     p->rmargin = p->maxrmargin;
438     p->flags = 0;
439 }

442 static void
443 print_mdofhead(struct term *p, const void *arg)
444 {
445     char          buf[BUFSIZ], title[BUFSIZ];
446     size_t        buflen, titlen;
447     const struct mdoc_meta *meta;
462     const struct mdoc_meta *m;

449     meta = (const struct mdoc_meta *)arg;
464     m = (const struct mdoc_meta *)arg;

451     /*
452     * The header is strange. It has three components, which are
453     * really two with the first duplicated. It goes like this:
454     *
455     * IDENTIFIER                TITLE                IDENTIFIER
456     *
457     * The IDENTIFIER is NAME(SECTION), which is the command-name
458     * (if given, or "unknown" if not) followed by the manual page
459     * section. These are given in 'Dt'. The TITLE is a free-form
460     * string depending on the manual volume. If not specified, it
461     * switches on the manual section.
462     */

464     p->offset = 0;
465     p->rmargin = p->maxrmargin;

467     assert(meta->vol);
468     strncpy(buf, meta->vol, BUFSIZ);
482     assert(m->vol);
483     strncpy(buf, m->vol, BUFSIZ);

```

```

469     buflen = term_strlen(p, buf);
471     if (meta->arch) {
486     if (m->arch) {
472         strlcat(buf, " (", BUFSIZ);
473         strlcat(buf, meta->arch, BUFSIZ);
488         strlcat(buf, m->arch, BUFSIZ);
474         strlcat(buf, ")", BUFSIZ);
475     }
477     snprintf(title, BUFSIZ, "%s(%s)", meta->title, meta->msec);
492     snprintf(title, BUFSIZ, "%s(%s)", m->title, m->msec);
478     titlen = term_strlen(p, title);
480     p->flags |= TERMP_NOBREAK | TERMP_NOSPACE;
481     p->trailspace = 1;
482     p->offset = 0;
483     p->rmargin = 2 * (titlen+1) + buflen < p->maxrmargin ?
484         (p->maxrmargin -
485          term_strlen(p, buf) + term_len(p, 1)) / 2 :
486         p->maxrmargin - buflen;
488     term_word(p, title);
489     term_flushln(p);
491     p->flags |= TERMP_NOSPACE;
492     p->offset = p->rmargin;
493     p->rmargin = p->offset + buflen + titlen < p->maxrmargin ?
494         p->maxrmargin - titlen : p->maxrmargin;
496     term_word(p, buf);
497     term_flushln(p);
499     p->flags &= ~TERMP_NOBREAK;
500     p->trailspace = 0;
501     if (p->rmargin + titlen <= p->maxrmargin) {
502         p->flags |= TERMP_NOSPACE;
503         p->offset = p->rmargin;
504         p->rmargin = p->maxrmargin;
505         term_word(p, title);
506         term_flushln(p);
507     }
509     p->flags &= ~TERMP_NOSPACE;
510     p->offset = 0;
511     p->rmargin = p->maxrmargin;
512 }
    unchanged_portion_omitted
616 /* ARGSUSED */
617 static int
618 term_p_it_pre(DECL_ARGS)
619 {
620     const struct mdoc_node *bl, *nn;
621     char buf[7];
622     int i;
623     size_t width, offset, ncols, dcol;
624     enum mdoc_list type;
626     if (MDOC_BLOCK == n->type) {
627         print_byspace(p, n->parent->parent, n);
628         return(1);
629     }
631     bl = n->parent->parent->parent;

```

```

632     type = bl->norm->Bl.type;
634     /*
635     * First calculate width and offset. This is pretty easy unless
636     * we're a -column list, in which case all prior columns must
637     * be accounted for.
638     */
640     width = offset = 0;
642     if (bl->norm->Bl.off)
643         offset = a2offs(p, bl->norm->Bl.off);
645     switch (type) {
646     case (LIST_column):
647         if (MDOC_HEAD == n->type)
648             break;
650         /*
651         * Imitate groff's column handling:
652         * - For each earlier column, add its width.
653         * - For less than 5 columns, add four more blanks per
654         *   column.
655         * - For exactly 5 columns, add three more blank per
656         *   column.
657         * - For more than 5 columns, add only one column.
658         */
659         ncols = bl->norm->Bl.ncols;
661         /* LINTED */
662         dcol = ncols < 5 ? term_len(p, 4) :
663             ncols == 5 ? term_len(p, 3) : term_len(p, 1);
665         /*
666         * Calculate the offset by applying all prior MDOC_BODY,
667         * so we stop at the MDOC_HEAD (NULL == nn->prev).
668         */
670         for (i = 0, nn = n->prev;
671              nn->prev && i < (int)ncols;
672              nn = nn->prev, i++)
673             offset += dcol + a2width
674                 (p, bl->norm->Bl.cols[i]);
676         /*
677         * When exceeding the declared number of columns, leave
678         * the remaining widths at 0. This will later be
679         * adjusted to the default width of 10, or, for the last
680         * column, stretched to the right margin.
681         */
682         if (i >= (int)ncols)
683             break;
685         /*
686         * Use the declared column widths, extended as explained
687         * in the preceding paragraph.
688         */
689         width = a2width(p, bl->norm->Bl.cols[i]) + dcol;
690         break;
691     default:
692         if (NULL == bl->norm->Bl.width)
693             break;
695         /*
696         * Note: buffer the width by 2, which is groff's magic
697         * number for buffering single arguments. See the above

```

```

698     * handling for column for how this changes.
699     */
700     assert(bl->norm->Bl.width);
701     width = a2width(p, bl->norm->Bl.width) + term_len(p, 2);
702     break;
703 }
704
705 /*
706  * List-type can override the width in the case of fixed-head
707  * values (bullet, dash/hyphen, enum).  Tags need a non-zero
708  * offset.
709  */
710
711 switch (type) {
712 case (LIST_bullet):
713     /* FALLTHROUGH */
714 case (LIST_dash):
715     /* FALLTHROUGH */
716 case (LIST_hyphen):
717     /* FALLTHROUGH */
718     if (width < term_len(p, 4))
719         width = term_len(p, 4);
720     break;
721 case (LIST_enum):
722     if (width < term_len(p, 2))
723         width = term_len(p, 2);
724     if (width < term_len(p, 5))
725         width = term_len(p, 5);
726     break;
727 case (LIST_hang):
728     if (0 == width)
729         width = term_len(p, 8);
730     break;
731 case (LIST_column):
732     /* FALLTHROUGH */
733 case (LIST_tag):
734     if (0 == width)
735         width = term_len(p, 10);
736     break;
737 default:
738     break;
739 }
740
741 p->flags |= TERMP_NOSPACE;
742
743 switch (type) {
744 case (LIST_diag):
745     if (MDOC_BODY == n->type)
746         term_word(p, "\\ \\ ");
747     break;
748 case (LIST_inset):
749     if (MDOC_BODY == n->type)
750         term_word(p, "\\ ");
751     break;
752 default:
753     break;
754 }
755
756 p->flags |= TERMP_NOSPACE;
757
758 switch (type) {

```

```

759     case (LIST_diag):
760         if (MDOC_HEAD == n->type)
761             term_fontpush(p, TERMFONT_BOLD);
762         break;
763 default:
764     break;
765 }
766
767 /*
768  * Pad and break control.  This is the tricky part.  These flags
769  * are documented in term_flushln() in term.c.  Note that we're
770  * going to unset all of these flags in term_post() when we
771  * exit.
772  */
773
774 switch (type) {
775 case (LIST_enum):
776     /*
777      * Weird special case.
778      * Very narrow enum lists actually hang.
779      */
780     if (width == term_len(p, 2))
781         p->flags |= TERMP_HANG;
782     /* FALLTHROUGH */
783 case (LIST_bullet):
784     /* FALLTHROUGH */
785 case (LIST_dash):
786     /* FALLTHROUGH */
787 case (LIST_enum):
788     /* FALLTHROUGH */
789 case (LIST_hyphen):
790     if (MDOC_HEAD != n->type)
791         break;
792     if (MDOC_HEAD == n->type)
793         p->flags |= TERMP_NOBREAK;
794     p->trailspace = 1;
795     break;
796 case (LIST_hang):
797     if (MDOC_HEAD != n->type)
798         if (MDOC_HEAD == n->type)
799             p->flags |= TERMP_NOBREAK;
800     else
801         break;
802
803     /*
804      * This is ugly.  If '-hang' is specified and the body
805      * is a 'Bl' or 'Bd', then we want basically to nullify
806      * the "overstep" effect in term_flushln() and treat
807      * this as a '-ohang' list instead.
808      */
809     if (n->next->child &&
810         (MDOC_Bl == n->next->child->tok ||
811          MDOC_Bd == n->next->child->tok))
812         p->flags &= ~TERMP_NOBREAK;
813     else
814         p->flags |= TERMP_HANG;
815     break;
816
817     p->flags |= TERMP_NOBREAK | TERMP_HANG;
818     p->trailspace = 1;
819     break;
820 case (LIST_tag):
821     if (MDOC_HEAD == n->type)
822         p->flags |= TERMP_NOBREAK | TERMP_TWOSPACE;
823
824     if (MDOC_HEAD != n->type)

```

```

813         break;
815         p->flags |= TERMP_NOBREAK;
816         p->trailspace = 2;
818         if (NULL == n->next || NULL == n->next->child)
819             p->flags |= TERMP_DANGLE;
820         break;
821     case (LIST_column):
822         if (MDOC_HEAD == n->type)
823             break;
825         if (NULL == n->next) {
832             if (NULL == n->next)
826                 p->flags &= ~TERMP_NOBREAK;
827                 p->trailspace = 0;
828             } else {
834                 else
829                 p->flags |= TERMP_NOBREAK;
830                 p->trailspace = 1;
831             }
833         break;
834     case (LIST_diag):
835         if (MDOC_HEAD != n->type)
836             break;
839         if (MDOC_HEAD == n->type)
837             p->flags |= TERMP_NOBREAK;
838             p->trailspace = 1;
839         break;
840     default:
841         break;
842     }
844     /*
845     * Margin control. Set-head-width lists have their right
846     * margins shortened. The body for these lists has the offset
847     * necessarily lengthened. Everybody gets the offset.
848     */
850     p->offset += offset;
852     switch (type) {
853     case (LIST_hang):
854         /*
855         * Same stipulation as above, regarding '-hang'. We
856         * don't want to recalculate rmargin and offsets when
857         * using 'Bd' or 'Bl' within '-hang' overstep lists.
858         */
859         if (MDOC_HEAD == n->type && n->next->child &&
860             (MDOC_Bl == n->next->child->tok ||
861              MDOC_Bd == n->next->child->tok))
862             break;
863         /* FALLTHROUGH */
864     case (LIST_bullet):
865         /* FALLTHROUGH */
866     case (LIST_dash):
867         /* FALLTHROUGH */
868     case (LIST_enum):
869         /* FALLTHROUGH */
870     case (LIST_hyphen):
871         /* FALLTHROUGH */
872     case (LIST_tag):
873         assert(width);
874         if (MDOC_HEAD == n->type)
875             p->rmargin = p->offset + width;

```

```

876         else
877             p->offset += width;
878         break;
879     case (LIST_column):
880         assert(width);
881         p->rmargin = p->offset + width;
882         /*
883         * XXX - this behaviour is not documented: the
884         * right-most column is filled to the right margin.
885         */
886         if (MDOC_HEAD == n->type)
887             break;
888         if (NULL == n->next && p->rmargin < p->maxrmargin)
889             p->rmargin = p->maxrmargin;
890         break;
891     default:
892         break;
893     }
895     /*
896     * The dash, hyphen, bullet and enum lists all have a special
897     * HEAD character (temporarily bold, in some cases).
898     */
900     if (MDOC_HEAD == n->type)
901         switch (type) {
902         case (LIST_bullet):
903             term_fontpush(p, TERMFONT_BOLD);
904             term_word(p, "\\[bu]");
905             term_fontpop(p);
906             break;
907         case (LIST_dash):
908             /* FALLTHROUGH */
909         case (LIST_hyphen):
910             term_fontpush(p, TERMFONT_BOLD);
911             term_word(p, "\\(hy)");
912             term_fontpop(p);
913             break;
914         case (LIST_enum):
915             (pair->ppair->ppair->count)++;
916             snprintf(buf, sizeof(buf), "%d.",
917                     pair->ppair->ppair->count);
918             term_word(p, buf);
919             break;
920         default:
921             break;
922         }
924     /*
925     * If we're not going to process our children, indicate so here.
926     */
928     switch (type) {
929     case (LIST_bullet):
930         /* FALLTHROUGH */
931     case (LIST_item):
932         /* FALLTHROUGH */
933     case (LIST_dash):
934         /* FALLTHROUGH */
935     case (LIST_hyphen):
936         /* FALLTHROUGH */
937     case (LIST_enum):
938         if (MDOC_HEAD == n->type)
939             return(0);
940         break;
941     case (LIST_column):

```

```

942         if (MDOC_HEAD == n->type)
943             return(0);
944         break;
945     default:
946         break;
947     }
949     return(1);
950 }

953 /* ARGSUSED */
954 static void
955 term_it_post(DECL_ARGS)
956 {
957     enum mdoc_list    type;

959     if (MDOC_BLOCK == n->type)
960         return;

962     type = n->parent->parent->parent->norm->Bl.type;

964     switch (type) {
965     case (LIST_item):
966         /* FALLTHROUGH */
967     case (LIST_diag):
968         /* FALLTHROUGH */
969     case (LIST_inset):
970         if (MDOC_BODY == n->type)
971             term_newln(p);
972         break;
973     case (LIST_column):
974         if (MDOC_BODY == n->type)
975             term_flushln(p);
976         break;
977     default:
978         term_newln(p);
979         break;
980     }

982     /*
983      * Now that our output is flushed, we can reset our tags. Since
984      * only 'It' sets these flags, we're free to assume that nobody
985      * has munged them in the meanwhile.
986      */

988     p->flags &= ~TERMP_DANGLE;
989     p->flags &= ~TERMP_NOBREAK;
992     p->flags &= ~TERMP_TWOSPACE;
990     p->flags &= ~TERMP_HANG;
991     p->trailspace = 0;
992 }

995 /* ARGSUSED */
996 static int
997 term_nm_pre(DECL_ARGS)
998 {

1000     if (MDOC_BLOCK == n->type) {
1001         p->flags |= TERMP_PREKEEP;
1002     } else if (MDOC_BLOCK == n->type)
1003         return(1);
1004     }

1005     if (MDOC_BODY == n->type) {

```

```

1006         if (NULL == n->child)
1007             return(0);
1008         p->flags |= TERMP_NOSPACE;
1009         p->offset += term_len(p, 1) +
1010             (NULL == n->prev->child ?
1011              term_strlen(p, meta->name) :
1012              (NULL == n->prev->child ? term_strlen(p, m->name) :
1013               MDOC_TEXT == n->prev->child->type ?
1014                term_strlen(p, n->prev->child->string) :
1015                term_len(p, 5)));
1016         return(1);
1017     }

1018     if (NULL == n->child && NULL == meta->name)
1019         if (NULL == n->child && NULL == m->name)
1020             return(0);

1021     if (MDOC_HEAD == n->type)
1022         synopsis_pre(p, n->parent);

1024     if (MDOC_HEAD == n->type && n->next->child) {
1025         p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;
1026         p->trailspace = 1;
1027         p->rmargin = p->offset + term_len(p, 1);
1028         if (NULL == n->child) {
1029             p->rmargin += term_strlen(p, meta->name);
1027             p->rmargin += term_strlen(p, m->name);
1030         } else if (MDOC_TEXT == n->child->type) {
1031             p->rmargin += term_strlen(p, n->child->string);
1032             if (n->child->next)
1033                 p->flags |= TERMP_HANG;
1034         } else {
1035             p->rmargin += term_len(p, 5);
1036             p->flags |= TERMP_HANG;
1037         }
1038     }

1040     term_fontpush(p, TERMFONT_BOLD);
1041     if (NULL == n->child)
1042         term_word(p, meta->name);
1040     term_word(p, m->name);
1043     return(1);
1044 }

1047 /* ARGSUSED */
1048 static void
1049 term_nm_post(DECL_ARGS)
1050 {

1052     if (MDOC_BLOCK == n->type) {
1053         p->flags &= ~(TERMP_KEEP | TERMP_PREKEEP);
1054     } else if (MDOC_HEAD == n->type && n->next->child) {
1050         if (MDOC_HEAD == n->type && n->next->child) {
1055             term_flushln(p);
1056             p->flags &= ~(TERMP_NOBREAK | TERMP_HANG);
1057             p->trailspace = 0;
1058         } else if (MDOC_BODY == n->type && n->child)
1059             term_flushln(p);
1060     }

    unchanged_portion_omitted

1377 static int
1378 term_vt_pre(DECL_ARGS)
1379 {

```

```

1381     if (MDOC_ELEM == n->type) {
1382         synopsis_pre(p, n);
1383         return(term_under_pre(p, pair, meta, n));
1384     } else if (MDOC_BLOCK == n->type) {
1385         synopsis_pre(p, n);
1386         return(1);
1387     } else if (MDOC_HEAD == n->type)
1388         return(0);

1390     return(term_under_pre(p, pair, meta, n));
1385     return(term_under_pre(p, pair, m, n));
1391 }
_____ unchanged_portion_omitted _____

1404 /* ARGSUSED */
1405 static int
1406 term_fd_pre(DECL_ARGS)
1407 {

1409     synopsis_pre(p, n);
1410     return(term_bold_pre(p, pair, meta, n));
1405     return(term_bold_pre(p, pair, m, n));
1411 }

1414 /* ARGSUSED */
1415 static void
1416 term_fd_post(DECL_ARGS)
1417 {

1419     term_newln(p);
1420 }

1423 /* ARGSUSED */
1424 static int
1425 term_sh_pre(DECL_ARGS)
1426 {

1428     /* No vspace between consecutive 'Sh' calls. */

1430     switch (n->type) {
1431     case (MDOC_BLOCK):
1432         if (n->prev && MDOC_Sh == n->prev->tok)
1433             if (NULL == n->prev->body->child)
1434                 break;
1435         term_vspace(p);
1436         break;
1437     case (MDOC_HEAD):
1438         term_fontpush(p, TERMFONT_BOLD);
1439         break;
1440     case (MDOC_BODY):
1441         p->offset = term_len(p, p->defindent);
1442         if (SEC_AUTHORS == n->sec)
1443             p->flags &= ~(TERMP_SPLIT|TERMP_NOSPLIT);
1444         break;
1445     default:
1446         break;
1447     }
1448     return(1);
1449 }
_____ unchanged_portion_omitted _____

```

```

1516 /* ARGSUSED */
1501 static void
1502 term_d1_post(DECL_ARGS)
1503 {

1505     if (MDOC_BLOCK != n->type)
1506         return;
1507     term_newln(p);
1508 }

1511 /* ARGSUSED */
1517 static int
1518 term_ft_pre(DECL_ARGS)
1519 {

1521     /* NB: MDOC_LINE does not effect this! */
1522     synopsis_pre(p, n);
1523     term_fontpush(p, TERMFONT_UNDER);
1524     return(1);
1525 }

1528 /* ARGSUSED */
1529 static int
1530 term_fn_pre(DECL_ARGS)
1531 {
1532     size_t      rmargin = 0;
1533     int         pretty;

1535     pretty = MDOC_SYNPRETTY & n->flags;

1537     synopsis_pre(p, n);

1539     if (NULL == (n = n->child))
1540         return(0);

1542     if (pretty) {
1543         rmargin = p->rmargin;
1544         p->rmargin = p->offset + term_len(p, 4);
1545         p->flags |= TERMP_NOBREAK | TERMP_HANG;
1546     }

1548     assert(MDOC_TEXT == n->type);
1549     term_fontpush(p, TERMFONT_BOLD);
1550     term_word(p, n->string);
1551     term_fontpop(p);

1553     if (pretty) {
1554         term_flushln(p);
1555         p->flags &= ~(TERMP_NOBREAK | TERMP_HANG);
1556         p->offset = p->rmargin;
1557         p->rmargin = rmargin;
1558     }

1560     p->flags |= TERMP_NOSPACE;
1561     term_word(p, "(");
1562     p->flags |= TERMP_NOSPACE;

1564     for (n = n->next; n; n = n->next) {
1565         assert(MDOC_TEXT == n->type);
1566         term_fontpush(p, TERMFONT_UNDER);
1567         if (pretty)
1568             p->flags |= TERMP_NBRWORD;
1569         term_word(p, n->string);

```

```

1570         term_fontpop(p);
1572         if (n->next) {
1573             p->flags |= TERMP_NOSPACE;
1574             term_word(p, ",");
1575         }
1576     }
1578     p->flags |= TERMP_NOSPACE;
1579     term_word(p, ",");
1581     if (pretty) {
1582         p->flags |= TERMP_NOSPACE;
1583         term_word(p, ",");
1584         term_flushln(p);
1585     }
1587     return(0);
1588 }

1591 /* ARGSUSED */
1592 static int
1593 term_fa_pre(DECL_ARGS)
1594 {
1595     const struct mdoc_node *nn;
1597     if (n->parent->tok != MDOC_Fo) {
1598         term_fontpush(p, TERMFONT_UNDER);
1599         return(1);
1600     }
1602     for (nn = n->child; nn; nn = nn->next) {
1603         term_fontpush(p, TERMFONT_UNDER);
1604         p->flags |= TERMP_NBRWORD;
1605         term_word(p, nn->string);
1606         term_fontpop(p);
1608         if (nn->next || (n->next && n->next->tok == MDOC_Fa)) {
1609             if (nn->next) {
1610                 p->flags |= TERMP_NOSPACE;
1611                 term_word(p, ",");
1612             }
1614             if (n->child && n->next && n->next->tok == MDOC_Fa) {
1615                 p->flags |= TERMP_NOSPACE;
1616                 term_word(p, ",");
1617             }
1618         }
1619     }
1620     return(0);
1621 }

1622 /* ARGSUSED */
1623 static int
1624 term_bd_pre(DECL_ARGS)
1625 {
1626     size_t          tabwidth, rm, rmax;
1627     struct mdoc_node *nn;
1628     const struct mdoc_node *nn;
1630     if (MDOC_BLOCK == n->type) {
1631         print_bvspace(p, n, n);
1632         return(1);
1633     } else if (MDOC_HEAD == n->type)

```

```

1629         return(0);
1631     if (n->norm->Bd.offfs)
1632         p->offset += a2offs(p, n->norm->Bd.offfs);
1634     /*
1635      * If -ragged or -filled are specified, the block does nothing
1636      * but change the indentation.  If -unfilled or -literal are
1637      * specified, text is printed exactly as entered in the display:
1638      * for macro lines, a newline is appended to the line.  Blank
1639      * lines are allowed.
1640      */
1641     if (DISP_literal != n->norm->Bd.type &&
1642         DISP_unfilled != n->norm->Bd.type)
1643         return(1);
1644     tabwidth = p->tabwidth;
1645     if (DISP_literal == n->norm->Bd.type)
1646         p->tabwidth = term_len(p, 8);
1648     rm = p->rmargin;
1649     rmax = p->maxrmargin;
1650     p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
1652     for (nn = n->child; nn; nn = nn->next) {
1653         print_mdoc_node(p, pair, meta, nn);
1654         print_mdoc_node(p, pair, m, nn);
1655     /*
1656      * If the printed node flushes its own line, then we
1657      * needn't do it here as well.  This is hacky, but the
1658      * notion of selective eoln whitespace is pretty dumb
1659      * anyway, so don't sweat it.
1660      */
1661     switch (nn->tok) {
1662     case (MDOC_Sm):
1663         /* FALLTHROUGH */
1664     case (MDOC_br):
1665         /* FALLTHROUGH */
1666     case (MDOC_sp):
1667         /* FALLTHROUGH */
1668     case (MDOC_Bl):
1669         /* FALLTHROUGH */
1670     case (MDOC_Dl):
1671         /* FALLTHROUGH */
1672     case (MDOC_Dl):
1673         /* FALLTHROUGH */
1674     case (MDOC_Lp):
1675         /* FALLTHROUGH */
1676     case (MDOC_Pp):
1677             continue;
1678     default:
1679         break;
1680     }
1681     if (nn->next && nn->next->line == nn->line)
1682         continue;
1683     term_flushln(p);
1684     p->flags |= TERMP_NOSPACE;
1685 }
1686 }

1688     p->tabwidth = tabwidth;
1689     p->rmargin = rm;
1690     p->maxrmargin = rmax;
1691     return(0);
1692 }

```

unchanged portion omitted

```

1744 /* ARGSUSED */
1745 static int
1746 term_p_xx_pre(DECL_ARGS)
1747 {
1748     const char    *pp;
1749     int            flags;

1751     pp = NULL;
1752     switch (n->tok) {
1753     case (MDOC_Bsx):
1754         pp = "BSD/OS";
1755         break;
1756     case (MDOC_Dx):
1757         pp = "DragonFly";
1758         break;
1759     case (MDOC_Fx):
1760         pp = "FreeBSD";
1761         break;
1762     case (MDOC_Nx):
1763         pp = "NetBSD";
1764         break;
1765     case (MDOC_Ox):
1766         pp = "OpenBSD";
1767         break;
1768     case (MDOC_Ux):
1769         pp = "UNIX";
1770         break;
1771     default:
1772         abort();
1773         /* NOTREACHED */
1774         break;
1775     }

1776     term_word(p, pp);
1777     if (n->child) {
1778         flags = p->flags;
1779         p->flags |= TERMP_KEEP;
1780         term_word(p, n->child->string);
1781         p->flags = flags;
1782     }
1783     return(0);
1784 }

```

```

1787 /* ARGSUSED */
1788 static int
1789 term_p_igndelim_pre(DECL_ARGS)
1790 {

```

```

1793     p->flags |= TERMP_IGNDELIM;
1794     return(1);
1795 }

```

```

1798 /* ARGSUSED */
1799 static void
1800 term_p_pf_post(DECL_ARGS)
1801 {

```

```

1804     p->flags |= TERMP_NOSPACE;
1805 }

```

unchanged portion omitted

```

1904 /* ARGSUSED */
1905 static int
1906 term_p_quote_pre(DECL_ARGS)
1907 {
1909     if (MDOC_BODY != n->type && MDOC_ELEM != n->type)
1910         return(1);

1912     switch (n->tok) {
1913     case (MDOC_Ao):
1914         /* FALLTHROUGH */
1915     case (MDOC_Aq):
1916         term_word(p, "<");
1917         break;
1918     case (MDOC_Bro):
1919         /* FALLTHROUGH */
1920     case (MDOC_Brq):
1921         term_word(p, "{");
1922         break;
1923     case (MDOC_Oo):
1924         /* FALLTHROUGH */
1925     case (MDOC_Op):
1926         /* FALLTHROUGH */
1927     case (MDOC_Bo):
1928         /* FALLTHROUGH */
1929     case (MDOC_Bq):
1930         term_word(p, "[");
1931         break;
1932     case (MDOC_Do):
1933         /* FALLTHROUGH */
1934     case (MDOC_Dq):
1935         term_word(p, "\\(lq");
1936         term_word(p, "'");
1937         break;
1938     case (MDOC_Eo):
1939         break;
1940     case (MDOC_Po):
1941         /* FALLTHROUGH */
1942     case (MDOC_Pq):
1943         term_word(p, "(");
1944         break;
1945     case (MDOC__T):
1946         /* FALLTHROUGH */
1947     case (MDOC_Qo):
1948         /* FALLTHROUGH */
1949     case (MDOC_Qq):
1950         term_word(p, "\\(oq");
1951         term_word(p, "'");
1952         break;
1953     case (MDOC_Ql):
1954         /* FALLTHROUGH */
1955     case (MDOC_So):
1956         /* FALLTHROUGH */
1957     case (MDOC_Sq):
1958         term_word(p, "\\(oq");
1959         term_word(p, "'");
1960         break;
1961     default:
1962         abort();
1963         /* NOTREACHED */
1964     }

1965     p->flags |= TERMP_NOSPACE;
1966     return(1);
1967 }

```



```

1968 /* ARGSUSED */
1969 static void
1970 term_quote_post(DECL_ARGS)
1971 {
1973     if (MDOC_BODY != n->type && MDOC_ELEM != n->type)
1974         return;
1976     p->flags |= TERMP_NOSPACE;
1978     switch (n->tok) {
1979     case (MDOC_Ao):
1980         /* FALLTHROUGH */
1981     case (MDOC_Aq):
1982         term_word(p, ">");
1983         break;
1984     case (MDOC_Bro):
1985         /* FALLTHROUGH */
1986     case (MDOC_Brq):
1987         term_word(p, " ");
1988         break;
1989     case (MDOC_Oo):
1990         /* FALLTHROUGH */
1991     case (MDOC_Op):
1992         /* FALLTHROUGH */
1993     case (MDOC_Bo):
1994         /* FALLTHROUGH */
1995     case (MDOC_Bq):
1996         term_word(p, "]");
1997         break;
1998     case (MDOC_Do):
1999         /* FALLTHROUGH */
2000     case (MDOC_Dq):
2001         term_word(p, "\\(rq");
2002         term_word(p, "'");
2003         break;
2004     case (MDOC_Eo):
2005         break;
2006     case (MDOC_Po):
2007         /* FALLTHROUGH */
2008     case (MDOC_Pq):
2009         term_word(p, " ");
2010         break;
2011     case (MDOC_T):
2012         /* FALLTHROUGH */
2013     case (MDOC_Qo):
2014         /* FALLTHROUGH */
2015     case (MDOC_Qq):
2016         term_word(p, "\\");
2017         break;
2018     case (MDOC_Ql):
2019         /* FALLTHROUGH */
2020     case (MDOC_So):
2021         /* FALLTHROUGH */
2022     case (MDOC_Sq):
2023         term_word(p, "\\(cq");
2024         term_word(p, "'");
2025         break;
2026     default:
2027         abort();
2028         /* NOTREACHED */
2029     }
2030 }

```

```
2031 /* ARGSUSED */
```

```

2032 static int
2033 term_fo_pre(DECL_ARGS)
2034 {
2035     size_t      rmargin = 0;
2036     int         pretty;
2038     pretty = MDOC_SYNPRETTY & n->flags;
2040     if (MDOC_BLOCK == n->type) {
2041         synopsis_pre(p, n);
2042         return(1);
2043     } else if (MDOC_BODY == n->type) {
2044         if (pretty) {
2045             rmargin = p->rmargin;
2046             p->rmargin = p->offset + term_len(p, 4);
2047             p->flags |= TERMP_NOBREAK | TERMP_HANG;
2048         }
2049         p->flags |= TERMP_NOSPACE;
2050         term_word(p, "(");
2051         p->flags |= TERMP_NOSPACE;
2052         if (pretty) {
2053             term_flushln(p);
2054             p->flags &= ~(TERMP_NOBREAK | TERMP_HANG);
2055             p->offset = p->rmargin;
2056             p->rmargin = rmargin;
2057         }
2058         return(1);
2059     }
2061     if (NULL == n->child)
2062         return(0);
2064     /* XXX: we drop non-initial arguments as per groff. */
2066     assert(n->child->string);
2067     term_fontpush(p, TERMPFONT_BOLD);
2068     term_word(p, n->child->string);
2069     return(0);
2070 }
2072 /* ARGSUSED */
2073 static void
2074 term_fo_post(DECL_ARGS)
2075 {
2078     if (MDOC_BODY != n->type)
2079         return;
2081     p->flags |= TERMP_NOSPACE;
2082     term_word(p, " ");
2084     if (MDOC_SYNPRETTY & n->flags) {
2085         p->flags |= TERMP_NOSPACE;
2086         term_word(p, ";");
2087         term_flushln(p);
2088     }
2089 }
2092 /* ARGSUSED */
2093 static int
2094 term_bf_pre(DECL_ARGS)
2095 {
2097     if (MDOC_HEAD == n->type)

```

```

2098         return(0);
2099     else if (MDOC_BODY != n->type)
2074     else if (MDOC_BLOCK != n->type)
2100         return(1);

2102     if (FONT_Em == n->norm->Bf.font)
2103         term_fontpush(p, TERMFONT_UNDER);
2104     else if (FONT_Sy == n->norm->Bf.font)
2105         term_fontpush(p, TERMFONT_BOLD);
2106     else
2107         term_fontpush(p, TERMFONT_NONE);

2109     return(1);
2110 }

```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```

2181 /* ARGSUSED */
2182 static int
2183 term_lk_pre(DECL_ARGS)
2184 {
2185     const struct mdoc_node *link, *descr;
2186     const struct mdoc_node *nn, *sv;

2187     if (NULL == (link = n->child))
2188         return(0);

2190     if (NULL != (descr = link->next)) {
2191         term_fontpush(p, TERMFONT_UNDER);
2192         while (NULL != descr) {
2193             term_word(p, descr->string);
2194             descr = descr->next;
2195         }

2164     nn = sv = n->child;

2166     if (NULL == nn || NULL == nn->next)
2167         return(1);

2169     for (nn = nn->next; nn; nn = nn->next)
2170         term_word(p, nn->string);

2172     term_fontpop(p);

2196     p->flags |= TERMP_NOSPACE;
2197     term_word(p, ":");
2198     term_fontpop(p);
2199 }

2201     term_fontpush(p, TERMFONT_BOLD);
2202     term_word(p, link->string);
2178     term_word(p, sv->string);
2203     term_fontpop(p);

```

```

2205     return(0);
2206 }

```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```

2241 /* ARGSUSED */
2242 static void
2243 term__t_post(DECL_ARGS)
2244 {

```

```

2246     /*
2247     * If we're in an 'Rs' and there's a journal present, then quote
2248     * us instead of underlining us (for disambiguation).

```

```

2249     /*
2250     if (n->parent && MDOC_Rs == n->parent->tok &&
2251         n->parent->norm->Rs.quote_T)
2252         term_quote_post(p, pair, meta, n);
2228         term_quote_post(p, pair, m, n);

```

```

2254     term__post(p, pair, meta, n);
2230     term__post(p, pair, m, n);
2255 }

```

```

2257 /* ARGSUSED */
2258 static int
2259 term__t_pre(DECL_ARGS)
2260 {

```

```

2262     /*
2263     * If we're in an 'Rs' and there's a journal present, then quote
2264     * us instead of underlining us (for disambiguation).
2265     */
2266     if (n->parent && MDOC_Rs == n->parent->tok &&
2267         n->parent->norm->Rs.quote_T)
2268         return(term_quote_pre(p, pair, meta, n));
2244         return(term_quote_pre(p, pair, m, n));

```

```

2270     term_fontpush(p, TERMFONT_UNDER);
2271     return(1);
2272 }

```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

new/usr/src/cmd/mandoc/mdoc\_validate.c

1

```
*****
55544 Fri Jul 25 15:00:14 2014
new/usr/src/cmd/mandoc/mdoc_validate.c
Update to 1.12.3.
*****
1 /* $Id: mdoc_validate.c,v 1.198 2013/12/15 21:23:52 schwarze Exp $ */
1 /* $Id: mdoc_validate.c,v 1.182 2012/03/23 05:50:25 kristaps Exp $ */
2 /*
3  * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #ifndef OSNAME
23 #include <sys/utsname.h>
24 #endif
25
26 #include <sys/types.h>
27
28 #include <assert.h>
29 #include <ctype.h>
30 #include <limits.h>
31 #include <stdio.h>
32 #include <stdlib.h>
33 #include <string.h>
34 #include <time.h>
35
36 #include "mdoc.h"
37 #include "mandoc.h"
38 #include "libmdoc.h"
39 #include "libmandoc.h"
40
41 /* FIXME: .Bl -diag can't have non-text children in HEAD. */
42
43 #define PRE_ARGS struct mdoc *mdoc, struct mdoc_node *n
44 #define POST_ARGS struct mdoc *mdoc
45
46 #define NUMSIZ 32
47 #define DATESIZE 32
48
49 enum check_ineq {
50     CHECK_LT,
51     CHECK_GT,
52     CHECK_EQ
53 };
54
55 unchanged_portion_omitted
56
68 static int check_count(struct mdoc *, enum mdoc_type,
69                        enum check_lvl, enum check_ineq, int);
70 static int check_parent(PRE_ARGS, enum mdoct, enum mdoc_type);
```

new/usr/src/cmd/mandoc/mdoc\_validate.c

2

```
71 static void check_text(struct mdoc *, int, int, char *);
72 static void check_argv(struct mdoc *,
73                        struct mdoc_node *, struct mdoc_argv *);
74 static void check_args(struct mdoc *, struct mdoc_node *);
75 static int concat(char *, const struct mdoc_node *, size_t);
76 static enum mdoc_sec a2sec(const char *);
77 static size_t macro2len(enum mdoct);
78
79 static int ebool(POST_ARGS);
80 static int berr_get(POST_ARGS);
81 static int bwarn_get(POST_ARGS);
82 static int ewarn_eq0(POST_ARGS);
83 static int ewarn_eq1(POST_ARGS);
84 static int ewarn_get1(POST_ARGS);
85 static int ewarn_le1(POST_ARGS);
86 static int hwarn_eq0(POST_ARGS);
87 static int hwarn_eq1(POST_ARGS);
88 static int hwarn_get1(POST_ARGS);
89 static int hwarn_le1(POST_ARGS);
90
91 static int post_an(POST_ARGS);
92 static int post_at(POST_ARGS);
93 static int post_bf(POST_ARGS);
94 static int post_bl(POST_ARGS);
95 static int post_bl_block(POST_ARGS);
96 static int post_bl_block_width(POST_ARGS);
97 static int post_bl_block_tag(POST_ARGS);
98 static int post_bl_head(POST_ARGS);
99 static int post_bx(POST_ARGS);
100 static int post_defaults(POST_ARGS);
101 static int post_dd(POST_ARGS);
102 static int post_dt(POST_ARGS);
103 static int post_defaults(POST_ARGS);
104 static int post_literal(POST_ARGS);
105 static int post_eoln(POST_ARGS);
106 static int post_hyph(POST_ARGS);
107 static int post_ignpar(POST_ARGS);
108 static int post_it(POST_ARGS);
109 static int post_lb(POST_ARGS);
110 static int post_literal(POST_ARGS);
111 static int post_nm(POST_ARGS);
112 static int post_ns(POST_ARGS);
113 static int post_os(POST_ARGS);
114 static int post_par(POST_ARGS);
115 static int post_ignpar(POST_ARGS);
116 static int post_prol(POST_ARGS);
117 static int post_root(POST_ARGS);
118 static int post_rs(POST_ARGS);
119 static int post_sh(POST_ARGS);
120 static int post_sh_body(POST_ARGS);
121 static int post_sh_head(POST_ARGS);
122 static int post_st(POST_ARGS);
123 static int post_std(POST_ARGS);
124 static int post_vt(POST_ARGS);
125 static int pre_an(PRE_ARGS);
126 static int pre_bd(PRE_ARGS);
127 static int pre_bl(PRE_ARGS);
128 static int pre_dd(PRE_ARGS);
129 static int pre_display(PRE_ARGS);
130 static int pre_dt(PRE_ARGS);
131 static int pre_it(PRE_ARGS);
132 static int pre_literal(PRE_ARGS);
133 static int pre_os(PRE_ARGS);
134 static int pre_par(PRE_ARGS);
135 static int pre_sh(PRE_ARGS);
136 static int pre_ss(PRE_ARGS);
```

```

134 static int pre_std(PRE_ARGS);

136 static v_post posts_an[] = { post_an, NULL };
137 static v_post posts_at[] = { post_at, post_defaults, NULL };
138 static v_post posts_bd[] = { post_literal, hwarn_eq0, bwarn_gel, NULL };
139 static v_post posts_bf[] = { hwarn_le1, post_bf, NULL };
140 static v_post posts_bk[] = { hwarn_eq0, bwarn_gel, NULL };
141 static v_post posts_bl[] = { bwarn_gel, post_bl, NULL };
142 static v_post posts_bx[] = { post_bx, NULL };
143 static v_post posts_bool[] = { ebool, NULL };
144 static v_post posts_eoln[] = { post_eoln, NULL };
145 static v_post posts_defaults[] = { post_defaults, NULL };
146 static v_post posts_dl[] = { bwarn_gel, post_hyph, NULL };
147 static v_post posts_dd[] = { post_dd, post_prol, NULL };
148 static v_post posts_dl[] = { post_literal, bwarn_gel, NULL };
149 static v_post posts_dt[] = { post_dt, post_prol, NULL };
150 static v_post posts_fo[] = { hwarn_eql, bwarn_gel, NULL };
151 static v_post posts_hyph[] = { post_hyph, NULL };
152 static v_post posts_hyptext[] = { ewarn_gel, post_hyph, NULL };
153 static v_post posts_it[] = { post_it, NULL };
154 static v_post posts_lb[] = { post_lb, NULL };
155 static v_post posts_nd[] = { berr_gel, post_hyph, NULL };
156 static v_post posts_nd[] = { berr_gel, NULL };
157 static v_post posts_nm[] = { post_nm, NULL };
158 static v_post posts_notext[] = { ewarn_eq0, NULL };
159 static v_post posts_ns[] = { post_ns, NULL };
160 static v_post posts_os[] = { post_os, post_prol, NULL };
161 static v_post posts_pp[] = { post_par, ewarn_eq0, NULL };
162 static v_post posts_rs[] = { post_rs, NULL };
163 static v_post posts_sh[] = { post_ignpar, hwarn_gel, post_sh, post_hyph, NULL };
164 static v_post posts_sh[] = { post_par, ewarn_le1, NULL };
165 static v_post posts_ss[] = { post_ignpar, hwarn_gel, post_hyph, NULL };
166 static v_post posts_sh[] = { post_ignpar, hwarn_gel, post_sh, NULL };
167 static v_post posts_sp[] = { ewarn_le1, NULL };
168 static v_post posts_ss[] = { post_ignpar, hwarn_gel, NULL };
169 static v_post posts_st[] = { post_st, NULL };
170 static v_post posts_std[] = { post_std, NULL };
171 static v_post posts_text[] = { ewarn_gel, NULL };
172 static v_post posts_text1[] = { ewarn_eql, NULL };
173 static v_post posts_vt[] = { post_vt, NULL };
174 static v_post posts_wline[] = { bwarn_gel, NULL };
175 static v_pre pres_an[] = { pre_an, NULL };
176 static v_pre pres_bd[] = { pre_display, pre_bd, pre_literal, pre_par, NULL };
177 static v_pre pres_bl[] = { pre_bl, pre_par, NULL };
178 static v_pre pres_dl[] = { pre_display, NULL };
179 static v_pre pres_dl[] = { pre_literal, pre_display, NULL };
180 static v_pre pres_dd[] = { pre_dd, NULL };
181 static v_pre pres_dt[] = { pre_dt, NULL };
182 static v_pre pres_er[] = { NULL, NULL };
183 static v_pre pres_fd[] = { NULL, NULL };
184 static v_pre pres_it[] = { pre_it, pre_par, NULL };
185 static v_pre pres_os[] = { pre_os, NULL };
186 static v_pre pres_pp[] = { pre_par, NULL };
187 static v_pre pres_sh[] = { pre_sh, NULL };
188 static v_pre pres_ss[] = { pre_ss, NULL };
189 static v_pre pres_std[] = { pre_std, NULL };

184 static const struct valids mdoc_valids[MDOC_MAX] = {
185     { NULL, NULL }, /* Ap */
186     { pres_dd, posts_dd }, /* Dd */
187     { pres_dt, posts_dt }, /* Dt */
188     { pres_os, posts_os }, /* Os */
189     { pres_sh, posts_sh }, /* Sh */
190     { pres_ss, posts_ss }, /* Ss */
191     { pres_pp, posts_pp }, /* Pp */
192     { pres_dl, posts_dl }, /* Dl */

```

```

188     { pres_pp, posts_notext }, /* Pp */
189     { pres_dl, posts_wline }, /* Dl */
193     { pres_dl, posts_dl }, /* Dl */
194     { pres_bd, posts_bd }, /* Bd */
195     { NULL, NULL }, /* Ed */
196     { pres_bl, posts_bl }, /* Bl */
197     { NULL, NULL }, /* El */
198     { pres_it, posts_it }, /* It */
199     { NULL, NULL }, /* Ad */
200     { pres_an, posts_an }, /* An */
201     { NULL, posts_defaults }, /* Ar */
202     { NULL, NULL }, /* Cd */
203     { NULL, NULL }, /* Cm */
204     { NULL, NULL }, /* Dv */
205     { NULL, NULL }, /* Er */
206     { pres_er, NULL }, /* Er */
207     { NULL, NULL }, /* Ev */
208     { pres_std, posts_std }, /* Ex */
209     { NULL, NULL }, /* Fa */
210     { NULL, posts_text }, /* Fd */
211     { pres_fd, posts_text }, /* Fd */
212     { NULL, NULL }, /* Fl */
213     { NULL, NULL }, /* Fn */
214     { NULL, NULL }, /* Ft */
215     { NULL, NULL }, /* Ic */
216     { NULL, posts_text1 }, /* In */
217     { NULL, posts_defaults }, /* Li */
218     { NULL, posts_nd }, /* Nd */
219     { NULL, posts_nm }, /* Nm */
220     { NULL, NULL }, /* Op */
221     { NULL, NULL }, /* Ot */
222     { NULL, posts_defaults }, /* Pa */
223     { pres_std, posts_std }, /* Rv */
224     { NULL, posts_st }, /* St */
225     { NULL, NULL }, /* Va */
226     { NULL, posts_vt }, /* Vt */
227     { NULL, posts_text }, /* Xr */
228     { NULL, posts_text }, /* %A */
229     { NULL, posts_hyptext }, /* %B */ /* FIXME: can be used o
230     { NULL, posts_text }, /* %B */ /* FIXME: can be used o
231     { NULL, posts_text }, /* %D */
232     { NULL, posts_text }, /* %I */
233     { NULL, posts_text }, /* %J */
234     { NULL, posts_hyptext }, /* %N */
235     { NULL, posts_hyptext }, /* %O */
236     { NULL, posts_text }, /* %N */
237     { NULL, posts_text }, /* %O */
238     { NULL, posts_text }, /* %P */
239     { NULL, posts_hyptext }, /* %R */
240     { NULL, posts_hyptext }, /* %T */ /* FIXME: can be used o
241     { NULL, posts_text }, /* %R */
242     { NULL, posts_text }, /* %T */ /* FIXME: can be used o
243     { NULL, posts_text }, /* %V */
244     { NULL, NULL }, /* Ac */
245     { NULL, NULL }, /* Ao */
246     { NULL, NULL }, /* Aq */
247     { NULL, posts_at }, /* At */
248     { NULL, NULL }, /* Bc */
249     { NULL, posts_bf }, /* Bf */
250     { NULL, NULL }, /* Bo */
251     { NULL, NULL }, /* Bq */
252     { NULL, NULL }, /* Bsx */
253     { NULL, posts_bx }, /* Bx */
254     { NULL, posts_bool }, /* Db */
255     { NULL, NULL }, /* Dc */
256     { NULL, NULL }, /* Do */

```

```

250 { NULL, NULL }, /* Dq */
251 { NULL, NULL }, /* Ec */
252 { NULL, NULL }, /* Ef */
253 { NULL, NULL }, /* Em */
254 { NULL, NULL }, /* Eo */
255 { NULL, NULL }, /* Fx */
256 { NULL, NULL }, /* Ms */
257 { NULL, posts_notext }, /* No */
258 { NULL, posts_ns }, /* Ns */
259 { NULL, NULL }, /* Nx */
260 { NULL, NULL }, /* Ox */
261 { NULL, NULL }, /* Pc */
262 { NULL, posts_text1 }, /* Pf */
263 { NULL, NULL }, /* Po */
264 { NULL, NULL }, /* Pq */
265 { NULL, NULL }, /* Qc */
266 { NULL, NULL }, /* Ql */
267 { NULL, NULL }, /* Qo */
268 { NULL, NULL }, /* Qq */
269 { NULL, NULL }, /* Re */
270 { NULL, posts_rs }, /* Rs */
271 { NULL, NULL }, /* Sc */
272 { NULL, NULL }, /* So */
273 { NULL, NULL }, /* Sq */
274 { NULL, posts_bool }, /* Sm */
275 { NULL, posts_hyph }, /* Sx */
276 { NULL, NULL }, /* Sx */
277 { NULL, NULL }, /* Sy */
278 { NULL, NULL }, /* Tn */
279 { NULL, NULL }, /* Ux */
280 { NULL, NULL }, /* Xc */
281 { NULL, posts_fo }, /* Xo */
282 { NULL, NULL }, /* Fc */
283 { NULL, NULL }, /* Oo */
284 { NULL, NULL }, /* Oc */
285 { NULL, posts_bk }, /* Bk */
286 { NULL, NULL }, /* Ek */
287 { NULL, posts_eoln }, /* Bt */
288 { NULL, NULL }, /* Hf */
289 { NULL, NULL }, /* Fr */
290 { NULL, posts_eoln }, /* Ud */
291 { NULL, posts_lb }, /* Lb */
292 { pres_pp, posts_pp }, /* Lp */
289 { NULL, posts_notext }, /* Lp */
293 { NULL, NULL }, /* Lk */
294 { NULL, posts_defaults }, /* Mt */
295 { NULL, NULL }, /* Brq */
296 { NULL, NULL }, /* Bro */
297 { NULL, NULL }, /* Brc */
298 { NULL, posts_text }, /* %C */
299 { NULL, NULL }, /* Es */
300 { NULL, NULL }, /* En */
301 { NULL, NULL }, /* Dx */
302 { NULL, posts_text }, /* %Q */
303 { NULL, posts_pp }, /* br */
304 { NULL, posts_sp }, /* sp */
300 { NULL, posts_notext }, /* br */
301 { pres_pp, posts_sp }, /* sp */
305 { NULL, posts_text1 }, /* %U */
306 { NULL, NULL }, /* Ta */
307 };

```

```
309 #define RSORD_MAX 14 /* Number of 'Rs' blocks. */
```

```
311 static const enum mdoct rsord[RSORD_MAX] = {
```

```

312 MDOC_A,
313 MDOC_T,
314 MDOC_B,
315 MDOC_I,
316 MDOC_J,
317 MDOC_R,
318 MDOC_N,
319 MDOC_V,
320 MDOC_U,
321 MDOC_P,
322 MDOC_Q,
323 MDOC_C,
324 MDOC_D,
325 MDOC_O,
320 MDOC_O,
321 MDOC_C,
322 MDOC_U
326 };
    unchanged_portion_omitted

419 static int
420 check_count(struct mdoc *mdoc, enum mdoc_type type,
417 check_count(struct mdoc *m, enum mdoc_type type,
421 enum check_lvl lvl, enum check_ineq ineq, int val)
422 {
423     const char *p;
424     enum mandocerr t;

426     if (mdoc->last->type != type)
423     if (m->last->type != type)
427         return(1);
428
429     switch (ineq) {
430     case (CHECK_LT):
431         p = "less than ";
432         if (mdoc->last->nchild < val)
429         if (m->last->nchild < val)
433             return(1);
434         break;
435     case (CHECK_GT):
436         p = "more than ";
437         if (mdoc->last->nchild > val)
434         if (m->last->nchild > val)
438             return(1);
439         break;
440     case (CHECK_EQ):
441         p = "";
442         if (val == mdoc->last->nchild)
439         if (val == m->last->nchild)
443             return(1);
444         break;
445     default:
446         abort();
447         /* NOTREACHED */
448     }

450     t = lvl == CHECK_WARN ? MANDOCERR_ARGCWARN : MANDOCERR_ARGCOUNT;
451     mandoc_vmsg(t, mdoc->parse, mdoc->last->line, mdoc->last->pos,
448     mandoc_vmsg(t, m->parse, m->last->line, m->last->pos,
452         "want %s%d children (have %d)",
453         p, val, mdoc->last->nchild);
450         p, val, m->last->nchild);
454     return(1);
455 }
    unchanged_portion_omitted

```

```

518 static void
519 check_args(struct mdoc *mdoc, struct mdoc_node *n)
520 {
521     int i;
522
523     if (NULL == n->args)
524         return;
525
526     assert(n->args->argc);
527     for (i = 0; i < (int)n->args->argc; i++)
528         check_argv(mdoc, n, &n->args->argv[i]);
529     check_argv(m, n, &n->args->argv[i]);
530 }
531
532 static void
533 check_argv(struct mdoc *mdoc, struct mdoc_node *n, struct mdoc_argv *v)
534 {
535     int i;
536
537     for (i = 0; i < (int)v->sz; i++)
538         check_text(mdoc, v->line, v->pos, v->value[i]);
539     check_text(m, v->line, v->pos, v->value[i]);
540 }
541
542 /* FIXME: move to post_std(). */
543
544 if (MDOC_Std == v->arg)
545     if ( ! (v->sz || mdoc->meta.name)
546         mdoc_nmsg(mdoc, n, MANDOCERR_NONAME);
547     if ( ! (v->sz || m->meta.name)
548         mdoc_nmsg(m, n, MANDOCERR_NONAME);
549 }
550
551 static void
552 check_text(struct mdoc *mdoc, int ln, int pos, char *p)
553 {
554     check_text(struct mdoc *m, int ln, int pos, char *p)
555     {
556         char *cp;
557
558         if (MDOC_LITERAL & mdoc->flags)
559             if (MDOC_LITERAL & m->flags)
560                 return;
561
562         for (cp = p; NULL != (p = strchr(p, '\t')); p++)
563             mdoc_pmsg(mdoc, ln, pos + (int)(p - cp), MANDOCERR_BADTAB);
564         mdoc_pmsg(m, ln, pos + (int)(p - cp), MANDOCERR_BADTAB);
565     }
566 }
567
568 unchanged_portion_omitted
569
570 static int
571 pre_bl(PRE_ARGS)
572 {
573     int i, comp, dup;
574     const char *offs, *width;
575     enum mdoc_list lt;
576     struct mdoc_node *np;
577
578     if (MDOC_BLOCK != n->type) {
579         if (ENDBODY_NOT != n->end) {
580             assert(n->pending);
581             np = n->pending->parent;
582         } else
583             np = n->parent;
584     }
585 }

```

```

609     assert(np);
610     assert(MDOC_BLOCK == np->type);
611     assert(MDOC_Bl == np->tok);
612     return(1);
613 }
614
615 /*
616  * First figure out which kind of list to use: bind ourselves to
617  * the first mentioned list type and warn about any remaining
618  * ones. If we find no list type, we default to LIST_item.
619  */
620
621 /* LINTED */
622 for (i = 0; n->args && i < (int)n->args->argc; i++) {
623     lt = LIST_NONE;
624     dup = comp = 0;
625     width = offs = NULL;
626     switch (n->args->argv[i].arg) {
627         /* Set list types. */
628         case (MDOC_Bullet):
629             lt = LIST_bullet;
630             break;
631         case (MDOC_Dash):
632             lt = LIST_dash;
633             break;
634         case (MDOC_Enum):
635             lt = LIST_enum;
636             break;
637         case (MDOC_Hyphen):
638             lt = LIST_hyphen;
639             break;
640         case (MDOC_Item):
641             lt = LIST_item;
642             break;
643         case (MDOC_Tag):
644             lt = LIST_tag;
645             break;
646         case (MDOC_Diag):
647             lt = LIST_diag;
648             break;
649         case (MDOC_Hang):
650             lt = LIST_hang;
651             break;
652         case (MDOC_Ohang):
653             lt = LIST_ohang;
654             break;
655         case (MDOC_Inset):
656             lt = LIST_inset;
657             break;
658         case (MDOC_Column):
659             lt = LIST_column;
660             break;
661         /* Set list arguments. */
662         case (MDOC_Compact):
663             dup = n->norm->Bl.comp;
664             comp = 1;
665             break;
666         case (MDOC_Width):
667             /* NB: this can be empty! */
668             if (n->args->argv[i].sz) {
669                 width = n->args->argv[i].value[0];
670                 dup = (NULL != n->norm->Bl.width);
671                 break;
672             }
673             mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
674             break;

```

```

675     case (MDOC_Offset):
676         /* NB: this can be empty! */
677         if (n->args->argv[i].sz) {
678             offs = n->args->argv[i].value[0];
679             dup = (NULL != n->norm->Bl.offss);
680             break;
681         }
682         mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
683         break;
684     default:
685         continue;
686 }

688 /* Check: duplicate auxiliary arguments. */

690 if (dup)
691     mdoc_nmsg(mdoc, n, MANDOCERR_ARGVREP);

693 if (comp && ! dup)
694     n->norm->Bl.comp = comp;
695 if (offs && ! dup)
696     n->norm->Bl.offss = offs;
697 if (width && ! dup)
698     n->norm->Bl.width = width;

700 /* Check: multiple list types. */

702 if (LIST_NONE != lt && n->norm->Bl.type != LIST_NONE)
703     mdoc_nmsg(mdoc, n, MANDOCERR_LISTREP);

705 /* Assign list type. */

707 if (LIST_NONE != lt && n->norm->Bl.type == LIST_NONE) {
708     n->norm->Bl.type = lt;
709     /* Set column information, too. */
710     if (LIST_column == lt) {
711         n->norm->Bl.ncols =
712             n->args->argv[i].sz;
713         n->norm->Bl.cols = (void *)
714             n->args->argv[i].value;
715     }
716 }

718 /* The list type should come first. */

720 if (n->norm->Bl.type == LIST_NONE)
721     if (n->norm->Bl.width ||
722         n->norm->Bl.offss ||
723         n->norm->Bl.comp)
724         mdoc_nmsg(mdoc, n, MANDOCERR_LISTFIRST);

726     continue;
727 }

729 /* Allow lists to default to LIST_item. */

731 if (LIST_NONE == n->norm->Bl.type) {
732     mdoc_nmsg(mdoc, n, MANDOCERR_LISTTYPE);
733     n->norm->Bl.type = LIST_item;
734 }

736 /*
737 * Validate the width field. Some list types don't need width
738 * types and should be warned about them. Others should have it
739 * and must also be warned. Yet others have a default and need
740 * no warning.

```

```

736     * and must also be warned.
741     */

743     switch (n->norm->Bl.type) {
744     case (LIST_tag):
745         if (NULL == n->norm->Bl.width)
746             if (n->norm->Bl.width)
747                 break;
748         mdoc_nmsg(mdoc, n, MANDOCERR_NOWIDTHARG);
749         break;
750     case (LIST_column):
751         /* FALLTHROUGH */
752     case (LIST_diag):
753         /* FALLTHROUGH */
754     case (LIST_ohang):
755         /* FALLTHROUGH */
756     case (LIST_inset):
757         /* FALLTHROUGH */
758     case (LIST_item):
759         if (n->norm->Bl.width)
760             mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
761         break;
762     case (LIST_bullet):
763         /* FALLTHROUGH */
764     case (LIST_dash):
765         /* FALLTHROUGH */
766     case (LIST_hyphen):
767         if (NULL == n->norm->Bl.width)
768             n->norm->Bl.width = "2n";
769         break;
770     case (LIST_enum):
771         if (NULL == n->norm->Bl.width)
772             n->norm->Bl.width = "3n";
773         break;
774     default:
775         break;
776 }

777     return(1);
778 }

unchanged_portion_omitted_

886 static int
887 pre_sh(PRE_ARGS)
888 {
889     if (MDOC_BLOCK != n->type)
890         return(1);

892     roff_regunset(mdoc->roff, REG_NS);
893     return(check_parent(mdoc, n, MDOC_MAX, MDOC_ROOT));
894 }

unchanged_portion_omitted_

1121 static int
1122 post_nm(POST_ARGS)
1123 {
1124     char        buf[BUFSIZ];
1125     int         c;

1127     if (NULL != mdoc->meta.name)
1128         /* If no child specified, make sure we have the meta name. */

1129     if (NULL == mdoc->last->child && NULL == mdoc->meta.name) {

```

```

1117         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NONAME);
1118         return(1);
1119     } else if (mdoc->meta.name)
1120         return(1);

1130     /* Try to use our children for setting the meta name. */
1122     /* If no meta name, set it from the child. */

1132     if (NULL != mdoc->last->child) {
1133         buf[0] = '\0';
1134         c = concat(buf, mdoc->last->child, BUFSIZ);
1135     } else
1136         c = 0;

1138     switch (c) {
1139     case (-1):
1125     if (-1 == (c = concat(buf, mdoc->last->child, BUFSIZ))) {
1140         mdoc_nmsg(mdoc, mdoc->last->child, MANDOCERR_MEM);
1141         return(0);
1142     case (0):
1143         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NONAME);
1144         mdoc->meta.name = mandoc_strdup("UNKNOWN");
1145         break;
1146     default:
1147         mdoc->meta.name = mandoc_strdup(buf);
1148         break;
1149     }

1130     assert(c);
1131     mdoc->meta.name = mandoc_strdup(buf);
1150     return(1);
1151 }
    _____
    unchanged_portion_omitted_

1352 static int
1353 post_bl_block(POST_ARGS)
1354 {
1355     struct mdoc_node *n, *ni, *nc;
1356     struct mdoc_node *n;

1357     /*
1358     * These are fairly complicated, so we've broken them into two
1359     * functions.  post_bl_block_tag() is called when a -tag is
1360     * specified, but no -width (it must be guessed).  The second
1361     * when a -width is specified (macro indicators must be
1362     * rewritten into real lengths).
1363     */

1365     n = mdoc->last;

1367     if (LIST_tag == n->norm->Bl.type &&
1368         NULL == n->norm->Bl.width) {
1369         if ( ! post_bl_block_tag(mdoc))
1370             return(0);
1371         assert(n->norm->Bl.width);
1372     } else if (NULL != n->norm->Bl.width) {
1373         if ( ! post_bl_block_width(mdoc))
1374             return(0);
1375         assert(n->norm->Bl.width);
1376     }

1378     for (ni = n->body->child; ni; ni = ni->next) {
1379         if (NULL == ni->body)
1380             continue;
1381         nc = ni->body->last;
1382         while (NULL != nc) {

```

```

1383         switch (nc->tok) {
1384         case (MDOC_Pp):
1385             /* FALLTHROUGH */
1386         case (MDOC_Lp):
1387             /* FALLTHROUGH */
1388         case (MDOC_br):
1389             break;
1390         default:
1391             nc = NULL;
1392             continue;
1393         }
1394         if (NULL == ni->next) {
1395             mdoc_nmsg(mdoc, nc, MANDOCERR_MOVEPAR);
1396             if ( ! mdoc_node_relink(mdoc, nc))
1397                 return(0);
1398         } else if (0 == n->norm->Bl.comp &&
1399                 LIST_column != n->norm->Bl.type) {
1400             mdoc_nmsg(mdoc, nc, MANDOCERR_IGNPAR);
1401             mdoc_node_delete(mdoc, nc);
1402         } else
1403             break;
1404         nc = ni->body->last;
1405     }
1406 }
1407     return(1);

1359     assert(n->norm->Bl.width);
1360     return(1);
1408 }
    _____
    unchanged_portion_omitted_

1591 static int
1592 post_bl(POST_ARGS)
1593 {
1594     struct mdoc_node *nparent, *nprev; /* of the Bl block */
1595     struct mdoc_node *nblock, *nbody; /* of the Bl */
1596     struct mdoc_node *nchild, *nnext; /* of the Bl body */
1597     struct mdoc_node *n;

1598     nbody = mdoc->last;
1599     switch (nbody->type) {
1600     case (MDOC_BLOCK):
1601         return(post_bl_block(mdoc));
1602     case (MDOC_HEAD):
1603         if (MDOC_HEAD == mdoc->last->type)
1604             return(post_bl_head(mdoc));
1605     case (MDOC_BODY):
1606         break;
1607     default:
1608         if (MDOC_BLOCK == mdoc->last->type)
1609             return(post_bl_block(mdoc));
1610         if (MDOC_BODY != mdoc->last->type)
1611             return(1);
1612     }

1610     nchild = nbody->child;
1611     while (NULL != nchild) {
1612         if (MDOC_It == nchild->tok || MDOC_Sm == nchild->tok) {
1613             nchild = nchild->next;
1614         } else {
1615             for (n = mdoc->last->child; n; n = n->next) {
1616                 switch (n->tok) {
1617                 case (MDOC_Lp):
1618                     /* FALLTHROUGH */
1619                 case (MDOC_Pp):
1620                     mdoc_nmsg(mdoc, n, MANDOCERR_CHILD);
1621                     /* FALLTHROUGH */

```



```

1563     case (MDOC_It):
1564         /* FALLTHROUGH */
1565     case (MDOC_Sm):
1566         continue;
1567     default:
1568         break;
1569 }
1570
1571 mdoc_nmsg(mdoc, nchild, MANDOCERR_CHILD);
1572
1573 /*
1574  * Move the node out of the B1 block.
1575  * First, collect all required node pointers.
1576  */
1577
1578 nblock = nbody->parent;
1579 nprev  = nblock->prev;
1580 nparent = nblock->parent;
1581 nnext  = nchild->next;
1582
1583 /*
1584  * Unlink this child.
1585  */
1586
1587 assert(NULL == nchild->prev);
1588 if (0 == --nbody->nchild) {
1589     nbody->child = NULL;
1590     nbody->last = NULL;
1591     assert(NULL == nnext);
1592 } else {
1593     nbody->child = nnext;
1594     nnext->prev = NULL;
1595 }
1596 mdoc_nmsg(mdoc, n, MANDOCERR_SYNTCHILD);
1597 return(0);
1598 }
1599
1600 /*
1601  * Relink this child.
1602  */
1603
1604 nchild->parent = nparent;
1605 nchild->prev  = nprev;
1606 nchild->next  = nblock;
1607
1608 nblock->prev = nchild;
1609 nparent->nchild++;
1610 if (NULL == nprev)
1611     nparent->child = nchild;
1612 else
1613     nprev->next = nchild;
1614
1615 nchild = nnext;
1616 }
1617
1618 return(1);
1619 }
1620
1621 static int
1622 ebool(struct mdoc *mdoc)
1623 {
1624     if (NULL == mdoc->last->child) {
1625         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_MACROEMPTY);
1626         mdoc_node_delete(mdoc, mdoc->last);
1627         return(1);
1628     }

```

```

1629     check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 1);
1630
1631     assert(MDOC_TEXT == mdoc->last->child->type);
1632
1633     if (0 == strcmp(mdoc->last->child->string, "on")) {
1634         if (MDOC_Sm == mdoc->last->tok)
1635             mdoc->flags &= ~MDOC_SMOFF;
1636         if (0 == strcmp(mdoc->last->child->string, "on"))
1637             return(1);
1638     }
1639     if (0 == strcmp(mdoc->last->child->string, "off")) {
1640         if (MDOC_Sm == mdoc->last->tok)
1641             mdoc->flags |= MDOC_SMOFF;
1642         if (0 == strcmp(mdoc->last->child->string, "off"))
1643             return(1);
1644     }
1645
1646     mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADBOOL);
1647     return(1);
1648 }
1649
1650 _____unchanged_portion_omitted_____
1651
1652 /*
1653  * For some arguments of some macros,
1654  * convert all breakable hyphens into ASCII_HYPH.
1655  */
1656
1657 static int
1658 post_hyph(POST_ARGS)
1659 {
1660     struct mdoc_node *n, *nch;
1661     char *cp;
1662
1663     n = mdoc->last;
1664     switch (n->type) {
1665     case (MDOC_HEAD):
1666         if (MDOC_Sh == n->tok || MDOC_Ss == n->tok)
1667             break;
1668         return(1);
1669     case (MDOC_BODY):
1670         if (MDOC_Dl == n->tok || MDOC_Nd == n->tok)
1671             break;
1672         return(1);
1673     case (MDOC_ELEM):
1674         break;
1675     default:
1676         return(1);
1677     }
1678
1679     for (nch = n->child; nch; nch = nch->next) {
1680         if (MDOC_TEXT != nch->type)
1681             continue;
1682         cp = nch->string;
1683         if (3 > strlen(cp, 3))
1684             continue;
1685         while ('\0' != *(++cp))
1686             if ('-' == *cp &&
1687                 isalpha((unsigned char)cp[-1]) &&
1688                 isalpha((unsigned char)cp[1]))
1689                 *cp = ASCII_HYPH;
1690     }
1691     return(1);
1692 }
1693
1694 static int
1695 post_ns(POST_ARGS)
1696 {

```

```

1911     if (MDOC_LINE & mdoc->last->flags)
1912         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_IGNNS);
1913     return(1);
1914 }
_____ unchanged portion omitted _____

1963 static int
1964 post_sh_head(POST_ARGS)
1965 {
1966     char          buf[BUFSIZ];
1967     struct mdoc_node *n;
1968     enum mdoc_sec  sec;
1969     int           c;

1971     /*
1972     * Process a new section.  Sections are either "named" or
1973     * "custom".  Custom sections are user-defined, while named ones
1974     * follow a conventional order and may only appear in certain
1975     * manual sections.
1976     */

1978     sec = SEC_CUSTOM;
1979     buf[0] = '\0';
1980     if (-1 == (c = concat(buf, mdoc->last->child, BUFSIZ))) {
1981         mdoc_nmsg(mdoc, mdoc->last->child, MANDOCERR_MEM);
1982         return(0);
1983     } else if (1 == c)
1984         sec = a2sec(buf);

1986     /* The NAME should be first. */

1988     if (SEC_NAME != sec && SEC_NONE == mdoc->lastnamed)
1989         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NAMESECFIRST);

1991     /* The SYNOPSIS gets special attention in other areas. */

1993     if (SEC_SYNOPSIS == sec) {
1994         roff_setreg(mdoc->roff, "nS", 1, '=');
1860     if (SEC_SYNOPSIS == sec)
1995         mdoc->flags |= MDOC_SYNOPSIS;
1996     } else {
1997         roff_setreg(mdoc->roff, "nS", 0, '=');
1862     else
1998         mdoc->flags &= ~MDOC_SYNOPSIS;
1999     }

2001     /* Mark our last section. */

2003     mdoc->lastsec = sec;

2005     /*
2006     * Set the section attribute for the current HEAD, for its
2007     * parent BLOCK, and for the HEAD children; the latter can
2008     * only be TEXT nodes, so no recursion is needed.
2009     * For other blocks and elements, including .Sh BODY, this is
2010     * done when allocating the node data structures, but for .Sh
2011     * BLOCK and HEAD, the section is still unknown at that time.
2012     */

2014     mdoc->last->parent->sec = sec;
2015     mdoc->last->sec = sec;
2016     for (n = mdoc->last->child; n; n = n->next)
2017         n->sec = sec;

2019     /* We don't care about custom sections after this. */

```

```

2021     if (SEC_CUSTOM == sec)
2022         return(1);

2024     /*
2025     * Check whether our non-custom section is being repeated or is
2026     * out of order.
2027     */

2029     if (sec == mdoc->lastnamed)
2030         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECREP);

2032     if (sec < mdoc->lastnamed)
2033         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECOOO);

2035     /* Mark the last named section. */

2037     mdoc->lastnamed = sec;

2039     /* Check particular section/manual conventions. */

2041     assert(mdoc->meta.msec);

2043     switch (sec) {
2044     case (SEC_RETURN_VALUES):
2045         /* FALLTHROUGH */
2046     case (SEC_ERRORS):
2047         /* FALLTHROUGH */
2048     case (SEC_LIBRARY):
2049         if (*mdoc->meta.msec == '2')
2050             break;
2051         if (*mdoc->meta.msec == '3')
2052             break;
2053         if (*mdoc->meta.msec == '9')
2054             break;
2055         mdoc_nmsg(MANDOCERR_SECMSEC, mdoc->parse,
2056                 mdoc->last->line, mdoc->last->pos, buf);
1919         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECMSEC);
2057         break;
2058     default:
2059         break;
2060     }

2062     return(1);
2063 }
_____ unchanged portion omitted _____

2088 static int
2089 pre_par(PRE_ARGS)
2090 {

2092     if (NULL == mdoc->last)
2093         return(1);
2094     if (MDOC_ELEM != n->type && MDOC_BLOCK != n->type)
2095         return(1);

2097     /*
2098     * Don't allow prior 'Lp' or 'Pp' prior to a paragraph-type
2099     * block: 'Lp', 'Pp', or non-compact 'Bd' or 'Bl'.
2100     */

2102     if (MDOC_Pp != mdoc->last->tok &&
2103         MDOC_Lp != mdoc->last->tok &&
2104         MDOC_br != mdoc->last->tok)
1965     if (MDOC_Pp != mdoc->last->tok && MDOC_Lp != mdoc->last->tok)
2105         return(1);

```

```

2106     if (MDOC_Bl == n->tok && n->norm->Bl.comp)
2107         return(1);
2108     if (MDOC_Bd == n->tok && n->norm->Bd.comp)
2109         return(1);
2110     if (MDOC_It == n->tok && n->parent->norm->Bl.comp)
2111         return(1);

2113     mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_IGNPAR);
2114     mdoc_node_delete(mdoc, mdoc->last);
2115     return(1);
2116 }

2118 static int
2119 post_par(POST_ARGS)
2120 {

2122     if (MDOC_ELEM != mdoc->last->type &&
2123         MDOC_BLOCK != mdoc->last->type)
2124         return(1);

2126     if (NULL == mdoc->last->prev) {
2127         if (MDOC_Sh != mdoc->last->parent->tok &&
2128             MDOC_Ss != mdoc->last->parent->tok)
2129             return(1);
2130     } else {
2131         if (MDOC_Pp != mdoc->last->prev->tok &&
2132             MDOC_Lp != mdoc->last->prev->tok &&
2133             (MDOC_br != mdoc->last->tok ||
2134              (MDOC_sp != mdoc->last->prev->tok &&
2135               MDOC_br != mdoc->last->prev->tok)))
2136             return(1);
2137     }

2139     mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_IGNPAR);
2140     mdoc_node_delete(mdoc, mdoc->last);
2141     return(1);
2142 }

2144 static int
2145 pre_literal(PRE_ARGS)
2146 {

2148     if (MDOC_BODY != n->type)
2149         return(1);

2151     /*
2152      * The 'Dl' (note "el" not "one") and 'Bd -literal' and 'Bd
2153      * -unfilled' macros set MDOC_LITERAL on entrance to the body.
2154      */

2156     switch (n->tok) {
2157     case (MDOC_Dl):
2158         mdoc->flags |= MDOC_LITERAL;
2159         break;
2160     case (MDOC_Bd):
2161         if (DISP_literal == n->norm->Bd.type)
2162             mdoc->flags |= MDOC_LITERAL;
2163         if (DISP_unfilled == n->norm->Bd.type)
2164             mdoc->flags |= MDOC_LITERAL;
2165         break;
2166     default:
2167         abort();
2168         /* NOTREACHED */
2169     }

2170     return(1);
2171 }

```

```

2172 }
      unchanged_portion_omitted

2204 static int
2205 post_dt(POST_ARGS)
2206 {
2207     struct mdoc_node *nn, *n;
2208     const char *cp;
2209     char *p;

2211     n = mdoc->last;

2213     if (mdoc->meta.title)
2214         free(mdoc->meta.title);
2215     if (mdoc->meta.vol)
2216         free(mdoc->meta.vol);
2217     if (mdoc->meta.arch)
2218         free(mdoc->meta.arch);

2220     mdoc->meta.title = mdoc->meta.vol = mdoc->meta.arch = NULL;

2222     /* First make all characters uppercase. */

2224     if (NULL != (nn = n->child))
2225         for (p = nn->string; *p; p++) {
2226             if (toupper((unsigned char)*p) == *p)
2227                 continue;

2229             /*
2230              * FIXME: don't be lazy: have this make all
2231              * characters be uppercase and just warn once.
2232              */
2233             mdoc_nmsg(mdoc, nn, MANDOCERR_UPPERCASE);
2234             break;
2235         }

2237     /* Handles: '.Dt'
2238      * --> title = unknown, volume = local, msec = 0, arch = NULL
2239      */

2241     if (NULL == (nn = n->child)) {
2242         /* XXX: make these macro values. */
2243         /* FIXME: warn about missing values. */
2244         mdoc->meta.title = mandoc_strdup("UNKNOWN");
2245         mdoc->meta.vol = mandoc_strdup("LOCAL");
2246         mdoc->meta.msec = mandoc_strdup("1");
2247         return(1);
2248     }

2250     /* Handles: '.Dt TITLE'
2251      * --> title = TITLE, volume = local, msec = 0, arch = NULL
2252      */

2254     mdoc->meta.title = mandoc_strdup
2255         ('\0' == nn->string[0] ? "UNKNOWN" : nn->string);

2257     if (NULL == (nn = nn->next)) {
2258         /* FIXME: warn about missing msec. */
2259         /* XXX: make this a macro value. */
2260         mdoc->meta.vol = mandoc_strdup("LOCAL");
2261         mdoc->meta.msec = mandoc_strdup("1");
2262         return(1);
2263     }

2265     /* Handles: '.Dt TITLE SEC'
2266      * --> title = TITLE, volume = SEC is msec ?

```

```

2267      *          format(msec) : SEC,
2268      *          msec = SEC is msec ? atoi(msec) : 0,
2269      *          arch = NULL
2270      */

2272 cp = mdoc_a2msec(nn->string);
2273 if (cp) {
2274     mdoc->meta.vol = mdoc_strdup(cp);
2275     mdoc->meta.msec = mdoc_strdup(nn->string);
2276 } else {
2277     mdoc_nmsg(mdoc, n, MANDOCERR_BADMSEC);
2278     mdoc->meta.vol = mdoc_strdup(nn->string);
2279     mdoc->meta.msec = mdoc_strdup(nn->string);
2280 }

2282 if (NULL == (nn = nn->next))
2283     return(1);

2285 /* Handles: '.Dt TITLE SEC VOL'
2286  * --> title = TITLE, volume = VOL is vol ?
2287  *          format(VOL) :
2288  *          VOL is arch ? format(arch) :
2289  *          VOL
2290  */

2292 cp = mdoc_a2vol(nn->string);
2293 if (cp) {
2294     free(mdoc->meta.vol);
2295     mdoc->meta.vol = mdoc_strdup(cp);
2296 } else {
2132     /* FIXME: warn about bad arch. */
2297     cp = mdoc_a2arch(nn->string);
2298     if (NULL == cp) {
2299         mdoc_nmsg(mdoc, nn, MANDOCERR_BADVOLARCH);
2300         free(mdoc->meta.vol);
2301         mdoc->meta.vol = mdoc_strdup(nn->string);
2302     } else
2303         mdoc->meta.arch = mdoc_strdup(cp);
2304 }

2306 /* Ignore any subsequent parameters... */
2307 /* FIXME: warn about subsequent parameters. */

2309     return(1);
2310 }
_____ unchanged portion omitted _____

2347 static int
2348 post_os(POST_ARGS)
2349 {
2350     struct mdoc_node *n;
2351     char      buf[BUFSIZ];
2352     int      c;
2353 #ifndef OSNAME
2354     struct utsname  utsname;
2355 #endif

2357     n = mdoc->last;

2359     /*
2360     * Set the operating system by way of the 'Os' macro.
2361     * The order of precedence is:
2362     * 1. the argument of the 'Os' macro, unless empty
2363     * 2. the -Ios=foo command line argument, if provided
2364     * 3. -DOSNAME="\foo\" is provided during compilation
2365     * 4. "sysname release" from uname(3)

```

```

2195     * Set the operating system by way of the 'Os' macro. Note that
2196     * if an argument isn't provided and -DOSNAME="\foo\" is
2197     * provided during compilation, this value will be used instead
2198     * of filling in "sysname release" from uname().
2366     */

2201     if (mdoc->meta.os)
2368         free(mdoc->meta.os);

2370     buf[0] = '\0';
2371     if (-1 == (c = concat(buf, n->child, BUFSIZ))) {
2372         mdoc_nmsg(mdoc, n->child, MANDOCERR_MEM);
2373         return(0);
2374     }

2376     assert(c);

2212     /* XXX: yes, these can all be dynamically-adjusted buffers, but
2213     * it's really not worth the extra hackery.
2214     */

2378     if ('\0' == buf[0]) {
2379         if (mdoc->defos) {
2380             mdoc->meta.os = mdoc_strdup(mdoc->defos);
2381             return(1);
2382         }
2383 #ifdef OSNAME
2384         if (strlcat(buf, OSNAME, BUFSIZ) >= BUFSIZ) {
2385             mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2386             return(0);
2387         }
2388 #else /*!OSNAME */
2389         if (-1 == uname(&utsname)) {
2390             mdoc_nmsg(mdoc, n, MANDOCERR_UNAME);
2391             mdoc->meta.os = mdoc_strdup("UNKNOWN");
2392             return(post_prol(mdoc));
2393         }
2395         if (strlcat(buf, utsname.sysname, BUFSIZ) >= BUFSIZ) {
2396             mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2397             return(0);
2398         }
2399         if (strlcat(buf, " ", BUFSIZ) >= BUFSIZ) {
2400             mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2401             return(0);
2402         }
2403         if (strlcat(buf, utsname.release, BUFSIZ) >= BUFSIZ) {
2404             mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2405             return(0);
2406         }
2407 #endif /*!OSNAME*/
2408     }

2410     mdoc->meta.os = mdoc_strdup(buf);
2411     return(1);
2412 }
_____ unchanged portion omitted _____

```

```

*****
5106 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/msec.in
local changes
Update to 1.12.3.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
14  */

16 LINE("1", "User Commands")
17 LINE("1B", "BSD Compatibility Package Commands")
18 LINE("1b", "illumos/BSD Compatibility Package Commands")
19 LINE("1c", "Communication Commands")
20 LINE("1c", "Communication Commands")
21 LINE("1f", "FMLI Commands")
22 LINE("1f", "FMLI Commands")
23 LINE("1g", "Graphics and CAD Commands")
24 LINE("1g", "Graphics and CAD Commands")
25 LINE("1has", "User Commands")
26 LINE("1has", "User Commands")
27 LINE("1m", "Maintenance Commands")
28 LINE("1m", "Maintenance Commands")
29 LINE("1s", "illumos Specific Commands")
30 LINE("1s", "illumos Specific Commands")
31 LINE("2", "System Calls")
32 LINE("3", "Introduction to Library Functions")
24 LINE("3BSDMALLOC", "BSD Memory Allocation Library")
33 LINE("3AIO", "Asynchronous I/O Library Functions")
34 LINE("3aio", "Asynchronous I/O Library Functions")
25 LINE("3BSM", "Security and Auditing Library Functions")
36 LINE("3bsm", "Security and Auditing Library Functions")
26 LINE("3C", "Standard C Library Functions")
27 LINE("3C+", "C++ Library Functions")
38 LINE("3c", "Standard C Library Functions")
28 LINE("3C_DB", "Threads Debugging Library Functions")
40 LINE("3c_db", "Threads Debugging Library Functions")
29 LINE("3CFGADM", "Configuration Administration Library Functions")
30 LINE("3COMMUTIL", "Communication Protocol Parser Utilities Library Functions")
42 LINE("3cfgadm", "Configuration Administration Library Functions")
43 LINE("3COMPUTIL", "Communication Protocol Parser Utilities Library Functions")
44 LINE("3computil", "Communication Protocol Parser Utilities Library Functions")
31 LINE("3CONTRACT", "Contract Management Library Functions")
46 LINE("3contract", "Contract Management Library Functions")
32 LINE("3CPC", "CPU Performance Counters Library Functions")
48 LINE("3cpc", "CPU Performance Counters Library Functions")
33 LINE("3CURSES", "Curses Library Functions")
50 LINE("3curses", "Curses Library Functions")
34 LINE("3DAT", "Direct Access Transport Library Functions")
52 LINE("3dat", "Direct Access Transport Library Functions")
35 LINE("3DEVID", "Device ID Library Functions")
54 LINE("3devid", "Device ID Library Functions")
36 LINE("3DEVINFO", "Device Information Library Functions")
56 LINE("3devinfo", "Device Information Library Functions")

```

```

57 LINE("3DL", "Dynamic Linking Library Functions")
58 LINE("3dl", "Dynamic Linking Library Functions")
37 LINE("3DLPI", "Data Link Provider Interface Library Functions")
60 LINE("3dlpi", "Data Link Provider Interface Library Functions")
61 LINE("3DMI", "DMI Library Functions")
62 LINE("3dmi", "DMI Library Functions")
38 LINE("3DNS_SD", "DNS Service Discovery Library Functions")
64 LINE("3dns_sd", "DNS Service Discovery Library Functions")
39 LINE("3DOOR", "Door Library Functions")
66 LINE("3door", "Door Library Functions")
40 LINE("3ELF", "ELF Library Functions")
68 LINE("3elf", "ELF Library Functions")
41 LINE("3EXACCT", "Extended Accounting File Access Library Functions")
70 LINE("3exacct", "Extended Accounting File Access Library Functions")
42 LINE("3EXT", "Extended Library Functions")
72 LINE("3ext", "Extended Library Functions")
43 LINE("3FCOE", "FCoE Port Management Library Functions")
74 LINE("3fcoe", "FCoE Port Management Library Functions")
44 LINE("3FSTYP", "File System Type Identification Library Functions")
76 LINE("3fstyp", "File System Type Identification Library Functions")
45 LINE("3GEN", "String Pattern-Matching Library Functions")
78 LINE("3gen", "String Pattern-Matching Library Functions")
46 LINE("3GSS", "Generic Security Services API Library Functions")
80 LINE("3gss", "Generic Security Services API Library Functions")
47 LINE("3HEAD", "Headers")
82 LINE("3head", "Headers")
48 LINE("3ISCSIT", "iSCSI Management Library Functions")
84 LINE("3iscsit", "iSCSI Management Library Functions")
49 LINE("3KRB", "Kerberos Library Functions")
50 LINE("3KRB5", "MIT Kerberos 5 Library Functions")
86 LINE("3krb", "Kerberos Library Functions")
51 LINE("3KSTAT", "Kernel Statistics Library Functions")
88 LINE("3kstat", "Kernel Statistics Library Functions")
52 LINE("3KVM", "Kernel VM Library Functions")
90 LINE("3kvm", "Kernel VM Library Functions")
53 LINE("3LDAP", "LDAP Library Functions")
92 LINE("3ldap", "LDAP Library Functions")
54 LINE("3LGRP", "Locality Group Library Functions")
94 LINE("3lgrp", "Locality Group Library Functions")
55 LINE("3LIB", "Interface Libraries")
96 LINE("3lib", "Interface Libraries")
97 LINE("3LIBUCB", "illumos/BSD Compatibility Interface Libraries")
98 LINE("3libucb", "illumos/BSD Compatibility Interface Libraries")
56 LINE("3M", "Mathematical Library Functions")
100 LINE("3m", "Mathematical Library Functions")
57 LINE("3MAIL", "User Mailbox Library Functions")
102 LINE("3mail", "User Mailbox Library Functions")
58 LINE("3MALLOC", "Memory Allocation Library Functions")
104 LINE("3malloc", "Memory Allocation Library Functions")
59 LINE("3MP", "Multiple Precision Library Functions")
106 LINE("3mp", "Multiple Precision Library Functions")
60 LINE("3MPAPI", "Common Multipath Management Library Functions")
108 LINE("3mpapi", "Common Multipath Management Library Functions")
61 LINE("3NSL", "Networking Services Library Functions")
110 LINE("3nsl", "Networking Services Library Functions")
62 LINE("3NVPAR", "Name-value Pair Library Functions")
112 LINE("3nvpair", "Name-value Pair Library Functions")
63 LINE("3PAM", "PAM Library Functions")
114 LINE("3pam", "PAM Library Functions")
64 LINE("3PAPI", "PAPI Library Functions")
116 LINE("3papi", "PAPI Library Functions")
65 LINE("3PERL", "Perl Library Functions")
118 LINE("3perl", "Perl Library Functions")
66 LINE("3PICL", "PICL Library Functions")
120 LINE("3picl", "PICL Library Functions")
67 LINE("3PICLTREE", "PICL Plug-In Library Functions")

```

```

122 LINE("3picltree", "PICL Plug-In Library Functions")
123 LINE("3PLOT", "Graphics Interface Library Functions")
124 LINE("3plot", "Graphics Interface Library Functions")
68 LINE("3POOL", "Pool Configuration Manipulation Library Functions")
126 LINE("3pool", "Pool Configuration Manipulation Library Functions")
69 LINE("3PROC", "Process Control Library Functions")
128 LINE("3proc", "Process Control Library Functions")
70 LINE("3PROJECT", "Project Database Access Library Functions")
130 LINE("3project", "Project Database Access Library Functions")
71 LINE("3RAC", "Remote Asynchronous Calls Library Functions")
132 LINE("3rac", "Remote Asynchronous Calls Library Functions")
72 LINE("3RESOLV", "Resolver Library Functions")
134 LINE("3resolv", "Resolver Library Functions")
73 LINE("3RPC", "RPC Library Functions")
136 LINE("3rpc", "RPC Library Functions")
74 LINE("3RSM", "Remote Shared Memory Library Functions")
138 LINE("3rsm", "Remote Shared Memory Library Functions")
75 LINE("3RT", "Realtime Library Functions")
140 LINE("3rt", "Realtime Library Functions")
76 LINE("3SASL", "Simple Authentication Security Layer Library Functions")
142 LINE("3sas1", "Simple Authentication Security Layer Library Functions")
77 LINE("3SCF", "Service Configuration Facility Library Functions")
144 LINE("3scf", "Service Configuration Facility Library Functions")
145 LINE("3SCHED", "LWP Scheduling Library Functions")
146 LINE("3sched", "LWP Scheduling Library Functions")
78 LINE("3SEC", "File Access Control Library Functions")
148 LINE("3sec", "File Access Control Library Functions")
79 LINE("3SECDB", "Security Attributes Database Library Functions")
150 LINE("3secdb", "Security Attributes Database Library Functions")
80 LINE("3SIP", "Session Initiation Protocol Library Functions")
152 LINE("3sip", "Session Initiation Protocol Library Functions")
81 LINE("3SLP", "Service Location Protocol Library Functions")
154 LINE("3slp", "Service Location Protocol Library Functions")
155 LINE("3SNMP", "SNMP Library Functions")
156 LINE("3snmp", "SNMP Library Functions")
82 LINE("3SOCKET", "Sockets Library Functions")
158 LINE("3socket", "Sockets Library Functions")
83 LINE("3STMF", "SCSI Target Mode Framework Library Functions")
160 LINE("3stmf", "SCSI Target Mode Framework Library Functions")
84 LINE("3SYSEVENT", "System Event Library Functions")
162 LINE("3ysevent", "System Event Library Functions")
85 LINE("3TECLA", "Interactive Command-line Input Library Functions")
164 LINE("3tecla", "Interactive Command-line Input Library Functions")
165 LINE("3THR", "Threads Library Functions")
166 LINE("3thr", "Threads Library Functions")
86 LINE("3TNF", "TNF Library Functions")
168 LINE("3tnf", "TNF Library Functions")
87 LINE("3TSOL", "Trusted Extensions Library Functions")
170 LINE("3tsol", "Trusted Extensions Library Functions")
171 LINE("3UCB", "illumos/BSD Compatibility Library Functions")
172 LINE("3ucb", "illumos/BSD Compatibility Library Functions")
88 LINE("3UUID", "Universally Unique Identifier Library Functions")
174 LINE("3uuid", "Universally Unique Identifier Library Functions")
89 LINE("3VOLMGT", "Volume Management Library Functions")
176 LINE("3volmgt", "Volume Management Library Functions")
90 LINE("3XCURSES", "X/Open Curses Library Functions")
178 LINE("3xcurses", "X/Open Curses Library Functions")
179 LINE("3XFN", "XFN Interface Library Functions")
180 LINE("3xfn", "XFN Interface Library Functions")
91 LINE("3XNET", "X/Open Networking Services Library Functions")
182 LINE("3xnet", "X/Open Networking Services Library Functions")
183 LINE("3B", "illumos/BSD Compatibility Library Functions")
184 LINE("3b", "illumos/BSD Compatibility Library Functions")
185 LINE("3E", "C Library Functions")
186 LINE("3e", "C Library Functions")
92 LINE("3F", "Fortran Library Routines")

```

```

188 LINE("3f", "Fortran Library Routines")
189 LINE("3G", "C Library Functions")
190 LINE("3g", "C Library Functions")
191 LINE("3K", "Kernel VM Library Functions")
192 LINE("3k", "Kernel VM Library Functions")
193 LINE("3L", "Lightweight Processes Library")
194 LINE("3l", "Lightweight Processes Library")
195 LINE("3N", "Network Functions")
196 LINE("3n", "Network Functions")
197 LINE("3R", "Realtime Library")
198 LINE("3r", "Realtime Library")
199 LINE("3S", "Standard I/O Functions")
200 LINE("3s", "Standard I/O Functions")
201 LINE("3T", "Threads Library")
202 LINE("3t", "Threads Library")
203 LINE("3W", "C Library Functions")
204 LINE("3w", "C Library Functions")
93 LINE("3X", "Miscellaneous Library Functions")
206 LINE("3x", "Miscellaneous Library Functions")
207 LINE("3XC", "X/Open Curses Library Functions")
208 LINE("3xc", "X/Open Curses Library Functions")
209 LINE("3XN", "X/Open Networking Services Library Functions")
210 LINE("3xn", "X/Open Networking Services Library Functions")
94 LINE("4", "File Formats")
212 LINE("4B", "illumos/BSD Compatibility Package File Formats")
213 LINE("4b", "illumos/BSD Compatibility Package File Formats")
95 LINE("5", "Standards, Environments, and Macros")
96 LINE("6", "Games and Demos")
97 LINE("7", "Device and Network Interfaces")
217 LINE("7B", "illumos/BSD Compatibility Special Files")
218 LINE("7b", "illumos/BSD Compatibility Special Files")
98 LINE("7D", "Devices")
220 LINE("7d", "Devices")
99 LINE("7FS", "File Systems")
222 LINE("7fs", "File Systems")
100 LINE("7I", "Ioctl Requests")
224 LINE("7i", "Ioctl Requests")
101 LINE("7IPP", "IP Quality of Service Modules")
226 LINE("7ipp", "IP Quality of Service Modules")
102 LINE("7M", "STREAMS Modules")
228 LINE("7m", "STREAMS Modules")
103 LINE("7P", "Protocols")
230 LINE("7p", "Protocols")
104 LINE("8", "Maintenance Procedures")
232 LINE("8C", "Maintenance Procedures")
233 LINE("8c", "Maintenance Procedures")
234 LINE("8S", "Maintenance Procedures")
235 LINE("8s", "Maintenance Procedures")
105 LINE("9", "Device Driver Interfaces")
106 LINE("9E", "Driver Entry Points")
238 LINE("9e", "Driver Entry Points")
107 LINE("9F", "Kernel Functions for Drivers")
240 LINE("9f", "Kernel Functions for Drivers")
108 LINE("9P", "Kernel Properties for Drivers")
242 LINE("9p", "Kernel Properties for Drivers")
109 LINE("9S", "Data Structures for Drivers")
244 LINE("9s", "Data Structures for Drivers")

```

```

*****
6210 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/out.c
Update to 1.12.3.
*****
1 /* $Id: out.c,v 1.46 2013/10/05 20:30:05 schwarze Exp $ */
1 /* $Id: out.c,v 1.43 2011/09/20 23:05:49 schwarze Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29 #include <time.h>
30
31 #include "mandoc.h"
32 #include "out.h"
33
34 static void tblcalc_data(struct rofftbl *, struct roffcol *,
35                         const struct tbl_opts *, const struct tbl_dat *);
36 static void tblcalc_literal(struct rofftbl *, struct roffcol *,
37                             const struct tbl_dat *);
38 static void tblcalc_number(struct rofftbl *, struct roffcol *,
39                             const struct tbl_opts *, const struct tbl_dat *);
40
41 /*
42  * Convert a 'scaling unit' to a consistent form, or fail.  Scaling
43  * units are documented in groff.7, mdoc.7, man.7.
44  */
45 int
46 a2roffsu(const char *src, struct roffsu *dst, enum roffscale def)
47 {
48     char buf[BUFSIZ], hasd;
49     int i;
50     enum roffscale unit;
51
52     if ('\0' == *src)
53         return(0);
54
55     i = hasd = 0;
56
57     switch (*src) {
58     case ('+'):

```

```

59         src++;
60         break;
61     case ('-'):
62         buf[i++] = *src++;
63         break;
64     default:
65         break;
66     }
67
68     if ('\0' == *src)
69         return(0);
70
71     while (i < BUFSIZ) {
72         if (! isdigit((unsigned char)*src)) {
73             if ('.' != *src)
74                 break;
75             else if (hasd)
76                 break;
77             else
78                 hasd = 1;
79         }
80         buf[i++] = *src++;
81     }
82
83     if (BUFSIZ == i || (*src && *(src + 1)))
84         return(0);
85
86     buf[i] = '\0';
87
88     switch (*src) {
89     case ('c'):
90         unit = SCALE_CM;
91         break;
92     case ('i'):
93         unit = SCALE_IN;
94         break;
95     case ('p'):
96         unit = SCALE_PC;
97         break;
98     case ('p'):
99         unit = SCALE_PT;
100        break;
101     case ('f'):
102         unit = SCALE_FS;
103         break;
104     case ('v'):
105         unit = SCALE_VS;
106         break;
107     case ('m'):
108         unit = SCALE_EM;
109         break;
110     case ('\0'):
111         if (SCALE_MAX == def)
112             return(0);
113         unit = SCALE_BU;
114         break;
115     case ('u'):
116         unit = SCALE_BU;
117         break;
118     case ('M'):
119         unit = SCALE_MM;
120         break;
121     case ('n'):
122         unit = SCALE_EN;
123         break;
124     default:

```

```

125         return(0);
126     }

128     /* FIXME: do this in the caller. */
129     if ((dst->scale = atof(buf)) < 0)
130         dst->scale = 0;
131     dst->unit = unit;
132     return(1);
133 }

135 /*
136 * Calculate the abstract widths and decimal positions of columns in a
137 * table. This routine allocates the columns structures then runs over
138 * all rows and cells in the table. The function pointers in "tbl" are
139 * used for the actual width calculations.
140 */
141 void
142 tblcalc(struct rofftbl *tbl, const struct tbl_span *sp)
143 {
144     const struct tbl_dat *dp;
145     const struct tbl_head *hp;
146     struct roffcol *col;
147     int spans;

148     /*
149     * Allocate the master column specifiers. These will hold the
150     * widths and decimal positions for all cells in the column. It
151     * must be freed and nullified by the caller.
152     */

154     assert(NULL == tbl->cols);
155     tbl->cols = mandoc_calloc
156         ((size_t)sp->opts->cols, sizeof(struct roffcol));
157         ((size_t)sp->tbl->cols, sizeof(struct roffcol));

159     hp = sp->head;

161     for ( ; sp; sp = sp->next) {
162         if (TBL_SPAN_DATA != sp->pos)
163             continue;
164         spans = 1;
165         /*
166         * Account for the data cells in the layout, matching it
167         * to data cells in the data section.
168         */
169         for (dp = sp->first; dp; dp = dp->next) {
170             /* Do not used spanned cells in the calculation. */
171             if (0 < --spans)
172                 continue;
173             spans = dp->spans;
174             if (1 < spans)
175                 continue;
176             assert(dp->layout);
177             col = &tbl->cols[dp->layout->head->ident];
178             tblcalc_data(tbl, col, sp->opts, dp);
179             tblcalc_data(tbl, col, sp->tbl, dp);
180         }
181     }

182     /*
183     * Calculate width of the spanners. These get one space for a
184     * vertical line, two for a double-vertical line.
185     */

187     for ( ; hp; hp = hp->next) {
188         col = &tbl->cols[hp->ident];

```

```

189         switch (hp->pos) {
190             case (TBL_HEAD_VERT):
191                 col->width = (*tbl->len)(1, tbl->arg);
192                 break;
193             case (TBL_HEAD_DVERT):
194                 col->width = (*tbl->len)(2, tbl->arg);
195                 break;
196             default:
197                 break;
198         }
199     }
200 }

202 static void
203 tblcalc_data(struct rofftbl *tbl, struct roffcol *col,
204             const struct tbl_opts *opts, const struct tbl_dat *dp)
205 {
206     size_t sz;

208     /* Branch down into data sub-types. */

210     switch (dp->layout->pos) {
211         case (TBL_CELL_HORIZ):
212             /* FALLTHROUGH */
213         case (TBL_CELL_DHORIZ):
214             sz = (*tbl->len)(1, tbl->arg);
215             if (col->width < sz)
216                 col->width = sz;
217             break;
218         case (TBL_CELL_LONG):
219             /* FALLTHROUGH */
220         case (TBL_CELL_CENTRE):
221             /* FALLTHROUGH */
222         case (TBL_CELL_LEFT):
223             /* FALLTHROUGH */
224         case (TBL_CELL_RIGHT):
225             tblcalc_literal(tbl, col, dp);
226             break;
227         case (TBL_CELL_NUMBER):
228             tblcalc_number(tbl, col, opts, dp);
229             tblcalc_number(tbl, col, tp, dp);
230             break;
231         case (TBL_CELL_DOWN):
232             break;
233         default:
234             abort();
235             /* NOTREACHED */
236     }
237 }

239 unchanged_portion_omitted

241 static void
242 tblcalc_number(struct rofftbl *tbl, struct roffcol *col,
243             const struct tbl_opts *opts, const struct tbl_dat *dp)
244 {
245     int i;
246     size_t sz, psz, ssz, d;
247     const char *str;
248     char *cp;
249     char buf[2];

251     /*
252     * First calculate number width and decimal place (last + 1 for
253     * non-decimal numbers). If the stored decimal is subsequent to

```



```
243     * ours, make our size longer by that difference
244     * (right-"shifting"); similarly, if ours is subsequent the
245     * stored, then extend the stored size by the difference.
246     * Finally, re-assign the stored values.
247     */
249     str = dp->string ? dp->string : "";
250     sz = (*tbl->slen)(str, tbl->arg);
252     /* FIXME: TBL_DATA_HORIZ et al.? */
254     buf[0] = opts->decimal;
256     buf[0] = tp->decimal;
255     buf[1] = '\0';
257     psz = (*tbl->slen)(buf, tbl->arg);
259     if (NULL != (cp = strrchr(str, opts->decimal))) {
281     if (NULL != (cp = strrchr(str, tp->decimal))) {
260         buf[1] = '\0';
261         for (ssz = 0, i = 0; cp != &str[i]; i++) {
262             buf[0] = str[i];
263             ssz += (*tbl->slen)(buf, tbl->arg);
264         }
265         d = ssz + psz;
266     } else
267         d = sz + psz;
269     /* Adjust the settings for this column. */
271     if (col->decimal > d) {
272         sz += col->decimal - d;
273         d = col->decimal;
274     } else
275         col->width += d - col->decimal;
277     if (sz > col->width)
278         col->width = sz;
279     if (d > col->decimal)
280         col->decimal = d;
281 }
unchanged portion omitted
```

```

*****
10314 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/preconv.c
Update to 1.12.3.
*****
1 /*      $Id: preconv.c,v 1.6 2013/06/02 03:52:21 schwarze Exp $ */
1 /*      $Id: preconv.c,v 1.5 2011/07/24 18:15:14 kristaps Exp $ */
2 /*
3  * Copyright (c) 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif

21 #ifdef HAVE_MMAP
22 #include <sys/stat.h>
23 #include <sys/mman.h>
24 #endif

26 #include <assert.h>
27 #include <fcntl.h>
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <string.h>
31 #include <unistd.h>

33 /*
34  * The read_whole_file() and resize_buf() functions are copied from
35  * read.c, including all dependency code.
36  * read.c, including all dependency code (MAP_FILE, etc.).
37  */

38 #ifndef MAP_FILE
39 #define MAP_FILE 0
40 #endif

38 enum      enc {
39     ENC_UTF_8, /* UTF-8 */
40     ENC_US_ASCII, /* US-ASCII */
41     ENC_LATIN_1, /* Latin-1 */
42     ENC_MAX
43 };
    unchanged_portion_omitted

241 static int
242 read_whole_file(const char *f, int fd,
243                struct buf *fb, int *with_mmap)
244 {
245     size_t      off;
246     ssize_t     ssize;

248 #ifdef HAVE_MMAP
249     struct stat  st;
250     if (-1 == fstat(fd, &st)) {

```

```

251         perror(f);
252         return(0);
253     }

255     /*
256     * If we're a regular file, try just reading in the whole entry
257     * via mmap().  This is faster than reading it into blocks, and
258     * since each file is only a few bytes to begin with, I'm not
259     * concerned that this is going to tank any machines.
260     */

262     if (S_ISREG(st.st_mode) && st.st_size >= (1U << 31)) {
263         fprintf(stderr, "%s: input too large\n", f);
264         return(0);
265     }

267     if (S_ISREG(st.st_mode)) {
268         *with_mmap = 1;
269         fb->sz = (size_t)st.st_size;
270         fb->buf = mmap(NULL, fb->sz, PROT_READ, MAP_SHARED, fd, 0);
271         fb->buf = mmap(NULL, fb->sz, PROT_READ,
272                       MAP_FILE|MAP_SHARED, fd, 0);
273         if (fb->buf != MAP_FAILED)
274             return(1);
275     }
276 #endif

277     /*
278     * If this isn't a regular file (like, say, stdin), then we must
279     * go the old way and just read things in bit by bit.
280     */

281     *with_mmap = 0;
282     off = 0;
283     fb->sz = 0;
284     fb->buf = NULL;
285     for (;;) {
286         if (off == fb->sz && fb->sz == (1U << 31)) {
287             fprintf(stderr, "%s: input too large\n", f);
288             break;
289         }
290         if (off == fb->sz)
291             resize_buf(fb, 65536);

294         ssize = read(fd, fb->buf + (int)off, fb->sz - off);
295         if (ssize == 0) {
296             fb->sz = off;
297             return(1);
298         }
299         if (ssize == -1) {
300             perror(f);
301             break;
302         }
303         off += (size_t)ssize;
304     }

306     free(fb->buf);
307     fb->buf = NULL;
308     return(0);
309 }
    unchanged_portion_omitted

```

```

*****
2099 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/predefs.in
Update to 1.12.3.
*****
1 /* $Id: predefs.in,v 1.4 2012/07/18 10:39:19 schwarze Exp $ */
1 /* $Id: predefs.in,v 1.3 2011/07/31 11:36:49 schwarze Exp $ */
2 /*
3 * Copyright (c) 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4 *
5 * Permission to use, copy, modify, and distribute this software for any
6 * purpose with or without fee is hereby granted, provided that the above
7 * copyright notice and this permission notice appear in all copies.
8 *
9 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */

18 /*
19 * The predefined-string translation tables. Each corresponds to a
20 * predefined strings from (e.g.) tmac/mdoc/doc-nroff. The left-hand
21 * side corresponds to the input sequence (\*x, \*(xx and so on). The
22 * right-hand side is what's produced by libroff.
23 *
24 * XXX - C-escape strings!
25 * XXX - update PREDEF_MAX in roff.c if adding more!
26 */

28 PREDEF("Am", "&")
29 PREDEF("Ba", "\\fR|\\fP")
29 PREDEF("Ba", "|")
30 PREDEF("Ge", "\\(>=")
31 PREDEF("Gt", ">")
32 PREDEF("If", "infinity")
33 PREDEF("Le", "\\(<=")
34 PREDEF("Lq", "\\(lq")
35 PREDEF("Lt", "<")
36 PREDEF("Na", "NaN")
37 PREDEF("Ne", "\\(!=")
38 PREDEF("Pi", "pi")
39 PREDEF("Pm", "\\(+-" )
40 PREDEF("Rq", "\\(rq")
41 PREDEF("left-bracket", "[")
42 PREDEF("left-parenthesis", "(")
43 PREDEF("lp", "(")
44 PREDEF("left-singlequote", "\\(oq")
45 PREDEF("q", "\\(dq")
46 PREDEF("quote-left", "\\(oq")
47 PREDEF("quote-right", "\\(cq")
48 PREDEF("R", "\\(rg")
49 PREDEF("right-bracket", "]")
50 PREDEF("right-parenthesis", ")")
51 PREDEF("rp", ")")
52 PREDEF("right-singlequote", "\\(cq")
53 PREDEF("Tm", "(Tm)")
54 PREDEF("Px", "POSIX")
55 PREDEF("Ai", "ANSI")
56 PREDEF("\\'", "\\(')")
57 PREDEF("aa", "\\(aa")
58 PREDEF("ga", "\\(ga")
59 PREDEF("`", "\\(')

```

```

60 PREDEF("lq", "\\(lq")
61 PREDEF("rq", "\\(rq")
62 PREDEF("ua", "\\(ua")
63 PREDEF("va", "\\(va")
64 PREDEF("<=", "\\(<=")
65 PREDEF(">=", "\\(>=")

```

```

*****
19844 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/read.c
Update to 1.12.3.
*****
1 /* $Id: read.c,v 1.39 2013/09/16 00:25:07 schwarze Exp $ */
1 /* $Id: read.c,v 1.28 2012/02/16 20:51:31 joerg Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #ifdef HAVE_MMAP
23 #include <sys/stat.h>
24 #include <sys/mman.h>
25 #endif
26
27 #include <assert.h>
28 #include <ctype.h>
29 #include <fcntl.h>
30 #include <stdarg.h>
31 #include <stdint.h>
32 #include <stdio.h>
33 #include <stdlib.h>
34 #include <string.h>
35 #include <unistd.h>
36
37 #include "mandoc.h"
38 #include "libmandoc.h"
39 #include "mdoc.h"
40 #include "man.h"
41 #include "main.h"
42
43 #ifndef MAP_FILE
44 #define MAP_FILE 0
45 #endif
46
47 #define REPARSE_LIMIT 1000
48
49 struct buf {
50     char          *buf; /* binary input buffer */
51     size_t        sz; /* size of binary buffer */
52 };
53
54 struct mparse {
55     enum mandoclevel file_status; /* status of current parse */
56     enum mandoclevel wlevel; /* ignore messages below this */
57     int line; /* line number in the file */
58     enum mparset inttype; /* which parser to use */
59     struct man *pman; /* persistent man parser */

```

```

56     struct mdoc *pmdoc; /* persistent mdoc parser */
57     struct man *man; /* man parser */
58     struct mdoc *mdoc; /* mdoc parser */
59     struct roff *roff; /* roff parser (!NULL) */
60     int reparse_count; /* finite interp. stack */
61     mandocmsg mmsg; /* warning/error message handler */
62     void *arg; /* argument to mmsg */
63     const char *file;
64     struct buf *secondary;
65     char *defos; /* default operating system */
66 };
67
68 static void resize_buf(struct buf *, size_t);
69 static void mparse_buf_r(struct mparse *, struct buf, int);
70 static void mparse_readfd_r(struct mparse *, int, const char *, int);
71 static void pset(const char *, int, struct mparse *);
72 static int read_whole_file(const char *, int, struct buf *, int *);
73 static void mparse_end(struct mparse *);
74 static void mparse_parse_buffer(struct mparse *, struct buf,
75     const char *);
76
77 static const enum mandocerr mandoclimits[MANDOCLEVEL_MAX] = {
78     MANDOCERR_OK,
79     MANDOCERR_WARNING,
80     MANDOCERR_WARNING,
81     MANDOCERR_ERROR,
82     MANDOCERR_FATAL,
83     MANDOCERR_MAX,
84     MANDOCERR_MAX
85 };
86
87 static const char * const mandocerrs[MANDOCERR_MAX] = {
88     "ok",
89     "generic warning",
90
91     /* related to the prologue */
92     "no title in document",
93     "document title should be all caps",
94     "unknown manual section",
95     "unknown manual volume or arch",
96     "date missing, using today's date",
97     "cannot parse date, using it verbatim",
98     "prologue macros out of order",
99     "duplicate prologue macro",
100    "macro not allowed in prologue",
101    "macro not allowed in body",
102
103    /* related to document structure */
104    ".so is fragile, better use ln(1)",
105    "NAME section must come first",
106    "bad NAME section contents",
107    "manual name not yet set",
108    "sections out of conventional order",
109    "duplicate section name",
110    "section header suited to sections 2, 3, and 9 only",
111    "section not in conventional manual section",
112
113    /* related to macros and nesting */
114    "skipping obsolete macro",
115    "skipping paragraph macro",
116    "moving paragraph macro out of list",
117    "skipping no-space macro",
118    "blocks badly nested",
119    "child violates parent syntax",
120    "nested displays are not portable",

```

```

119     "already in literal mode",
120     "line scope broken",

122     /* related to missing macro arguments */
123     "skipping empty macro",
124     "argument count wrong",
125     "missing display type",
126     "list type must come first",
127     "tag lists require a width argument",
128     "missing font type",
129     "skipping end of block that is not open",

131     /* related to bad macro arguments */
132     "skipping argument",
133     "duplicate argument",
134     "duplicate display type",
135     "duplicate list type",
136     "unknown AT&T UNIX version",
137     "bad Boolean value",
138     "unknown font",
139     "unknown standard specifier",
140     "bad width argument",

142     /* related to plain text */
143     "blank line in non-literal context",
144     "tab in non-literal context",
145     "end of line whitespace",
146     "bad comment style",
147     "bad escape sequence",
148     "unterminated quoted string",

150     /* related to equations */
151     "unexpected literal in equation",
152
153     "generic error",

155     /* related to equations */
156     "unexpected equation scope closure",
157     "equation scope open on exit",
158     "overlapping equation scopes",
159     "unexpected end of equation",
160     "equation syntax error",

162     /* related to tables */
163     "bad table syntax",
164     "bad table option",
165     "bad table layout",
166     "no table layout cells specified",
167     "no table data cells specified",
168     "ignore data in cell",
169     "data block still open",
170     "ignoring extra data cells",

172     "input stack limit exceeded, infinite loop?",
173     "skipping bad character",
174     "escaped character not allowed in a name",
175     "manual name not yet set",
176     "skipping text before the first section header",
177     "skipping unknown macro",
178     "NOT IMPLEMENTED, please use groff: skipping request",
179     "argument count wrong",
180     "skipping column outside column list",
181     "skipping end of block that is not open",
182     "missing end of block",
183     "scope open on exit",
184     "uname(3) system call failed",

```

```

185     "macro requires line argument(s)",
186     "macro requires body argument(s)",
187     "macro requires argument(s)",
188     "request requires a numeric argument",
189     "missing list type",
190     "line argument(s) will be lost",
191     "body argument(s) will be lost",

193     "generic fatal error",

195     "not a manual",
196     "column syntax is inconsistent",
197     "NOT IMPLEMENTED: .Bd -file",
198     "argument count wrong, violates syntax",
199     "child violates parent syntax",
200     "argument count wrong, violates syntax",
201     "NOT IMPLEMENTED: .so with absolute path or \"..\"",
202     "no document body",
203     "no document prologue",
204     "static buffer exhausted",
205 };
    unchanged portion omitted

225 static void
226 pset(const char *buf, int pos, struct mparse *curp)
227 {
228     int            i;

230     /*
231     * Try to intuit which kind of manual parser should be used.  If
232     * passed in by command-line (-man, -mdoc), then use that
233     * explicitly.  If passed as -mandoc, then try to guess from the
234     * line: either skip dot-lines, use -mdoc when finding '.Dt', or
235     * default to -man, which is more lenient.
236     *
237     * Separate out pmdoc/pman from mdoc/man: the first persists
238     * through all parsers, while the latter is used per-parse.
239     */

241     if ('.' == buf[0] || '\\' == buf[0]) {
242         for (i = 1; buf[i]; i++)
243             if (' ' != buf[i] && '\t' != buf[i])
244                 break;
245         if ('\0' == buf[i])
246             return;
247     }

249     switch (curp->inttype) {
250     case (MPARSE_MDOC):
251         if (NULL == curp->pmdoc)
252             curp->pmdoc = mdoc_alloc(curp->roff, curp,
253             curp->defos);
254             curp->pmdoc = mdoc_alloc(curp->roff, curp);
255         assert(curp->pmdoc);
256         curp->mdoc = curp->pmdoc;
257         return;
258     case (MPARSE_MAN):
259         if (NULL == curp->pman)
260             curp->pman = man_alloc(curp->roff, curp);
261         assert(curp->pman);
262         curp->man = curp->pman;
263         return;
264     default:
265         break;
    }

```

```

267     if (pos >= 3 && 0 == memcmp(buf, ".Dd", 3)) {
268         if (NULL == curp->pmddoc)
269             curp->pmddoc = mdoc_alloc(curp->roff, curp,
270                                     curp->defos);
266         curp->pmddoc = mdoc_alloc(curp->roff, curp);
271         assert(curp->pmddoc);
272         curp->mdoc = curp->pmddoc;
273         return;
274     }

276     if (NULL == curp->pman)
277         curp->pman = man_alloc(curp->roff, curp);
278     assert(curp->pman);
279     curp->man = curp->pman;
280 }

282 /*
283  * Main parse routine for an opened file. This is called for each
284  * opened file and simply loops around the full input file, possibly
285  * nesting (i.e., with 'so').
286  */
287 static void
288 mparse_buf_r(struct mparse *curp, struct buf blk, int start)
289 {
290     const struct tbl_span *span;
291     struct buf ln;
292     enum rofferr rr;
293     int i, of, rc;
294     int pos; /* byte number in the ln buffer */
295     int lnn; /* line number in the real file */
296     unsigned char c;

298     memset(&ln, 0, sizeof(struct buf));

300     lnn = curp->line;
301     pos = 0;

303     for (i = 0; i < (int)blk.sz; ) {
304         if (0 == pos && '\0' == blk.buf[i])
305             break;

307         if (start) {
308             curp->line = lnn;
309             curp->reparse_count = 0;
310         }

312         while (i < (int)blk.sz && (start || '\0' != blk.buf[i])) {

314             /*
315              * When finding an unescaped newline character,
316              * leave the character loop to process the line.
317              * Skip a preceding carriage return, if any.
318              */

320             if ('\r' == blk.buf[i] && i + 1 < (int)blk.sz &&
321                 '\n' == blk.buf[i + 1])
322                 ++i;
323             if ('\n' == blk.buf[i]) {
324                 ++i;
325                 ++lnn;
326                 break;
327             }

329             /*
330              * Make sure we have space for at least
331              * one backslash and one other character

```

```

332         * and the trailing NUL byte.
333         */

335         if (pos + 2 >= (int)ln.sz)
336             resize_buf(&ln, 256);

338         /*
339          * Warn about bogus characters. If you're using
340          * non-ASCII encoding, you're screwing your
341          * readers. Since I'd rather this not happen,
342          * I'll be helpful and replace these characters
343          * with "?", so we don't display gibberish.
344          * Note to manual writers: use special characters.
345          */

347         c = (unsigned char) blk.buf[i];

349         if ( ! (isascii(c) &&
350                (isgraph(c) || isblank(c)))) {
351             mandoc_msg(MANDOCERR_BADCHAR, curp,
352                       curp->line, pos, NULL);
353             ++i;
354             if (pos >= (int)ln.sz)
355                 resize_buf(&ln, 256);
356             ln.buf[pos++] = '?';
357             continue;
358         }

360         /* Trailing backslash = a plain char. */

362         if ('\\" != blk.buf[i] || i + 1 == (int)blk.sz) {
363             if (pos >= (int)ln.sz)
364                 resize_buf(&ln, 256);
365             ln.buf[pos++] = blk.buf[i++];
366             continue;
367         }

369         /* Found escape and at least one other character.
370          * When it's a newline character, skip it.
371          * When there is a carriage return in between,
372          * skip that one as well.
373          */

375         if ('\r' == blk.buf[i + 1] && i + 2 < (int)blk.sz &&
376             '\n' == blk.buf[i + 2])
377             ++i;
378         if ('\n' == blk.buf[i + 1]) {
379             i += 2;
380             ++lnn;
381             continue;
382         }

384         if ('"' == blk.buf[i + 1] || '#' == blk.buf[i + 1]) {
385             i += 2;
386             /* Comment, skip to end of line */
387             for (; i < (int)blk.sz; ++i) {
388                 if ('\n' == blk.buf[i]) {
389                     ++i;
390                     ++lnn;
391                     break;
392                 }
393             }

395             /* Backout trailing whitespaces */
396             for (; pos > 0; --pos) {

```

```

394         if (ln.buf[pos - 1] != ' ')
395             break;
396         if (pos > 2 && ln.buf[pos - 2] == '\\\')
397             break;
398     }
399     break;
400 }
401
402 /* Catch escaped bogus characters. */
403
404 c = (unsigned char) blk.buf[i+1];
405
406 if ( ! (isascii(c) &&
407        (isgraph(c) || isblank(c)))) {
408     mandoc_msg(MANDOCERR_BADCHAR, curp,
409               curp->line, pos, NULL);
410     i += 2;
411     ln.buf[pos++] = '?';
412     continue;
413 }
414
415 /* Some other escape sequence, copy & cont. */
416
417 if (pos + 1 >= (int)ln.sz)
418     resize_buf(&ln, 256);
419
420 ln.buf[pos++] = blk.buf[i++];
421 ln.buf[pos++] = blk.buf[i++];
422 }
423
424 if (pos >= (int)ln.sz)
425     resize_buf(&ln, 256);
426
427 ln.buf[pos] = '\0';
428
429 /*
430  * A significant amount of complexity is contained by
431  * the roff preprocessor. It's line-oriented but can be
432  * expressed on one line, so we need at times to
433  * readjust our starting point and re-run it. The roff
434  * preprocessor can also readjust the buffers with new
435  * data, so we pass them in wholesale.
436  */
437
438 of = 0;
439
440 /*
441  * Maintain a lookaside buffer of all parsed lines. We
442  * only do this if mparse_keep() has been invoked (the
443  * buffer may be accessed with mparse_getkeep()).
444  */
445
446 if (curp->secondary) {
447     curp->secondary->buf =
448         mandoc_realloc
449         (curp->secondary->buf,
450          curp->secondary->sz + pos + 2);
451     memcpy(curp->secondary->buf +
452            curp->secondary->sz,
453            ln.buf, pos);
454     curp->secondary->sz += pos;
455     curp->secondary->buf
456         [curp->secondary->sz] = '\n';
457     curp->secondary->sz++;
458     curp->secondary->buf
459         [curp->secondary->sz] = '\0';

```

```

457     }
458     rerun:
459     rr = roff_parseln
460         (curp->roff, curp->line,
461          &ln.buf, &ln.sz, of, &of);
462
463     switch (rr) {
464     case (ROFF_REPARSE):
465         if (REPARSE_LIMIT >= ++curp->reparse_count)
466             mparse_buf_r(curp, ln, 0);
467         else
468             mandoc_msg(MANDOCERR_ROFFLOOP, curp,
469                       curp->line, pos, NULL);
470         pos = 0;
471         continue;
472     case (ROFF_APPEND):
473         pos = (int)strlen(ln.buf);
474         continue;
475     case (ROFF_RERUN):
476         goto rerun;
477     case (ROFF_IGN):
478         pos = 0;
479         continue;
480     case (ROFF_ERR):
481         assert(MANDOCLEVEL_FATAL <= curp->file_status);
482         break;
483     case (ROFF_SO):
484         /*
485          * We remove 'so' clauses from our lookaside
486          * buffer because we're going to descend into
487          * the file recursively.
488          */
489         if (curp->secondary)
490             curp->secondary->sz -= pos + 1;
491         mparse_readfd(curp, -1, ln.buf + of);
492         mparse_readfd_r(curp, -1, ln.buf + of, 1);
493         if (MANDOCLEVEL_FATAL <= curp->file_status)
494             break;
495         pos = 0;
496         continue;
497     default:
498         break;
499     }
500
501     /*
502     * If we encounter errors in the recursive parse, make
503     * sure we don't continue parsing.
504     */
505
506     if (MANDOCLEVEL_FATAL <= curp->file_status)
507         break;
508
509     /*
510     * If input parsers have not been allocated, do so now.
511     * We keep these instanced between parsers, but set them
512     * locally per parse routine since we can use different
513     * parsers with each one.
514     */
515
516     if ( ! (curp->man || curp->mdoc))
517         pset(ln.buf + of, pos - of, curp);
518
519     /*
520     * Lastly, push down into the parsers themselves. One
521     * of these will have already been set in the pset()
522     * routine.

```

```

522     * If libroff returns ROFF_TBL, then add it to the
523     * currently open parse.  Since we only get here if
524     * there does exist data (see tbl_data.c), we're
525     * guaranteed that something's been allocated.
526     * Do the same for ROFF_EQN.
527     */
529     rc = -1;

531     if (ROFF_TBL == rr)
532         while (NULL != (span = roff_span(curp->roff))) {
533             rc = curp->man ?
534                 man_addspan(curp->man, span) :
535                 mdoc_addspan(curp->mdoc, span);
536             if (0 == rc)
537                 break;
538         }
539     else if (ROFF_EQN == rr)
540         rc = curp->mdoc ?
541             mdoc_addeqn(curp->mdoc,
542                       roff_eqn(curp->roff)) :
543             man_addeqn(curp->man,
544                      roff_eqn(curp->roff));
545     else if (curp->man || curp->mdoc)
546         rc = curp->man ?
547             man_parseln(curp->man,
548                       curp->line, ln.buf, of) :
549             mdoc_parseln(curp->mdoc,
550                       curp->line, ln.buf, of);

552     if (0 == rc) {
553         assert(MANDOCLEVEL_FATAL <= curp->file_status);
554         break;
555     }

557     /* Temporary buffers typically are not full. */

559     if (0 == start && '\0' == blk.buf[i])
560         break;

562     /* Start the next input line. */

564     pos = 0;
565 }

567     free(ln.buf);
568 }

570 static int
571 read_whole_file(const char *file, int fd, struct buf *fb, int *with_mmap)
572 {
573     size_t     off;
574     ssize_t    ssz;

576 #ifdef HAVE_MMAP
577     struct stat st;
578     if (-1 == fstat(fd, &st)) {
579         perror(file);
580         return(0);
581     }
583     /*
584     * If we're a regular file, try just reading in the whole entry
585     * via mmap().  This is faster than reading it into blocks, and
586     * since each file is only a few bytes to begin with, I'm not
587     * concerned that this is going to tank any machines.

```

```

588     /*
590     if (S_ISREG(st.st_mode)) {
591         if (st.st_size >= (1U << 31)) {
592             fprintf(stderr, "%s: input too large\n", file);
593             return(0);
594         }
595         *with_mmap = 1;
596         fb->sz = (size_t)st.st_size;
597         fb->buf = mmap(NULL, fb->sz, PROT_READ, MAP_SHARED, fd, 0);
598         fb->buf = mmap(NULL, fb->sz, PROT_READ,
599                      MAP_FILE|MAP_SHARED, fd, 0);
600         if (fb->buf != MAP_FAILED)
601             return(1);
602     }
603 #endif

604     /*
605     * If this isn't a regular file (like, say, stdin), then we must
606     * go the old way and just read things in bit by bit.
607     */

608     *with_mmap = 0;
609     off = 0;
610     fb->sz = 0;
611     fb->buf = NULL;
612     for (;;) {
613         if (off == fb->sz) {
614             if (fb->sz == (1U << 31)) {
615                 fprintf(stderr, "%s: input too large\n", file);
616                 break;
617             }
618             resize_buf(fb, 65536);
619         }
620         ssz = read(fd, fb->buf + (int)off, fb->sz - off);
621         if (ssz == 0) {
622             fb->sz = off;
623             return(1);
624         }
625         if (ssz == -1) {
626             perror(file);
627             break;
628         }
629         off += (size_t)ssz;
630     }

632     free(fb->buf);
633     fb->buf = NULL;
634     return(0);
635 }

_____unchanged_portion_omitted_____

663 static void
664 mparse_parse_buffer(struct mparse *curp, struct buf blk, const char *file)
665 mparse_parse_buffer(struct mparse *curp, struct buf blk, const char *file,
666                    int re)
667 {
668     const char *svfile;
669     static int recursion_depth;

670     if (64 < recursion_depth) {
671         mandoc_msg(MANDOCERR_ROFFLOOP, curp, curp->line, 0, NULL);
672         return;
673     }

674     /* Line number is per-file. */

```



```

675     svfile = curp->file;
676     curp->file = file;
677     curp->line = 1;
678     recursion_depth++;

680     mparse_buf_r(curp, blk, 1);

682     if (0 == --recursion_depth && MANDOCLEVEL_FATAL > curp->file_status)
658     if (0 == re && MANDOCLEVEL_FATAL > curp->file_status)
683         mparse_end(curp);

685     curp->file = svfile;
686 }

688 enum mandoclevel
689 mparse_readmem(struct mparse *curp, const void *buf, size_t len,
690               const char *file)
691 {
692     struct buf blk;

694     blk.buf = UNCONST(buf);
695     blk.sz = len;

697     mparse_parse_buffer(curp, blk, file);
673     mparse_parse_buffer(curp, blk, file, 0);
698     return(curp->file_status);
699 }

701 enum mandoclevel
702 mparse_readfd(struct mparse *curp, int fd, const char *file)
677 static void
703 mparse_readfd_r(struct mparse *curp, int fd, const char *file, int re)
704 {
705     struct buf     blk;
706     int            with_mmap;

707     if (-1 == fd)
708         if (-1 == (fd = open(file, O_RDONLY, 0))) {
709             perror(file);
710             curp->file_status = MANDOCLEVEL_SYSERR;
711             goto out;
687             return;
712         }
713     /*
714     * Run for each opened file; may be called more than once for
715     * each full parse sequence if the opened file is nested (i.e.,
716     * from 'so'). Simply sucks in the whole file and moves into
717     * the parse phase for the file.
718     */

720     if (! read_whole_file(file, fd, &blk, &with_mmap)) {
721         curp->file_status = MANDOCLEVEL_SYSERR;
722         goto out;
698         return;
723     }

725     mparse_parse_buffer(curp, blk, file);
701     mparse_parse_buffer(curp, blk, file, re);

727 #ifdef HAVE_MMAP
728     if (with_mmap)
729         munmap(blk.buf, blk.sz);
730     else
731 #endif
732     free(blk.buf);

```

```

734     if (STDIN_FILENO != fd && -1 == close(fd))
735         perror(file);
736 out:
712 }

714 enum mandoclevel
715 mparse_readfd(struct mparse *curp, int fd, const char *file)
716 {
718     mparse_readfd_r(curp, fd, file, 0);
737     return(curp->file_status);
738 }

740 struct mparse *
741 mparse_alloc(enum mparset inttype, enum mandoclevel wlevel,
742             mandocmsg mmsg, void *arg, char *defos)
723 mparse_alloc(enum mparset inttype, enum mandoclevel wlevel, mandocmsg mmsg, void
743 {
744     struct mparse *curp;

746     assert(wlevel <= MANDOCLEVEL_FATAL);

748     curp = mandoc_calloc(1, sizeof(struct mparse));

750     curp->wlevel = wlevel;
751     curp->mmsg = mmsg;
752     curp->arg = arg;
753     curp->inttype = inttype;
754     curp->defos = defos;

756     curp->roff = roff_alloc(inttype, curp);
736     curp->roff = roff_alloc(curp);
757     return(curp);
758 }

```

unchanged\_portion\_omitted

```

*****
43919 Fri Jul 25 15:00:15 2014
new/usr/src/cmd/mandoc/roff.c
Update to 1.12.3.
*****
1 /* $Id: roff.c,v 1.189 2013/12/30 18:44:06 schwarze Exp $ */
1 /* $Id: roff.c,v 1.172 2011/10/24 21:41:45 schwarze Exp $ */
2 /*
3  * Copyright (c) 2010, 2011, 2012 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
3  * Copyright (c) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHORS DISCLAIM ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdio.h>
25 #include <stdlib.h>
26 #include <string.h>
27
28 #include "mandoc.h"
29 #include "libroff.h"
30 #include "libmandoc.h"
31
32 /* Maximum number of nested if-else conditionals. */
33 #define RSTACK_MAX 128
34
35 /* Maximum number of string expansions per line, to break infinite loops. */
36 #define EXPAND_LIMIT 1000
37
38 enum rofft {
39     ROFF_ad,
40     ROFF_am,
41     ROFF_ami,
42     ROFF_aml,
43     ROFF_cc,
44     ROFF_de,
45     ROFF_dei,
46     ROFF_del,
47     ROFF_ds,
48     ROFF_el,
49     ROFF_fam,
50     ROFF_hw,
51     ROFF_hy,
52     ROFF_ie,
53     ROFF_if,
54     ROFF_ig,
55     ROFF_it,
56     ROFF_ne,
57     ROFF_nh,
58     ROFF_nr,

```

```

59     ROFF_ns,
60     ROFF_ps,
61     ROFF_rm,
62     ROFF_so,
63     ROFF_ta,
64     ROFF_tr,
65     ROFF_Dd,
66     ROFF_TH,
67     ROFF_TS,
68     ROFF_TE,
69     ROFF_T_,
70     ROFF_EQ,
71     ROFF_EN,
72     ROFF_cblock,
73     ROFF_ccond,
74     ROFF_USERDEF,
75     ROFF_MAX
76 };
77
78 enum roffrule {
79     ROFFRULE_DENY,
80     ROFFRULE_ALLOW,
73     ROFFRULE_ALLOW,
74     ROFFRULE_DENY
81 };
82
83 /*
84  * A single register entity. If "set" is zero, the value of the
85  * register should be the default one, which is per-register.
86  * Registers are assumed to be unsigned ints for now.
87 */
88 struct reg {
89     int set; /* whether set or not */
90     unsigned int u; /* unsigned integer */
91 };
92
93 /*
94  * An incredibly-simple string buffer.
95 */
96 struct roffstr {
97     char *p; /* nil-terminated buffer */
98     size_t sz; /* saved strlen(p) */
99 };
100
101 unchanged_portion_omitted
102
103 /*
104  * A single number register as part of a singly-linked list.
105 */
106 struct roffreg {
107     struct roffstr key;
108     int val;
109     struct roffreg *next;
110 };
111
112 struct roff {
113     enum mparset parsetype; /* requested parse type */
114     struct mparse *parse; /* parse point */
115     struct roffnode *last; /* leaf of stack */
116     enum roffrule rstack[RSTACK_MAX]; /* stack of 'ie' rules */
117     char control; /* control character */
118     int rstackpos; /* position in rstack */
119     struct roffreg *regtab; /* number registers */
120     struct reg regs[REG_MAX];
121     struct roffkv *strtab; /* user-defined strings & macros */
122     struct roffkv *xmbtab; /* multi-byte trans table ('tr') */
123     struct roffstr *xtab; /* single-byte trans table ('tr') */

```

```

120     const char *current_string; /* value of last called user macro */
121     struct tbl_node *first_tbl; /* first table parsed */
122     struct tbl_node *last_tbl; /* last table parsed */
123     struct tbl_node *tbl; /* current table being parsed */
124     struct eqn_node *last_eqn; /* last equation parsed */
125     struct eqn_node *first_eqn; /* first equation parsed */
126     struct eqn_node *eqn; /* current equation being parsed */
127 };
    unchanged_portion_omitted
166 #define PREDEF(__name, __str) \
167     { (__name), (__str) },
169 static enum roffft      roffhash_find(const char *, size_t);
170 static void             roffhash_init(void);
171 static void             roffnode_cleanscope(struct roff *);
172 static void             roffnode_pop(struct roff *);
173 static void             roffnode_push(struct roff *, enum roffft,
174                                     const char *, int, int);
175 static enum rofferr     roff_block(ROFF_ARGS);
176 static enum rofferr     roff_block_text(ROFF_ARGS);
177 static enum rofferr     roff_block_sub(ROFF_ARGS);
178 static enum rofferr     roff_cblock(ROFF_ARGS);
179 static enum rofferr     roff_cc(ROFF_ARGS);
180 static enum rofferr     roff_ccond(ROFF_ARGS);
181 static enum rofferr     roff_cond(ROFF_ARGS);
182 static enum rofferr     roff_cond_text(ROFF_ARGS);
183 static enum rofferr     roff_cond_sub(ROFF_ARGS);
184 static enum rofferr     roff_ds(ROFF_ARGS);
185 static enum roffrule    roff_evalcond(const char *, int *);
186 static void             roff_freel(struct roff *);
187 static void             roff_freereg(struct roffreg *);
188 static void             roff_freestr(struct roffkv *);
189 static char             *roff_getname(struct roff *, char **, int, int);
190 static int              roff_getnum(const char *, int *, int *);
191 static int              roff_gettop(const char *, int *, char *);
192 static int              roff_getreg(const struct roff *,
193                                   const char *, size_t);
194 static const char      *roff_getstr(const struct roff *,
195                                   const char *, size_t);
196 static enum rofferr     roff_it(ROFF_ARGS);
197 static enum rofferr     roff_line_ignore(ROFF_ARGS);
198 static enum rofferr     roff_nr(ROFF_ARGS);
199 static void             roff_openeqn(struct roff *, const char *,
200                                   int, int, const char *);
201 static enum roffft      roff_parse(struct roff *, const char *, int *);
202 static enum rofferr     roff_parsertext(char **, size_t *, int, int *);
188 static enum rofferr     roff_parsertext(char *);
203 static enum rofferr     roff_res(struct roff *,
204                                   char **, size_t *, int, int);
205 static enum rofferr     roff_rm(ROFF_ARGS);
206 static void             roff_setstr(struct roff *,
207                                   const char *, const char *, int);
208 static void             roff_setstrn(struct roffkv **, const char *,
209                                   size_t, const char *, size_t, int);
210 static enum rofferr     roff_so(ROFF_ARGS);
211 static enum rofferr     roff_tr(ROFF_ARGS);
212 static enum rofferr     roff_Dd(ROFF_ARGS);
213 static enum rofferr     roff_TH(ROFF_ARGS);
214 static enum rofferr     roff_TE(ROFF_ARGS);
215 static enum rofferr     roff_TS(ROFF_ARGS);
216 static enum rofferr     roff_EQ(ROFF_ARGS);
217 static enum rofferr     roff_EN(ROFF_ARGS);
218 static enum rofferr     roff_T_(ROFF_ARGS);
219 static enum rofferr     roff_userdef(ROFF_ARGS);

```

```

221 /* See roffhash_find() */
223 #define ASCII_HI        126
224 #define ASCII_LO        33
225 #define HASHWIDTH      (ASCII_HI - ASCII_LO + 1)
227 static struct roffmac *hash[HASHWIDTH];
229 static struct roffmac roffs[ROFF_MAX] = {
230     {"ad", roff_line_ignore, NULL, NULL, 0, NULL },
231     {"am", roff_block, roff_block_text, roff_block_sub, 0, NULL },
232     {"ami", roff_block, roff_block_text, roff_block_sub, 0, NULL },
233     {"aml", roff_block, roff_block_text, roff_block_sub, 0, NULL },
234     {"cc", roff_cc, NULL, NULL, 0, NULL },
235     {"de", roff_block, roff_block_text, roff_block_sub, 0, NULL },
236     {"dei", roff_block, roff_block_text, roff_block_sub, 0, NULL },
237     {"del", roff_block, roff_block_text, roff_block_sub, 0, NULL },
238     {"ds", roff_ds, NULL, NULL, 0, NULL },
239     {"el", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
240     {"fam", roff_line_ignore, NULL, NULL, 0, NULL },
241     {"hw", roff_line_ignore, NULL, NULL, 0, NULL },
242     {"hy", roff_line_ignore, NULL, NULL, 0, NULL },
243     {"ie", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
244     {"if", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
245     {"ig", roff_block, roff_block_text, roff_block_sub, 0, NULL },
246     {"it", roff_it, NULL, NULL, 0, NULL },
247     {"it", roff_line_ignore, NULL, NULL, 0, NULL },
248     {"ne", roff_line_ignore, NULL, NULL, 0, NULL },
249     {"nh", roff_line_ignore, NULL, NULL, 0, NULL },
250     {"nr", roff_nr, NULL, NULL, 0, NULL },
251     {"ns", roff_line_ignore, NULL, NULL, 0, NULL },
252     {"ps", roff_line_ignore, NULL, NULL, 0, NULL },
253     {"rm", roff_rm, NULL, NULL, 0, NULL },
254     {"so", roff_so, NULL, NULL, 0, NULL },
255     {"ta", roff_line_ignore, NULL, NULL, 0, NULL },
256     {"tr", roff_tr, NULL, NULL, 0, NULL },
257     {"Dd", roff_Dd, NULL, NULL, 0, NULL },
258     {"TH", roff_TH, NULL, NULL, 0, NULL },
259     {"TS", roff_TS, NULL, NULL, 0, NULL },
260     {"TE", roff_TE, NULL, NULL, 0, NULL },
261     {"T&", roff_T_, NULL, NULL, 0, NULL },
262     {"EQ", roff_EQ, NULL, NULL, 0, NULL },
263     {"EN", roff_EN, NULL, NULL, 0, NULL },
264     {".", roff_cblock, NULL, NULL, 0, NULL },
265     {"\\", roff_ccond, NULL, NULL, 0, NULL },
266     {NULL, roff_userdef, NULL, NULL, 0, NULL }
267 };
268 const char *const __mdoc_reserved[] = {
269     "Ac", "Ad", "An", "Ao", "Ap", "Aq", "Ar", "At",
270     "Bc", "Bd", "Bf", "Bk", "Bl", "Bo", "Bq",
271     "Brc", "Bro", "Brq", "Bsx", "Bt", "Bx",
272     "Cd", "Cm", "Db", "Dc", "Dd", "Dl", "Do", "Dq",
273     "Ds", "Dt", "Dv", "Dx", "Dl",
274     "Ec", "Ed", "Ef", "Ek", "El", "Em", "em",
275     "En", "Eo", "Eq", "Er", "Es", "Ev", "Ex",
276     "Fa", "Fc", "Fd", "Fl", "Fn", "Fo", "Fr", "Ft", "Fx",
277     "Hf", "Ic", "In", "It", "Lb", "Li", "Lk", "Lp", "LP",
278     "Me", "Ms", "Mt", "Nd", "Nm", "No", "Ns", "Nx",
279     "Oc", "Oo", "Op", "Os", "Ot", "Ox",
280     "Pa", "Pc", "Pf", "Po", "Pp", "Pp", "Pp", "Pq",
281     "Qc", "Ql", "Qo", "Qq", "Or", "Rd", "Re", "Rs", "Rv",
282     "Sc", "Sf", "Sh", "SH", "Sm", "So", "Sq",
283     "Ss", "St", "Sx", "Sy",
284     "Ta", "Tn", "Ud", "Ux", "Va", "Vt", "Xc", "Xo", "Xr",
285     "%A", "%B", "%D", "%I", "%J", "%N", "%O",

```

```

286 "%P", "%Q", "%R", "%T", "%U", "%V",
287 NULL
288 };

290 const char *const __man_reserved[] = {
291 "AT", "B", "BI", "BR", "BT", "DE", "DS", "DT",
292 "EE", "EN", "EQ", "EX", "HF", "HP", "I", "IB", "IP", "IR",
293 "LP", "ME", "MT", "OP", "P", "PD", "PP", "PT",
294 "R", "RB", "RE", "RI", "RS", "SB", "SH", "SM", "SS", "SY",
295 "TE", "TH", "TP", "TQ", "TS", "T&", "UC", "UE", "UR", "YS",
296 NULL
297 };

299 /* Array of injected predefined strings. */
300 #define PREDEFS_MAX 38
301 static const struct predef predefs[PREDEFS_MAX] = {
302 #include "predefs.in"
303 };

305 /* See roffhash_find() */
306 #define ROFF_HASH(p) (p[0] - ASCII_LO)

308 static int roffit_lines; /* number of lines to delay */
309 static char *roffit_macro; /* nil-terminated macro line */

311 static void
312 roffhash_init(void)
313 {
314     struct roffmac *n;
315     int buc, i;

317     for (i = 0; i < (int)ROFF_USERDEF; i++) {
318         assert(roffs[i].name[0] >= ASCII_LO);
319         assert(roffs[i].name[0] <= ASCII_HI);

321         buc = ROFF_HASH(roffs[i].name);

323         if (NULL != (n = hash[buc])) {
324             for ( ; n->next; n = n->next)
325                 /* Do nothing. */ ;
326             n->next = &roffs[i];
327         } else
328             hash[buc] = &roffs[i];
329     }
330 }
    unchanged portion omitted

406 static void
407 roff_freel(struct roff *r)
408 {
409     struct tbl_node *tbl;
410     struct tbl_node *t;
411     struct eqn_node *e;
412     int i;

413     while (NULL != (tbl = r->first_tbl)) {
414         r->first_tbl = tbl->next;
415         tbl_free(tbl);
416     }
417     while (NULL != (t = r->first_tbl)) {
418         r->first_tbl = t->next;
419         tbl_free(t);
420     }

421     r->first_tbl = r->last_tbl = r->tbl = NULL;

```

```

420     while (NULL != (e = r->first_eqn)) {
421         r->first_eqn = e->next;
422         eqn_free(e);
423     }

425     r->first_eqn = r->last_eqn = r->eqn = NULL;

427     while (r->last)
428         roffnode_pop(r);

430     roff_freestr(r->strtab);
431     roff_freestr(r->xmbtab);

433     r->strtab = r->xmbtab = NULL;

435     roff_freereg(r->regtab);

437     r->regtab = NULL;

439     if (r->xtab)
440         for (i = 0; i < 128; i++)
441             free(r->xtab[i].p);

443     free(r->xtab);
444     r->xtab = NULL;
445 }

447 void
448 roff_reset(struct roff *r)
449 {
450     int i;

452     roff_freel(r);

454     r->control = 0;
455     memset(&r->regs, 0, sizeof(struct reg) * REG_MAX);

456     for (i = 0; i < PREDEFS_MAX; i++)
457         roff_setstr(r, predefs[i].name, predefs[i].str, 0);
458 }
    unchanged portion omitted

470 struct roff *
471 roff_alloc(enum mparset type, struct mparse *parse)
472 {
473     struct roff *r;
474     int i;

476     r = mandoc_calloc(1, sizeof(struct roff));
477     r->parsetype = type;
478     r->parse = parse;
479     r->rstackpos = -1;
480
481     roffhash_init();

483     for (i = 0; i < PREDEFS_MAX; i++)
484         roff_setstr(r, predefs[i].name, predefs[i].str, 0);

486     return(r);
487 }

489 /*
490 * In the current line, expand user-defined strings ("\\*")
491 * and references to number registers ("\\n").

```

```

492 * Also check the syntax of other escape sequences.
430 * Pre-filter each and every line for reserved words (one beginning with
431 * '\*', e.g., '\*(ab)'). These must be handled before the actual line
432 * is processed.
433 * This also checks the syntax of regular escapes.
493 */
494 static enum rofferr
495 roff_res(struct roff *r, char **bufp, size_t *szp, int ln, int pos)
496 {
497     char          ubuf[12]; /* buffer to print the number */
498     enum mandoc_esc esc;
499     const char    *stesc; /* start of an escape sequence ('\') */
500     const char    *stnam; /* start of the name, after "[(*" */
501     const char    *cp;    /* end of the name, e.g. before '[' */
502     const char    *res;   /* the string to be substituted */
503     char          *nbuf;  /* new buffer to copy bufp to */
504     size_t        nsz;    /* size of the new buffer */
505     size_t        maxl;   /* expected length of the escape name */
506     size_t        naml;   /* actual length of the escape name */
507     int           expand_count; /* to avoid infinite loops */
508     int           i, maxl, expand_count;
509     size_t        nsz;
510     char          *n;
511
512     expand_count = 0;
513
514 again:
515     cp = *bufp + pos;
516     while (NULL != (cp = strchr(cp, '\'))) {
517         stesc = cp++;
518
519         /*
520          * The second character must be an asterisk or an n.
521          * The second character must be an asterisk.
522          * If it isn't, skip it anyway: It is escaped,
523          * so it can't start another escape sequence.
524          */
525         if ('\0' == *cp)
526             return(ROFF_CONT);
527
528         switch (*cp) {
529             case ('*'):
530                 res = NULL;
531                 break;
532             case ('n'):
533                 res = ubuf;
534                 break;
535             default:
536                 if (ESCAPE_ERROR != mandoc_escape(&cp, NULL, NULL))
537                     if ('*' != *cp) {
538                         res = cp;
539                         esc = mandoc_escape(&cp, NULL, NULL);
540                         if (ESCAPE_ERROR != esc)
541                             continue;
542                         cp = res;
543                         mandoc_msg
544                             (MANDOCERR_BADESCAPE, r->parse,
545                              ln, (int)(stesc - *bufp), NULL);
546                         return(ROFF_CONT);
547                     }
548         }
549
550         cp++;
551
552         /*
553          * The third character decides the length

```

```

544         * of the name of the string or register.
545         * of the name of the string.
546         * Save a pointer to the name.
547         */
548
549         switch (*cp) {
550             case ('\0'):
551                 return(ROFF_CONT);
552             case ('('):
553                 cp++;
554                 maxl = 2;
555                 break;
556             case ('['):
557                 cp++;
558                 maxl = 0;
559                 break;
560             default:
561                 maxl = 1;
562                 break;
563         }
564         stnam = cp;
565
566         /* Advance to the end of the name. */
567         for (naml = 0; 0 == maxl || naml < maxl; naml++, cp++) {
568             for (i = 0; 0 == maxl || i < maxl; i++, cp++) {
569                 if ('\0' == *cp) {
570                     mandoc_msg
571                         (MANDOCERR_BADESCAPE,
572                          r->parse, ln,
573                          (int)(stesc - *bufp), NULL);
574                     return(ROFF_CONT);
575                 }
576                 if (0 == maxl && '[' == *cp)
577                     break;
578             }
579
580             /*
581              * Retrieve the replacement string; if it is
582              * undefined, resume searching for escapes.
583              */
584             if (NULL == res)
585                 res = roff_getstrn(r, stnam, naml);
586             else
587                 snprintf(ubuf, sizeof(ubuf), "%d",
588                          roff_getregn(r, stnam, naml));
589             res = roff_getstrn(r, stnam, (size_t)i);
590
591             if (NULL == res) {
592                 mandoc_msg
593                     (MANDOCERR_BADESCAPE, r->parse,
594                      ln, (int)(stesc - *bufp), NULL);
595                 res = "";
596             }
597
598             /* Replace the escape sequence by the string. */
599             pos = stesc - *bufp;
600
601             nsz = *szp + strlen(res) + 1;
602             nbuf = mandoc_malloc(nsz);
603             n = mandoc_malloc(nsz);
604
605             strncpy(nbuf, *bufp, (size_t)(stesc - *bufp + 1));
606             strcat(nbuf, res, nsz);

```

```

606     strlcat(nbuf, cp + (maxl ? 0 : 1), nsz);
607     strlcpy(n, *bufp, (size_t)(stesc - *bufp + 1));
608     strlcat(n, res, nsz);
609     strlcat(n, cp + (maxl ? 0 : 1), nsz);
610
611     free(*bufp);
612
613     *bufp = nbuf;
614     *bufp = n;
615     *szp = nsz;
616
617     if (EXPAND_LIMIT >= ++expand_count)
618         goto again;
619
620     /* Just leave the string unexpanded. */
621     mandoc_msg(MANDOCERR_ROFFLOOP, r->parse, ln, pos, NULL);
622     return(ROFF_IGN);
623 }
624
625 return(ROFF_CONT);
626 }
627
628 /*
629  * Process text streams:
630  * Convert all breakable hyphens into ASCII_HYPH.
631  * Decrement and spring input line trap.
632  * Process text streams: convert all breakable hyphens into ASCII_HYPH.
633  */
634 static enum rofferr
635 roff_parsertext(char **bufp, size_t *szp, int pos, int *offs)
636 roff_parsertext(char *p)
637 {
638     size_t      sz;
639     const char  *start;
640     char        *p;
641     int         isz;
642     enum mandoc_esc  esc;
643
644     start = p = *bufp + pos;
645     start = p;
646
647     while ('\0' != *p) {
648         sz = strcspn(p, "-\\");
649         p += sz;
650
651         if ('\0' == *p)
652             break;
653
654         if ('\\' == *p) {
655             /* Skip over escapes. */
656             p++;
657             esc = mandoc_escape((const char **)&p, NULL, NULL);
658             esc = mandoc_escape
659                 ((const char **)&p, NULL, NULL);
660             if (ESCAPE_ERROR == esc)
661                 break;
662             continue;
663         } else if (p == start) {
664             p++;
665             continue;
666         }
667
668         if (isalpha((unsigned char)p[-1]) &&
669             isalpha((unsigned char)p[1]))
670             *p = ASCII_HYPH;
671         p++;
672     }
673 }

```

```

664     /* Spring the input line trap. */
665     if (1 == roffit_lines) {
666         isz = asprintf(&p, "%s\n.%s", *bufp, roffit_macro);
667         if (-1 == isz) {
668             perror(NULL);
669             exit((int)MANDOCLEVEL_SYSERR);
670         }
671         free(*bufp);
672         *bufp = p;
673         *szp = isz + 1;
674         *offs = 0;
675         free(roffit_macro);
676         roffit_lines = 0;
677         return(ROFF_REPARSE);
678     } else if (1 < roffit_lines)
679         --roffit_lines;
680     return(ROFF_CONT);
681 }
682
683 enum rofferr
684 roff_parseln(struct roff *r, int ln, char **bufp,
685             size_t *szp, int pos, int *offs)
686 {
687     enum roffit      t;
688     enum rofferr     e;
689     int              ppos, ctl;
690
691     /*
692      * Run the reserved-word filter only if we have some reserved
693      * words to fill in.
694      */
695
696     e = roff_res(r, bufp, szp, ln, pos);
697     if (ROFF_IGN == e)
698         return(e);
699     assert(ROFF_CONT == e);
700
701     ppos = pos;
702     ctl = roff_getcontrol(r, *bufp, &pos);
703     ctl = mandoc_getcontrol(*bufp, &pos);
704
705     /*
706      * First, if a scope is open and we're not a macro, pass the
707      * text through the macro's filter. If a scope isn't open and
708      * we're not a macro, just let it through.
709      * Finally, if there's an equation scope open, divert it into it
710      * no matter our state.
711      */
712
713     if (r->last && !ctl) {
714         t = r->last->tok;
715         assert(roffs[t].text);
716         e = (*roffs[t].text)
717             (r, t, bufp, szp, ln, pos, pos, offs);
718         assert(ROFF_IGN == e || ROFF_CONT == e);
719         if (ROFF_CONT != e)
720             return(e);
721     }
722     if (r->eqn)
723         return(eqn_read(&r->eqn, ln, *bufp, ppos, offs));
724     if (!ctl) {
725         return(eqn_read(&r->eqn, ln, *bufp, pos, offs));
726         if (r->tbl)
727             return(tbl_read(r->tbl, ln, *bufp, pos));
728         return(roff_parsertext(bufp, szp, pos, offs));
729     }

```

```

727     }
736     return(roff_parsetext(*bufp + pos));
737 } else if ( !ctl) {
738     if (r->eqn)
739         return(eqn_read(&r->eqn, ln, *bufp, pos, offs));
740     if (r->tbl)
741         return(tbl_read(r->tbl, ln, *bufp, pos));
742     return(roff_parsetext(*bufp + pos));
743 } else if (r->eqn)
744     return(eqn_read(&r->eqn, ln, *bufp, ppos, offs));

```

```

729 /*
730  * If a scope is open, go to the child handler for that macro,
731  * as it may want to preprocess before doing anything with it.
732  * Don't do so if an equation is open.
733  */

```

```

735 if (r->last) {
736     t = r->last->tok;
737     assert(roffs[t].sub);
738     return((*roffs[t].sub)
739         (r, t, bufp, szp,
740          ln, ppos, pos, offs));
741 }

```

```

743 /*
744  * Lastly, as we've no scope open, try to look up and execute
745  * the new macro. If no macro is found, simply return and let
746  * the compilers handle it.
747  */

```

```

749 if (ROFF_MAX == (t = roff_parse(r, *bufp, &pos)))
750     return(ROFF_CONT);

```

```

752 assert(roffs[t].proc);
753 return((*roffs[t].proc)
754     (r, t, bufp, szp,
755      ln, ppos, pos, offs));
756 }

```

unchanged portion omitted

```

859 static void
860 roffnode_cleanscope(struct roff *r)
861 {

```

```

863     while (r->last) {
864         if (--r->last->endspan != 0)
865             if (--r->last->endspan < 0)
866                 break;
867         roffnode_pop(r);
868     }

```

unchanged portion omitted

```

1060 /* ARGSUSED */
1061 static enum rofferr
1062 roff_cond_sub(ROFF_ARGS)
1063 {
1064     enum roffft    t;
1065     enum roffrule  rr;
1066     char          *ep;

```

```

1068     rr = r->last->rule;
1069     roffnode_cleanscope(r);

```

```

1070     t = roff_parse(r, *bufp, &pos);

```

```

1072 /*
1073  * Fully handle known macros when they are structurally
1074  * required or when the conditional evaluated to true.
1075  * If the macro is unknown, first check if it contains a closing
1076  * delimiter '\}'. If it does, close out our scope and return
1077  * the currently-scoped rule (ignore or continue). Else, drop
1078  * into the currently-scoped rule.
1079  */

```

```

1077 if ((ROFF_MAX != t) &&
1078     (ROFF_ccond == t || ROFFRULE_ALLOW == rr ||
1079      ROFFMAC_STRUCT & roffs[t].flags)) {
1080     assert(roffs[t].proc);
1081     return((*roffs[t].proc)(r, t, bufp, szp,
1082                             ln, ppos, pos, offs));
1083 }

```

```

1085 /* Always check for the closing delimiter '\}'. */

```

```

995 if (ROFF_MAX == (t = roff_parse(r, *bufp, &pos))) {
1087     ep = &(*bufp)[pos];
1088     while (NULL != (ep = strchr(ep, '\\'))) {
1089         if ('}' != *(++ep))
1090             for ( ; NULL != (ep = strchr(ep, '\\')); ep++) {
1091                 ep++;
1092                 if ('}' != *ep)
1093                     continue;

```

```

1092 /*
1093  * Make the \} go away.
1094  * This is a little haphazard, as it's not quite
1095  * clear how nroff does this.
1096  * If we're at the end of line, then just chop
1097  * off the \} and resize the buffer.
1098  * If we aren't, then convert it to spaces.
1099  * If we aren't, then conver it to spaces.
1100  */

```

```

1098 if ('\0' == *(ep + 1)) {
1099     *--ep = '\0';
1100     *szp -= 2;
1101 } else
1102     *(ep - 1) = *ep = ' ';

```

```

1104     roff_ccond(r, ROFF_ccond, bufp, szp,
1105              ln, pos, pos + 2, offs);

```

```

1106     break;
1107 }
1108 return(ROFFRULE_DENY == rr ? ROFF_IGN : ROFF_CONT);
1109 }

```

```

1024 /*
1025  * A denied conditional must evaluate its children if and only
1026  * if they're either structurally required (such as loops and
1027  * conditionals) or a closing macro.
1028  */

```

```

1030 if (ROFFRULE_DENY == rr)
1031     if ( ! (ROFFMAC_STRUCT & roffs[t].flags))
1032         if (ROFF_ccond != t)
1033             return(ROFF_IGN);

```

```

1035 assert(roffs[t].proc);
1036 return((*roffs[t].proc)(r, t, bufp, szp,

```

```

1037         ln, ppos, pos, offs));
1109 }
    unchanged_portion_omitted_

1133 static int
1134 roff_getnum(const char *v, int *pos, int *res)
1135 {
1136     int p, n;

1138     p = *pos;
1139     n = v[p] == '-';
1140     if (n)
1141         p++;

1143     for (*res = 0; isdigit((unsigned char)v[p]); p++)
1144         *res += 10 * *res + v[p] - '0';
1145     if (p == *pos + n)
1146         return 0;

1148     if (n)
1149         *res = -*res;

1151     *pos = p;
1152     return 1;
1153 }

1155 static int
1156 roff_getop(const char *v, int *pos, char *res)
1157 {
1158     int e;

1160     *res = v[*pos];
1161     e = v[*pos + 1] == '=';

1163     switch (*res) {
1164     case '=':
1165         break;
1166     case '>':
1167         if (e)
1168             *res = 'g';
1169         break;
1170     case '<':
1171         if (e)
1172             *res = 'l';
1173         break;
1174     default:
1175         return(0);
1176     }

1178     *pos += 1 + e;

1180     return(*res);
1181 }

1183 static enum roffrule
1184 roff_evalcond(const char *v, int *pos)
1185 {
1186     int    not, lh, rh;
1187     char   op;

1189     switch (v[*pos]) {
1190     case ('n'):
1191         (*pos)++;
1192         return(ROFFRULE_ALLOW);
1193     case ('e'):
1194         /* FALLTHROUGH */

```

```

1195     case ('o'):
1196         /* FALLTHROUGH */
1197     case ('t'):
1198         (*pos)++;
1199         return(ROFFRULE_DENY);
1200     case ('!'):
1201         (*pos)++;
1202         not = 1;
1203         break;
1204     default:
1205         not = 0;
1206         break;
1207     }

1209     if (!roff_getnum(v, pos, &lh))
1210         return ROFFRULE_DENY;
1211     if (!roff_getop(v, pos, &op) {
1212         if (lh < 0)
1213             lh = 0;
1214         goto out;
1215     }
1216     if (!roff_getnum(v, pos, &rh))
1217         return ROFFRULE_DENY;
1218     switch (op) {
1219     case 'g':
1220         lh = lh >= rh;
1221         break;
1222     case 'l':
1223         lh = lh <= rh;
1224         break;
1225     case '=':
1226         lh = lh == rh;
1227         break;
1228     case '>':
1229         lh = lh > rh;
1230         break;
1231     case '<':
1232         lh = lh < rh;
1233         break;
1234     default:
1235         return ROFFRULE_DENY;
1236     }
1237     out:
1238     if (not)
1239         lh = !lh;
1240     return lh ? ROFFRULE_ALLOW : ROFFRULE_DENY;
1081     while (v[*pos] && ' ' != v[*pos])
1082         (*pos)++;
1083     return(ROFFRULE_DENY);
1241 }

1243 /* ARGSUSED */
1244 static enum rofferr
1245 roff_line_ignore(ROFF_ARGS)
1246 {

1091     if (ROFF_it == tok)
1092         mandoc_msg(MANDOCERR_REQUEST, r->parse, ln, ppos, "it");

1248     return(ROFF_IGN);
1249 }

1251 /* ARGSUSED */
1252 static enum rofferr
1253 roff_cond(ROFF_ARGS)
1254 {

```



```

1101     int          sv;
1102     enum roffrule rule;

1256     roffnode_push(r, tok, NULL, ln, ppos);

1258     /*
1259     * An '.el' has no conditional body: it will consume the value
1260     * of the current rstack entry set in prior 'ie' calls or
1261     * defaults to DENY.
1262     *
1263     * If we're not an 'el', however, then evaluate the conditional.
1264     */

1266     r->last->rule = ROFF_el == tok ?
1112     rule = ROFF_el == tok ?
1267         (r->rstackpos < 0 ?
1268         ROFFRULE_DENY : r->rstack[r->rstackpos--]) :
1269         roff_evalcond(*bufp, &ppos);

1117     sv = ppos;
1118     while (' ' == (*bufp)[pos])
1119         pos++;

1271     /*
1272     * Roff is weird. If we have just white-space after the
1273     * conditional, it's considered the BODY and we exit without
1274     * really doing anything. Warn about this. It's probably
1275     * wrong.
1276     */

1128     if ('\0' == (*bufp)[pos] && sv != pos) {
1129         mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
1130         return(ROFF_IGN);
1131     }

1133     roffnode_push(r, tok, NULL, ln, ppos);

1135     r->last->rule = rule;

1137     /*
1272     * An if-else will put the NEGATION of the current evaluated
1273     * conditional into the stack of rules.
1274     */

1276     if (ROFF_ie == tok) {
1277         if (r->rstackpos == RSTACK_MAX - 1) {
1278             mandoc_msg(MANDOCERR_MEM,
1279                 r->parse, ln, ppos, NULL);
1280             return(ROFF_ERR);
1281         }
1282         r->rstack[++r->rstackpos] =
1283             ROFFRULE_DENY == r->last->rule ?
1284             ROFFRULE_ALLOW : ROFFRULE_DENY;
1285     }

1287     /* If the parent has false as its rule, then so do we. */

1289     if (r->last->parent && ROFFRULE_DENY == r->last->parent->rule)
1290         r->last->rule = ROFFRULE_DENY;

1292     /*
1293     * Determine scope.
1294     * If there is nothing on the line after the conditional,
1295     * not even whitespace, use next-line scope.
1159     * Determine scope. If we're invoked with "{ " trailing the
1160     * conditional, then we're in a multiline scope. Else our scope

```

```

1161     * expires on the next line.
1296     */

1298     if ('\0' == (*bufp)[pos]) {
1299         r->last->endspan = 2;
1300         goto out;
1301     }
1164     r->last->endspan = 1;

1303     while (' ' == (*bufp)[pos])
1304         pos++;

1306     /* An opening brace requests multiline scope. */

1308     if ('\0' == (*bufp)[pos] && '{' == (*bufp)[pos + 1]) {
1309         r->last->endspan = -1;
1310         pos += 2;
1311         goto out;
1312     }

1314     /*
1315     * Anything else following the conditional causes
1316     * single-line scope. Warn if the scope contains
1317     * nothing but trailing whitespace.
1172     * If there are no arguments on the line, the next-line scope is
1173     * assumed.
1318     */

1320     if ('\0' == (*bufp)[pos])
1321         mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
1177     return(ROFF_IGN);

1323     r->last->endspan = 1;
1179     /* Otherwise re-run the roff parser after recalculating. */

1325 out:
1326     *offs = pos;
1327     return(ROFF_RERUN);
1328 }

_____unchanged_portion_omitted_____

1361 void
1362 roff_setreg(struct roff *r, const char *name, int val, char sign)
1363 {
1364     struct roffreg *reg;

1366     /* Search for an existing register with the same name. */
1367     reg = r->regtab;

1369     while (reg && strcmp(name, reg->key.p))
1370         reg = reg->next;

1372     if (NULL == reg) {
1373         /* Create a new register. */
1374         reg = mandoc_malloc(sizeof(struct roffreg));
1375         reg->key.p = mandoc_strdup(name);
1376         reg->key.sz = strlen(name);
1377         reg->val = 0;
1378         reg->next = r->regtab;
1379         r->regtab = reg;
1380     }

1382     if ('+' == sign)
1383         reg->val += val;
1384     else if ('-' == sign)
1385         reg->val -= val;

```

```

1386     else
1387         reg->val = val;
1388 }

1390 int
1391 roff_getreg(const struct roff *r, const char *name)
1217 roff_regisset(const struct roff *r, enum regs reg)
1392 {
1393     struct roffreg *reg;

1395     for (reg = r->regtab; reg; reg = reg->next)
1396         if (0 == strcmp(name, reg->key.p))
1397             return(reg->val);

1399     return(0);
1220     return(r->regs[(int)reg].set);
1400 }

1402 static int
1403 roff_getregn(const struct roff *r, const char *name, size_t len)
1223 unsigned int
1224 roff_regget(const struct roff *r, enum regs reg)
1404 {
1405     struct roffreg *reg;

1407     for (reg = r->regtab; reg; reg = reg->next)
1408         if (len == reg->key.sz &&
1409             0 == strncmp(name, reg->key.p, len))
1410             return(reg->val);

1412     return(0);
1227     return(r->regs[(int)reg].u);
1413 }

1415 static void
1416 roff_freereg(struct roffreg *reg)
1230 void
1231 roff_regunset(struct roff *r, enum regs reg)
1417 {
1418     struct roffreg *old_reg;

1420     while (NULL != reg) {
1421         free(reg->key.p);
1422         old_reg = reg;
1423         reg = reg->next;
1424         free(old_reg);
1425     }
1234     r->regs[(int)reg].set = 0;
1426 }

1428 /* ARGSUSED */
1429 static enum rofferr
1430 roff_nr(ROFF_ARGS)
1431 {
1432     const char *key;
1433     char *val;
1434     size_t sz;
1435     int iv;
1436     char sign;

1438     val = *bufp + pos;
1439     key = roff_getname(r, &val, ln, pos);

1441     sign = *val;
1442     if ('+' == sign || '-' == sign)
1443         val++;

```

```

1248     if (0 == strcmp(key, "nS")) {
1249         r->regs[(int)REG_nS].set = 1;
1250         if ((iv = mandoc_strntoi(val, strlen(val), 10)) >= 0)
1251             r->regs[(int)REG_nS].u = (unsigned)iv;
1252     } else
1253         r->regs[(int)REG_nS].u = 0u;
1254 }

1445     sz = strspn(val, "0123456789");
1446     iv = sz ? mandoc_strntoi(val, sz, 10) : 0;

1448     roff_setreg(r, key, iv, sign);

1450     return(ROFF_IGN);
1451 }
_____unchanged_portion_omitted_____

1469 /* ARGSUSED */
1470 static enum rofferr
1471 roff_it(ROFF_ARGS)
1472 {
1473     char *cp;
1474     size_t len;
1475     int iv;

1477     /* Parse the number of lines. */
1478     cp = *bufp + pos;
1479     len = strcspn(cp, " \t");
1480     cp[len] = '\0';
1481     if ((iv = mandoc_strntoi(cp, len, 10)) <= 0) {
1482         mandoc_msg(MANDOCERR_NUMERIC, r->parse,
1483                 ln, ppos, *bufp + 1);
1484         return(ROFF_IGN);
1485     }
1486     cp += len + 1;

1488     /* Arm the input line trap. */
1489     roffit_lines = iv;
1490     roffit_macro = mandoc_strdup(cp);
1491     return(ROFF_IGN);
1492 }

1494 /* ARGSUSED */
1495 static enum rofferr
1496 roff_Dd(ROFF_ARGS)
1497 {
1498     const char *const *cp;

1500     if (MPARSE_MDOC != r->parsetype)
1501         for (cp = __mdoc_reserved; *cp; cp++)
1502             roff_setstr(r, *cp, NULL, 0);

1504     return(ROFF_CONT);
1505 }

1507 /* ARGSUSED */
1508 static enum rofferr
1509 roff_TH(ROFF_ARGS)
1510 {
1511     const char *const *cp;

1513     if (MPARSE_MDOC != r->parsetype)
1514         for (cp = __man_reserved; *cp; cp++)
1515             roff_setstr(r, *cp, NULL, 0);

1517     return(ROFF_CONT);

```

```

1518 }

1520 /* ARGSUSED */
1521 static enum rofferr
1522 roff_TE(ROFF_ARGS)
1523 {
1525     if (NULL == r->tbl)
1526         mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
1527     else
1528         tbl_end(&r->tbl);

1530     return(ROFF_IGN);
1531 }
_____ unchanged_portion_omitted _____

1596 /* ARGSUSED */
1597 static enum rofferr
1598 roff_TS(ROFF_ARGS)
1599 {
1600     struct tbl_node *tbl;
1601     struct tbl_node *t;

1602     if (r->tbl) {
1603         mandoc_msg(MANDOCERR_SCOPEBROKEN, r->parse, ln, ppos, NULL);
1604         tbl_end(&r->tbl);
1605     }

1607     tbl = tbl_alloc(ppos, ln, r->parse);
1608     t = tbl_alloc(ppos, ln, r->parse);

1609     if (r->last_tbl)
1610         r->last_tbl->next = tbl;
1611     else
1612         r->first_tbl = r->last_tbl = tbl;
1613     r->first_tbl = r->last_tbl = t;

1614     r->tbl = r->last_tbl = tbl;
1615     r->tbl = r->last_tbl = t;
1616     return(ROFF_IGN);
1617 }

1618 /* ARGSUSED */
1619 static enum rofferr
1620 roff_CC(ROFF_ARGS)
1621 {
1622     const char *p;

1624     p = *bufp + pos;

1626     if ('\0' == *p || '.' == (r->control = *p++))
1627         r->control = 0;

1629     if ('\0' != *p)
1630         mandoc_msg(MANDOCERR_ARGCOUNT, r->parse, ln, ppos, NULL);

1632     return(ROFF_IGN);
1633 }

1635 /* ARGSUSED */
1636 static enum rofferr
1637 roff_TR(ROFF_ARGS)
1638 {
1639     const char *p, *first, *second;
1640     size_t fsz, ssz;

```

```

1641     enum mandoc_esc esc;

1643     p = *bufp + pos;

1645     if ('\0' == *p) {
1646         mandoc_msg(MANDOCERR_ARGCOUNT, r->parse, ln, ppos, NULL);
1647         return(ROFF_IGN);
1648     }

1650     while ('\0' != *p) {
1651         fsz = ssz = 1;

1653         first = p++;
1654         if ('\\" == *first) {
1655             esc = mandoc_escape(&p, NULL, NULL);
1656             if (ESCAPE_ERROR == esc) {
1657                 mandoc_msg
1658                     (MANDOCERR_BADESCAPE, r->parse,
1659                      ln, (int)(p - *bufp), NULL);
1660                 return(ROFF_IGN);
1661             }
1662             fsz = (size_t)(p - first);
1663         }

1665         second = p++;
1666         if ('\\" == *second) {
1667             esc = mandoc_escape(&p, NULL, NULL);
1668             if (ESCAPE_ERROR == esc) {
1669                 mandoc_msg
1670                     (MANDOCERR_BADESCAPE, r->parse,
1671                      ln, (int)(p - *bufp), NULL);
1672                 return(ROFF_IGN);
1673             }
1674             ssz = (size_t)(p - second);
1675         } else if ('\0' == *second) {
1676             mandoc_msg(MANDOCERR_ARGCOUNT, r->parse,
1677                        ln, (int)(p - *bufp), NULL);
1678             second = " ";
1679             p--;
1680         }

1682         if (fsz > 1) {
1683             roff_setstrn(&r->xmftab, first,
1684                        fsz, second, ssz, 0);
1685             continue;
1686         }

1688         if (NULL == r->xtab)
1689             r->xtab = mandoc_calloc
1690                 (128, sizeof(struct roffstr));

1692         free(r->xtab[(int)*first].p);
1693         r->xtab[(int)*first].p = mandoc_strdup(second, ssz);
1694         r->xtab[(int)*first].sz = ssz;
1695     }

1697     return(ROFF_IGN);
1698 }
_____ unchanged_portion_omitted _____

1725 /* ARGSUSED */
1726 static enum rofferr
1727 roff_USERDEF(ROFF_ARGS)
1728 {
1729     const char *arg[9];
1730     char *cp, *n1, *n2;

```

```

1731     int             i;

1733     /*
1734     * Collect pointers to macro argument strings
1735     * and NUL-terminate them.
1736     * and null-terminate them.
1737     */
1738     cp = *bufp + pos;
1739     for (i = 0; i < 9; i++)
1740         arg[i] = '\0' == *cp ? "" :
            mandoc_getarg(r->parse, &cp, ln, &pos);

1742     /*
1743     * Expand macro arguments.
1744     */
1745     *szp = 0;
1746     nl = cp = mandoc_strdup(r->current_string);
1747     while (NULL != (cp = strstr(cp, "\\$"))) {
1748         i = cp[2] - '1';
1749         if (0 > i || 8 < i) {
1750             /* Not an argument invocation. */
1751             cp += 2;
1752             continue;
1753         }

1755         *szp = strlen(nl) - 3 + strlen(arg[i]) + 1;
1756         n2 = mandoc_malloc(*szp);

1758         strncpy(n2, nl, (size_t)(cp - nl + 1));
1759         strcat(n2, arg[i], *szp);
1760         strcat(n2, cp + 3, *szp);

1762         cp = n2 + (cp - nl);
1763         free(nl);
1764         nl = n2;
1765     }

1767     /*
1768     * Replace the macro invocation
1769     * by the expanded macro.
1770     */
1771     free(*bufp);
1772     *bufp = nl;
1773     if (0 == *szp)
1774         *szp = strlen(*bufp) + 1;

1776     return(*szp > 1 && '\n' == (*bufp)[(int)*szp - 2] ?
1777         ROFF_REPARSE : ROFF_APPEND);
1778 }

```

unchanged\_portion\_omitted\_

```

1940 /*
1941 * Duplicate an input string, making the appropriate character
1942 * conversations (as stipulated by 'tr') along the way.
1943 * Returns a heap-allocated string with all the replacements made.
1944 */
1945 char *
1946 roff_strdup(const struct roff *r, const char *p)
1947 {
1948     const struct roffkv *cp;
1949     char             *res;
1950     const char       *pp;
1951     size_t           ssz, sz;
1952     enum mandoc_esc  esc;

1954     if (NULL == r->xmbtab && NULL == r->xtab)

```

```

1955         return(mandoc_strdup(p));
1956     else if ('\0' == *p)
1957         return(mandoc_strdup(""));

1959     /*
1960     * Step through each character looking for term matches
1961     * (remember that a 'tr' can be invoked with an escape, which is
1962     * a glyph but the escape is multi-character).
1963     * We only do this if the character hash has been initialised
1964     * and the string is >0 length.
1965     */

1967     res = NULL;
1968     ssz = 0;

1970     while ('\0' != *p) {
1971         if ('\0' != *p && r->xtab && r->xtab[(int)*p].p) {
1972             sz = r->xtab[(int)*p].sz;
1973             res = mandoc_realloc(res, ssz + sz + 1);
1974             memcpy(res + ssz, r->xtab[(int)*p].p, sz);
1975             ssz += sz;
1976             p++;
1977             continue;
1978         } else if ('\0' != *p) {
1979             res = mandoc_realloc(res, ssz + 2);
1980             res[ssz++] = *p++;
1981             continue;
1982         }

1984         /* Search for term matches. */
1985         for (cp = r->xmbtab; cp; cp = cp->next)
1986             if (0 == strcmp(p, cp->key.p, cp->key.sz))
1987                 break;

1989         if (NULL != cp) {
1990             /*
1991             * A match has been found.
1992             * Append the match to the array and move
1993             * forward by its keysize.
1994             */
1995             res = mandoc_realloc
1996                 (res, ssz + cp->val.sz + 1);
1997             memcpy(res + ssz, cp->val.p, cp->val.sz);
1998             ssz += cp->val.sz;
1999             p += (int)cp->key.sz;
2000             continue;
2001         }

2003         /*
2004         * Handle escapes carefully: we need to copy
2005         * over just the escape itself, or else we might
2006         * do replacements within the escape itself.
2007         * Make sure to pass along the bogus string.
2008         */
2009         pp = p++;
2010         esc = mandoc_escape(&p, NULL, NULL);
2011         if (ESCAPE_ERROR == esc) {
2012             sz = strlen(pp);
2013             res = mandoc_realloc(res, ssz + sz + 1);
2014             memcpy(res + ssz, pp, sz);
2015             break;
2016         }
2017     /*
2018     * We bail out on bad escapes.
2019     * No need to warn: we already did so when
2020     * roff_res() was called.

```

```
2021         */
2022         sz = (int)(p - pp);
2023         res = mandoc_realloc(res, ssz + sz + 1);
2024         memcpy(res + ssz, pp, sz);
2025         ssz += sz;
2026     }

2028     res[(int)ssz] = '\0';
2029     return(res);
2030 }

2032 /*
2033  * Find out whether a line is a macro line or not.
2034  * If it is, adjust the current position and return one; if it isn't,
2035  * return zero and don't change the current position.
2036  * If the control character has been set with '.cc', then let that grain
2037  * precedence.
2038  * This is slightly contrary to groff, where using the non-breaking
2039  * control character when 'cc' has been invoked will cause the
2040  * non-breaking macro contents to be printed verbatim.
2041  */
2042 int
2043 roff_getcontrol(const struct roff *r, const char *cp, int *ppos)
2044 {
2045     int          pos;

2047     pos = *ppos;

2049     if (0 != r->control && cp[pos] == r->control)
2050         pos++;
2051     else if (0 != r->control)
2052         return(0);
2053     else if ('\\" == cp[pos] && '.' == cp[pos + 1])
2054         pos += 2;
2055     else if ('.' == cp[pos] || '\'' == cp[pos])
2056         pos++;
2057     else
2058         return(0);

2060     while (' ' == cp[pos] || '\t' == cp[pos])
2061         pos++;

2063     *ppos = pos;
2064     return(1);
2065 }
_unchanged_portion_omitted_
```

\*\*\*\*\*

4887 Fri Jul 25 15:00:16 2014

new/usr/src/cmd/mandoc/st.in

Update to 1.12.3.

\*\*\*\*\*

```

1 /* $Id: st.in,v 1.22 2013/12/25 14:09:32 schwarze Exp $ */
1 /* $Id: st.in,v 1.19 2012/02/26 21:47:09 schwarze Exp $ */
2 /*
3  * Copyright (c) 2009, 2010 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
18 /*
19  * This file defines the .St macro arguments. If you add a new
20  * standard, make sure that the left-and side corresponds to the .St
21  * argument (like .St -p1003.1) and the right-hand side corresponds to
22  * the formatted output string.
23  *
24  * Be sure to escape strings.
25  * The non-breaking blanks prevent ending an output line right before
26  * a number. Groff prevent line breaks at the same places.
27  *
28  * REMEMBER TO ADD NEW STANDARDS TO MDOC.7!
29  */
31 LINE("-p1003.1-88", "IEEE Std 1003.1-1988 (\\(lqPOSIX.1\\(rq)")
32 LINE("-p1003.1-90", "IEEE Std 1003.1-1990 (\\(lqPOSIX.1\\(rq)")
33 LINE("-p1003.1-96", "ISO/IEC 9945-1:1996 (\\(lqPOSIX.1\\(rq)")
34 LINE("-p1003.1-2001", "IEEE Std 1003.1-2001 (\\(lqPOSIX.1\\(rq)")
35 LINE("-p1003.1-2004", "IEEE Std 1003.1-2004 (\\(lqPOSIX.1\\(rq)")
36 LINE("-p1003.1-2008", "IEEE Std 1003.1-2008 (\\(lqPOSIX.1\\(rq)")
37 LINE("-p1003.1", "IEEE Std 1003.1 (\\(lqPOSIX.1\\(rq)")
38 LINE("-p1003.1b", "IEEE Std 1003.1b (\\(lqPOSIX.1b\\(rq)")
39 LINE("-p1003.1b-93", "IEEE Std 1003.1b-1993 (\\(lqPOSIX.1b\\(rq)")
40 LINE("-p1003.1c-95", "IEEE Std 1003.1c-1995 (\\(lqPOSIX.1c\\(rq)")
41 LINE("-p1003.1d-99", "IEEE Std 1003.1d-1999 (\\(lqPOSIX.1d\\(rq)")
42 LINE("-p1003.1g-2000", "IEEE Std 1003.1g-2000 (\\(lqPOSIX.1g\\(rq)")
43 LINE("-p1003.1i-95", "IEEE Std 1003.1i-1995 (\\(lqPOSIX.1i\\(rq)")
44 LINE("-p1003.1j-2000", "IEEE Std 1003.1j-2000 (\\(lqPOSIX.1j\\(rq)")
45 LINE("-p1003.1q-2000", "IEEE Std 1003.1q-2000 (\\(lqPOSIX.1q\\(rq)")
46 LINE("-p1003.2", "IEEE Std 1003.2 (\\(lqPOSIX.2\\(rq)")
38 LINE("-p1003.1b", "IEEE Std 1003.1b (\\(lqPOSIX.1\\(rq)")
39 LINE("-p1003.1b-93", "IEEE Std 1003.1b-1993 (\\(lqPOSIX.1\\(rq)")
40 LINE("-p1003.1c-95", "IEEE Std 1003.1c-1995 (\\(lqPOSIX.1\\(rq)")
41 LINE("-p1003.1g-2000", "IEEE Std 1003.1g-2000 (\\(lqPOSIX.1\\(rq)")
42 LINE("-p1003.1i-95", "IEEE Std 1003.1i-1995 (\\(lqPOSIX.1\\(rq)")
47 LINE("-p1003.2-92", "IEEE Std 1003.2-1992 (\\(lqPOSIX.2\\(rq)")
48 LINE("-p1003.2a-92", "IEEE Std 1003.2a-1992 (\\(lqPOSIX.2\\(rq)")
49 LINE("-p1387.2", "IEEE Std 1387.2 (\\(lqPOSIX.7.2\\(rq)")
50 LINE("-p1387.2-95", "IEEE Std 1387.2-1995 (\\(lqPOSIX.7.2\\(rq)")
46 LINE("-p1003.2", "IEEE Std 1003.2 (\\(lqPOSIX.2\\(rq)")
47 LINE("-p1387.2", "IEEE Std 1387.2 (\\(lqPOSIX.7.2\\(rq)")
51 LINE("-isoC", "ISO/IEC 9899:1990 (\\(lqISO\\(rq)")
52 LINE("-isoC-90", "ISO/IEC 9899:1990 (\\(lqISO\\(rq)")
53 LINE("-isoC-amd1", "ISO/IEC 9899/AMD1:1995 (\\(lqISO\\(rq)")

```

```

54 LINE("-isoC-tcor1", "ISO/IEC 9899/TCOR1:1994 (\\(lqISO\\(rq)")
55 LINE("-isoC-tcor2", "ISO/IEC 9899/TCOR2:1995 (\\(lqISO\\(rq)")
56 LINE("-isoC-99", "ISO/IEC 9899:1999 (\\(lqISO\\(rq)")
57 LINE("-isoC-2011", "ISO/IEC 9899:2011 (\\(lqISO\\(rq)")
58 LINE("-iso9945-1-90", "ISO/IEC 9945-1:1990 (\\(lqPOSIX.1\\(rq)")
59 LINE("-iso9945-1-96", "ISO/IEC 9945-1:1996 (\\(lqPOSIX.1\\(rq)")
60 LINE("-iso9945-2-93", "ISO/IEC 9945-2:1993 (\\(lqPOSIX.2\\(rq)")
61 LINE("-ansiC", "ANSI X3.159-1989 (\\(lqANSI\\(rq)")
62 LINE("-ansiC-89", "ANSI X3.159-1989 (\\(lqANSI\\(rq)")
63 LINE("-ansiC-99", "ANSI/ISO/IEC 9899-1999 (\\(lqANSI\\(rq)")
64 LINE("-ieee754", "IEEE Std 754-1985")
65 LINE("-iso8802-3", "ISO 8802-3: 1989")
66 LINE("-iso8601", "ISO 8601")
67 LINE("-ieeel275-94", "IEEE Std 1275-1994 (\\(lqOpen Firmware\\(rq)")
68 LINE("-xpg3", "X/Open Portability Guide Issue\\(rq)")
69 LINE("-xpg4", "X/Open Portability Guide Issue\\(rq)")
70 LINE("-xpg4.2", "X/Open Portability Guide Issue\\(rq)", Version\\(rq)", 2)
71 LINE("-xpg4.3", "X/Open Portability Guide Issue\\(rq)", Version\\(rq)", 3)
72 LINE("-xbd5", "X/Open Base Definitions Issue\\(rq)", 5)
73 LINE("-xcu5", "X/Open Commands and Utilities Issue\\(rq)", 5)
74 LINE("-xsh4.2", "X/Open System Interfaces and Headers Issue\\(rq)", 4, Version
75 LINE("-xsh5", "X/Open System Interfaces and Headers Issue\\(rq)", 5)
76 LINE("-xns5", "X/Open Networking Services Issue\\(rq)", 5)
77 LINE("-xns5.2", "X/Open Networking Services Issue\\(rq)", 5.2)
78 LINE("-xns5.2d2.0", "X/Open Networking Services Issue\\(rq)", 5.2 Draft\\(rq)", 2.0)
79 LINE("-xcurses4.2", "X/Open Curses Issue\\(rq)", 4, Version\\(rq)", 2)
80 LINE("-susv2", "Version\\(rq)", 2 of the Single UNIX Specification (\\(lqSUSV
81 LINE("-susv3", "Version\\(rq)", 3 of the Single UNIX Specification (\\(lqSUSV
76 LINE("-susv2", "Version\\(rq)", 2 of the Single UNIX Specification")
77 LINE("-susv3", "Version\\(rq)", 3 of the Single UNIX Specification")
82 LINE("-svid4", "System\\(rq)", V Interface Definition, Fourth Edition (\\(lqS

```

new/usr/src/cmd/mandoc/tbl.c

1

```
*****
4085 Fri Jul 25 15:00:16 2014
new/usr/src/cmd/mandoc/tbl.c
Update to 1.12.3.
*****
1 /* $Id: tbl.c,v 1.27 2013/05/31 22:08:09 schwarze Exp $ */
1 /* $Id: tbl.c,v 1.26 2011/07/25 15:37:00 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <stdio.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <time.h>
27
28 #include "mandoc.h"
29 #include "libmandoc.h"
30 #include "libroff.h"
31
32 enum rofferr
33 tbl_read(struct tbl_node *tbl, int ln, const char *p, int offs)
34 {
35     int len;
36     const char *cp;
37
38     cp = &p[offs];
39     len = (int)strlen(cp);
40
41     /*
42      * If we're in the options section and we don't have a
43      * terminating semicolon, assume we've moved directly into the
44      * layout section. No need to report a warning: this is,
45      * apparently, standard behaviour.
46      */
47
48     if (TBL_PART_OPTS == tbl->part && len)
49         if (';' != cp[len - 1])
50             tbl->part = TBL_PART_LAYOUT;
51
52     /* Now process each logical section of the table. */
53
54     switch (tbl->part) {
55     case (TBL_PART_OPTS):
56         return(tbl_option(tbl, ln, p) ? ROFF_IGN : ROFF_ERR);
57     case (TBL_PART_LAYOUT):
58         return(tbl_layout(tbl, ln, p) ? ROFF_IGN : ROFF_ERR);
59     case (TBL_PART_CDATA):
60         return(tbl_cdata(tbl, ln, p) ? ROFF_TBL : ROFF_IGN);

```

new/usr/src/cmd/mandoc/tbl.c

2

```
61     default:
62         break;
63     }
64
65     /*
66      * This only returns zero if the line is empty, so we ignore it
67      * and continue on.
68      */
69     return(tbl_data(tbl, ln, p) ? ROFF_TBL : ROFF_IGN);
70 }
71
72 struct tbl_node *
73 tbl_alloc(int pos, int line, struct mparse *parse)
74 {
75     struct tbl_node *tbl;
76     struct tbl_node *p;
77
78     tbl = mandoc_malloc(1, sizeof(struct tbl_node));
79     tbl->line = line;
80     tbl->pos = pos;
81     tbl->parse = parse;
82     tbl->part = TBL_PART_OPTS;
83     tbl->opts.tab = '\t';
84     tbl->opts.linesize = 12;
85     tbl->opts.decimal = '.';
86     return(tbl);
87
88     p = mandoc_malloc(1, sizeof(struct tbl_node));
89     p->line = line;
90     p->pos = pos;
91     p->parse = parse;
92     p->part = TBL_PART_OPTS;
93     p->opts.tab = '\t';
94     p->opts.linesize = 12;
95     p->opts.decimal = '.';
96     return(p);
97 }
98
99 void
100 tbl_free(struct tbl_node *tbl)
101 {
102     struct tbl_row *rp;
103     struct tbl_cell *cp;
104     struct tbl_span *sp;
105     struct tbl_dat *dp;
106     struct tbl_head *hp;
107
108     while (NULL != (rp = tbl->first_row)) {
109         tbl->first_row = rp->next;
110         while (NULL != (cp = rp->first_cell)) {
111             rp->first_cell = cp->next;
112             while (NULL != (sp = cp->first_span)) {
113                 cp->first_span = sp->next;
114                 while (NULL != (dp = sp->first_dat)) {
115                     sp->first_dat = dp->next;

```

```
112         if (dp->string)
113             free(dp->string);
114         free(dp);
115     }
116     free(sp);
117 }
```

```
119 while (NULL != (hp = tbl->first_head)) {
120     tbl->first_head = hp->next;
119 while (NULL != (hp = p->first_head)) {
120     p->first_head = hp->next;
121     free(hp);
122 }
```

```
124     free(tbl);
124     free(p);
125 }
```

unchanged\_portion\_omitted



```

*****
6344 Fri Jul 25 15:00:16 2014
new/usr/src/cmd/mandoc/tbl_data.c
Update to 1.12.3.
*****
1 /* $Id: tbl_data.c,v 1.27 2013/06/01 04:56:50 schwarze Exp $ */
1 /* $Id: tbl_data.c,v 1.24 2011/03/20 16:02:05 kristaps Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <time.h>
27
28 #include "mandoc.h"
29 #include "libmandoc.h"
30 #include "libroff.h"
31
32 static int data(struct tbl_node *, struct tbl_span *,
33                int, const char *, int *);
34 static struct tbl_span *newspan(struct tbl_node *, int,
35                                struct tbl_row *);
36
37 static int
38 data(struct tbl_node *tbl, struct tbl_span *dp,
39       int ln, const char *p, int *pos)
40 {
41     struct tbl_dat *dat;
42     struct tbl_cell *cp;
43     int sv, spans;
44
45     cp = NULL;
46     if (dp->last && dp->last->layout)
47         cp = dp->last->layout->next;
48     else if (NULL == dp->last)
49         cp = dp->layout->first;
50
51     /*
52      * Skip over spanners, since
53      * Skip over spanners and vertical lines to data formats, since
54      * we want to match data with data layout cells in the header.
55      */
56     while (cp && TBL_CELL_SPAN == cp->pos)
57     while (cp && (TBL_CELL_VERT == cp->pos ||
58                 TBL_CELL_DVERT == cp->pos ||
59                 TBL_CELL_SPAN == cp->pos))

```

```

57         cp = cp->next;
58
59     /*
60      * Stop processing when we reach the end of the available layout
61      * cells. This means that we have extra input.
62      */
63
64     if (NULL == cp) {
65         mandoc_msg(MANDOCERR_TBLEXTRADAT,
66                  tbl->parse, ln, *pos, NULL);
67         /* Skip to the end... */
68         while (p[*pos])
69             (*pos)++;
70         return(1);
71     }
72
73     dat = mandoc_calloc(1, sizeof(struct tbl_dat));
74     dat->layout = cp;
75     dat->pos = TBL_DATA_NONE;
76
77     assert(TBL_CELL_SPAN != cp->pos);
78
79     for (spans = 0, cp = cp->next; cp; cp = cp->next)
80         if (TBL_CELL_SPAN == cp->pos)
81             spans++;
82         else
83             break;
84
85     dat->spans = spans;
86
87     if (dp->last) {
88         dp->last->next = dat;
89         dp->last = dat;
90     } else
91         dp->last = dp->first = dat;
92
93     sv = *pos;
94     while (p[*pos] && p[*pos] != tbl->opts.tab)
95         (*pos)++;
96
97     /*
98      * Check for a continued-data scope opening. This consists of a
99      * trailing 'T{' at the end of the line. Subsequent lines,
100     * until a standalone 'T}', are included in our cell.
101     */
102
103     if (*pos - sv == 2 && 'T' == p[sv] && '{' == p[sv + 1]) {
104         tbl->part = TBL_PART_CDATA;
105         return(1);
106     }
107     return(0);
108
109     assert(*pos - sv >= 0);
110
111     dat->string = mandoc_malloc((size_t)(*pos - sv + 1));
112     memcpy(dat->string, &p[sv], (size_t)(*pos - sv));
113     dat->string[*pos - sv] = '\0';
114
115     if (p[*pos])
116         (*pos)++;
117
118     if ( ! strcmp(dat->string, "-"))
119         dat->pos = TBL_DATA_HORIZ;
120     else if ( ! strcmp(dat->string, "="))
121         dat->pos = TBL_DATA_DHORIZ;
122     else if ( ! strcmp(dat->string, "\\_"))

```

```
122         dat->pos = TBL_DATA_NHORIZ;
123     else if (! strcmp(dat->string, "\\="))
124         dat->pos = TBL_DATA_NDHORIZ;
125     else
126         dat->pos = TBL_DATA_DATA;

128     if (TBL_CELL_HORIZ == dat->layout->pos ||
129         TBL_CELL_DHORIZ == dat->layout->pos ||
130         TBL_CELL_DOWN == dat->layout->pos)
131         if (TBL_DATA_DATA == dat->pos && '\0' != *dat->string)
132             mandoc_msg(MANDOCERR_TBLIGNDATA,
133                       tbl->parse, ln, sv, NULL);

135     return(1);
136 }
```

---

unchanged portion omitted

```
181 static struct tbl_span *
182 newspan(struct tbl_node *tbl, int line, struct tbl_row *rp)
183 {
184     struct tbl_span *dp;

186     dp = mandoc_calloc(1, sizeof(struct tbl_span));
187     dp->line = line;
188     dp->opts = &tbl->opts;
189     dp->tbl = &tbl->opts;
189     dp->layout = rp;
190     dp->head = tbl->first_head;

192     if (tbl->last_span) {
193         tbl->last_span->next = dp;
194         tbl->last_span = dp;
195     } else {
196         tbl->last_span = tbl->first_span = dp;
197         tbl->current_span = NULL;
198         dp->flags |= TBL_SPAN_FIRST;
199     }

201     return(dp);
202 }
```

---

unchanged portion omitted

```

*****
3160 Fri Jul 25 15:00:16 2014
new/usr/src/cmd/mandoc/tbl_html.c
Update to 1.12.3.
*****
1 /* $Id: tbl_html.c,v 1.10 2012/05/27 17:54:54 schwarze Exp $ */
1 /* $Id: tbl_html.c,v 1.9 2011/09/18 14:14:15 schwarze Exp $ */
2 /*
3  * Copyright (c) 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif
21 #include <assert.h>
22 #include <stdio.h>
23 #include <stdlib.h>
24 #include <string.h>
26 #include "mandoc.h"
27 #include "out.h"
28 #include "html.h"
30 static void html_tblogen(struct html *, const struct tbl_span *);
31 static size_t html_tbl_len(size_t, void *);
32 static size_t html_tbl_strlen(const char *, void *);
34 /* ARGSUSED */
35 static size_t
36 html_tbl_len(size_t sz, void *arg)
37 {
38
39     return(sz);
40 }
42 unchanged portion omitted
44
45 void
46 print_tbl(struct html *h, const struct tbl_span *sp)
47 {
48     const struct tbl_head *hp;
49     const struct tbl_dat *dp;
50     struct htmlpair tag;
51     struct tag *tt;
53
54     /* Inhibit printing of spaces: we do padding ourselves. */
56
57     if (NULL == h->tblt)
58         html_tblogen(h, sp);
60
61     assert(h->tblt);
63
64     h->flags |= HTML_NONOSPACE;
65     h->flags |= HTML_NOSPACE;

```

```

107     tt = print_otag(h, TAG_TR, 0, NULL);
109     switch (sp->pos) {
110     case (TBL_SPAN_HORIZ):
111         /* FALLTHROUGH */
112     case (TBL_SPAN_DHORIZ):
113         PAIR_INIT(&tag, ATTR_COLSPAN, "0");
114         print_otag(h, TAG_TD, 1, &tag);
115         break;
116     default:
117         dp = sp->first;
118         for (hp = sp->head; hp; hp = hp->next) {
119             print_stagq(h, tt);
120             print_otag(h, TAG_TD, 0, NULL);
122         switch (hp->pos) {
123         case (TBL_HEAD_VERT):
124             /* FALLTHROUGH */
125         case (TBL_HEAD_DVERT):
126             continue;
127         case (TBL_HEAD_DATA):
128             if (NULL == dp)
129                 break;
130             if (TBL_CELL_DOWN != dp->layout->pos)
131                 if (dp->string)
132                     print_text(h, dp->string);
133             dp = dp->next;
134             break;
135         }
136         break;
137     }
138     }
139     print_tagq(h, tt);
140
141     h->flags &= ~HTML_NONOSPACE;
142
143     if (TBL_SPAN_LAST & sp->flags) {
144         assert(h->tbl.cols);
145         free(h->tbl.cols);
146         h->tbl.cols = NULL;
147         print_tbclose(h);
148     }
149 }
150 unchanged portion omitted

```

new/usr/src/cmd/mandoc/tbl\_layout.c

1

```
*****
8025 Fri Jul 25 15:00:16 2014
new/usr/src/cmd/mandoc/tbl_layout.c
Update to 1.12.3.
*****
1 /* $Id: tbl_layout.c,v 1.23 2012/05/27 17:54:54 schwarze Exp $ */
1 /* $Id: tbl_layout.c,v 1.22 2011/09/18 14:14:15 schwarze Exp $ */
2 /*
3  * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2012 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <time.h>
27
28 #include "mandoc.h"
29 #include "libmandoc.h"
30 #include "libroff.h"
31
32 struct tbl_phrase {
33     char name;
34     enum tbl_cellt key;
35 };
36
37 /*
38  * FIXME: we can make this parse a lot nicer by, when an error is
39  * encountered in a layout key, bailing to the next key (i.e. to the
40  * next whitespace then continuing).
41  */
42
43 #define KEYS_MAX 11
44
45 static const struct tbl_phrase keys[KEYS_MAX] = {
46     { 'c', TBL_CELL_CENTRE },
47     { 'r', TBL_CELL_RIGHT },
48     { 'l', TBL_CELL_LEFT },
49     { 'n', TBL_CELL_NUMBER },
50     { 's', TBL_CELL_SPAN },
51     { 'a', TBL_CELL_LONG },
52     { '^', TBL_CELL_DOWN },
53     { '-', TBL_CELL_HORIZ },
54     { '_', TBL_CELL_HORIZ },
55     { '=', TBL_CELL_DHORIZ },
56     { ',', TBL_CELL_DHORIZ },
57     { '|', TBL_CELL_VERT }
58 };
59
60 static int
61 mods(struct tbl_node *, struct tbl_cell *,
```

new/usr/src/cmd/mandoc/tbl\_layout.c

2

```

59     int, const char *, int *);
60 static int
61     cell(struct tbl_node *, struct tbl_row *,
62         int, const char *, int *);
63 static void
64     row(struct tbl_node *, int, const char *, int *);
65 static struct tbl_cell *cell_alloc(struct tbl_node *, struct tbl_row *,
66     enum tbl_cellt, int vert);
67 static struct tbl_cell *cell_alloc(struct tbl_node *,
68     struct tbl_row *, enum tbl_cellt);
69 static void
70     head_adjust(const struct tbl_cell *,
71     struct tbl_head *);
72
73 static int
74 mods(struct tbl_node *tbl, struct tbl_cell *cp,
75     int ln, const char *p, int *pos)
76 {
77     char buf[5];
78     int i;
79
80     /* Not all types accept modifiers. */
81
82     switch (cp->pos) {
83     case (TBL_CELL_DOWN):
84         /* FALLTHROUGH */
85     case (TBL_CELL_HORIZ):
86         /* FALLTHROUGH */
87     case (TBL_CELL_DHORIZ):
88         /* FALLTHROUGH */
89     case (TBL_CELL_VERT):
90         /* FALLTHROUGH */
91     case (TBL_CELL_DVERT):
92         return(1);
93     default:
94         break;
95     }
96
97 mod:
98     /*
99     * XXX: since, at least for now, modifiers are non-conflicting
100    * (are separable by value, regardless of position), we let
101    * modifiers come in any order. The existing tbl doesn't let
102    * this happen.
103    */
104     switch (p[*pos]) {
105     case ('\0'):
106         /* FALLTHROUGH */
107     case (' '):
108         /* FALLTHROUGH */
109     case ('\t'):
110         /* FALLTHROUGH */
111     case (','):
112         /* FALLTHROUGH */
113     case ('.'):
114         return(1);
115     default:
116         break;
117     }
118
119     /* Throw away parenthesised expression. */
120
121     if (('(' == p[*pos]) {
122         (*pos)++;
123         while (p[*pos] && ')' != p[*pos])
124             (*pos)++;
125         if (')' == p[*pos]) {
126             (*pos)++;
127             goto mod;
128         }
129     }
130 }
```

```

117     }
118     mandoc_msg(MANDOCERR_TBLAYOUT,
119               tbl->parse, ln, *pos, NULL);
120     return(0);
121 }
122
123 /* Parse numerical spacing from modifier string. */
124
125 if (isdigit((unsigned char)p[*pos])) {
126     for (i = 0; i < 4; i++) {
127         if (! isdigit((unsigned char)p[*pos + i]))
128             break;
129         buf[i] = p[*pos + i];
130     }
131     buf[i] = '\0';
132
133     /* No greater than 4 digits. */
134
135     if (4 == i) {
136         mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
137                   ln, *pos, NULL);
138         return(0);
139     }
140
141     *pos += i;
142     cp->spacing = (size_t)atoi(buf);
143
144     goto mod;
145     /* NOTREACHED */
146 }
147
148 /* TODO: GNU has many more extensions. */
149
150 switch (tolower((unsigned char)p[(*pos)+])) {
151 case ('z'):
152     cp->flags |= TBL_CELL_WIGN;
153     goto mod;
154 case ('u'):
155     cp->flags |= TBL_CELL_UP;
156     goto mod;
157 case ('e'):
158     cp->flags |= TBL_CELL_EQUAL;
159     goto mod;
160 case ('t'):
161     cp->flags |= TBL_CELL_TALIGN;
162     goto mod;
163 case ('d'):
164     cp->flags |= TBL_CELL_BALIGN;
165     goto mod;
166 case ('w'): /* XXX for now, ignore minimal column width */
167     goto mod;
168 case ('f'):
169     break;
170 case ('r'):
171     /* FALLTHROUGH */
172 case ('b'):
173     /* FALLTHROUGH */
174 case ('i'):
175     (*pos)--;
176     break;
177 default:
178     mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
179               ln, *pos - 1, NULL);
180     return(0);
181 }

```

```

183     switch (tolower((unsigned char)p[(*pos)+])) {
184     case ('3'):
185         /* FALLTHROUGH */
186     case ('b'):
187         cp->flags |= TBL_CELL_BOLD;
188         goto mod;
189     case ('2'):
190         /* FALLTHROUGH */
191     case ('i'):
192         cp->flags |= TBL_CELL_ITALIC;
193         goto mod;
194     case ('1'):
195         /* FALLTHROUGH */
196     case ('r'):
197         goto mod;
198     default:
199         break;
200     }
201
202     mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
203               ln, *pos - 1, NULL);
204     return(0);
205 }
206
207 static int
208 cell(struct tbl_node *tbl, struct tbl_row *rp,
209       int ln, const char *p, int *pos)
210 {
211     int          vert, i;
212     int          i;
213     enum tbl_cellt  c;
214
215     /* Handle vertical lines. */
216     /* Parse the column position ('r', 'R', '/', ...). */
217
218     for (vert = 0; '|' == p[*pos]; ++*pos)
219         vert++;
220     while (' ' == p[*pos])
221         (*pos)++;
222
223     /* Parse the column position ('c', 'l', 'r', ...). */
224
225     for (i = 0; i < KEYS_MAX; i++)
226         if (tolower((unsigned char)p[*pos]) == keys[i].name)
227             break;
228
229     if (KEYS_MAX == i) {
230         mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
231                   ln, *pos, NULL);
232         return(0);
233     }
234
235     c = keys[i].key;
236
237     /*
238      * If a span cell is found first, raise a warning and abort the
239      * parse. If a span cell is found and the last layout element
240      * isn't a "normal" layout, bail.
241      *
242      * FIXME: recover from this somehow?
243      */
244
245     if (TBL_CELL_SPAN == c) {
246         if (NULL == rp->first) {
247             mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
248                       ln, *pos, NULL);

```

```

247         return(0);
248     } else if (rp->last)
249         switch (rp->last->pos) {
250             case (TBL_CELL_VERT):
251             case (TBL_CELL_DVERT):
252             case (TBL_CELL_HORIZ):
253             case (TBL_CELL_DHORIZ):
254                 mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse,
255                     ln, *pos, NULL);
256                 return(0);
257             default:
258                 break;
259         }
260     }
261     /*
262     * If a vertical spanner is found, we may not be in the first
263     * row.
264     */
265     if (TBL_CELL_DOWN == c && rp == tbl->first_row) {
266         mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse, ln, *pos, NULL);
267         return(0);
268     }
269     (*pos)++;
270
271     /* Extra check for the double-vertical. */
272     if (TBL_CELL_VERT == c && '|' == p[*pos]) {
273         (*pos)++;
274         c = TBL_CELL_DVERT;
275     }
276
277     /* Disallow adjacent spacers. */
278     if (vert > 2) {
279         if (rp->last && (TBL_CELL_VERT == c || TBL_CELL_DVERT == c) &&
280             (TBL_CELL_VERT == rp->last->pos ||
281              TBL_CELL_DVERT == rp->last->pos)) {
282             mandoc_msg(MANDOCERR_TBLAYOUT, tbl->parse, ln, *pos - 1, NULL);
283             return(0);
284         }
285     }
286
287     /* Allocate cell then parse its modifiers. */
288     return(mods(tbl, cell_alloc(tbl, rp, c, vert), ln, p, pos));
289     return(mods(tbl, cell_alloc(tbl, rp, c), ln, p, pos));
290 }
291
292 static void
293 row(struct tbl_node *tbl, int ln, const char *p, int *pos)
294 {
295     struct tbl_row *rp;
296
297     /*
298     * EBNF describing this section:
299     *
300     * row      ::= row_list [:space:]* [.]?[\n]
301     * row_list ::= [:space:]* row_elem row_tail
302     * row_tail ::= [:space:]*[,] row_list |
303     *              epsilon
304     * row_elem ::= [\t\ ]*[:alpha:]+
305     */

```

```

300     rp = mandoc_malloc(1, sizeof(struct tbl_row));
301     if (tbl->last_row)
302         if (tbl->last_row) {
303             tbl->last_row->next = rp;
304         } else
305             tbl->first_row = rp;
306     tbl->last_row = rp;
307     } else
308         tbl->last_row = tbl->first_row = rp;
309
310 cell:
311     while (isspace((unsigned char)p[*pos]))
312         (*pos)++;
313
314     /* Safely exit layout context. */
315     if ('.' == p[*pos]) {
316         tbl->part = TBL_PART_DATA;
317         if (NULL == tbl->first_row)
318             mandoc_msg(MANDOCERR_TBLNOLAYOUT, tbl->parse,
319                 ln, *pos, NULL);
320         (*pos)++;
321         return;
322     }
323
324     /* End (and possibly restart) a row. */
325     if (',' == p[*pos]) {
326         (*pos)++;
327         goto row;
328     } else if ('\0' == p[*pos])
329         return;
330
331     if (! cell(tbl, rp, ln, p, pos))
332         return;
333
334     goto cell;
335     /* NOTREACHED */
336 }
337
338 unchanged_portion_omitted
339
340 static struct tbl_cell *
341 cell_alloc(struct tbl_node *tbl, struct tbl_row *rp, enum tbl_cellt pos,
342            int vert)
343 cell_alloc(struct tbl_node *tbl, struct tbl_row *rp, enum tbl_cellt pos)
344 {
345     struct tbl_cell *p, *pp;
346     struct tbl_head *h, *hp;
347
348     p = mandoc_malloc(1, sizeof(struct tbl_cell));
349
350     if (NULL != (pp = rp->last)) {
351         pp->next = p;
352         h = pp->head->next;
353     } else {
354         rp->first = p;
355         h = tbl->first_head;
356     }
357     rp->last->next = p;
358     rp->last = p;
359     } else
360         rp->last = rp->first = p;
361
362     p->pos = pos;
363     p->vert = vert;

```

```

370  /* Re-use header. */
375  /*
376  * This is a little bit complicated. Here we determine the
377  * header the corresponds to a cell. We add headers dynamically
378  * when need be or re-use them, otherwise. As an example, given
379  * the following:
380  *
381  *     1  c || 1
382  *     2  | c | 1
383  *     3  1 1
384  *     3  || c | 1 |.
385  *
386  * We first add the new headers (as there are none) in (1); then
387  * in (2) we insert the first spanner (as it doesn't match up
388  * with the header); then we re-use the prior data headers,
389  * skipping over the spanners; then we re-use everything and add
390  * a last spanner. Note that VERT headers are made into DVERT
391  * ones.
392  */

394  h = pp ? pp->head->next : tbl->first_head;

372  if (h) {
397      /* Re-use data header. */
398      if (TBL_HEAD_DATA == h->pos &&
399          (TBL_CELL_VERT != p->pos &&
400           TBL_CELL_DVERT != p->pos)) {
373          p->head = h;
374          return(p);
375      }

405      /* Re-use spanner header. */
406      if (TBL_HEAD_DATA != h->pos &&
407          (TBL_CELL_VERT == p->pos ||
408           TBL_CELL_DVERT == p->pos)) {
409          head_adjust(p, h);
410          p->head = h;
411          return(p);
412      }

414      /* Right-shift headers with a new spanner. */
415      if (TBL_HEAD_DATA == h->pos &&
416          (TBL_CELL_VERT == p->pos ||
417           TBL_CELL_DVERT == p->pos)) {
377          hp = mandoc_calloc(1, sizeof(struct tbl_head));
378          hp->ident = tbl->opts.cols++;
379          hp->vert = vert;
420          hp->prev = h->prev;
421          if (h->prev)
422              h->prev->next = hp;
423          if (h == tbl->first_head)
424              tbl->first_head = hp;
425          h->prev = hp;
426          hp->next = h;
427          head_adjust(p, hp);
428          p->head = hp;
429          return(p);
430      }

432      if (NULL != (h = h->next)) {
433          head_adjust(p, h);
434          p->head = h;
435          return(p);
436      }

438      /* Fall through to default case... */

```

```

439      }

441      hp = mandoc_calloc(1, sizeof(struct tbl_head));
442      hp->ident = tbl->opts.cols++;

381      if (tbl->last_head) {
382          hp->prev = tbl->last_head;
383          tbl->last_head->next = hp;
384      } else
385          tbl->first_head = hp;
386      tbl->last_head = hp;
448      } else
449          tbl->last_head = tbl->first_head = hp;

451      head_adjust(p, hp);
388      p->head = hp;
389      return(p);
390  }

456  static void
457  head_adjust(const struct tbl_cell *cellp, struct tbl_head *head)
458  {
459      if (TBL_CELL_VERT != cellp->pos &&
460          TBL_CELL_DVERT != cellp->pos) {
461          head->pos = TBL_HEAD_DATA;
462          return;
463      }

465      if (TBL_CELL_VERT == cellp->pos)
466          if (TBL_HEAD_DVERT != head->pos)
467              head->pos = TBL_HEAD_VERT;

469      if (TBL_CELL_DVERT == cellp->pos)
470          head->pos = TBL_HEAD_DVERT;
471  }

```

```

*****
9576 Fri Jul 25 15:00:16 2014
new/usr/src/cmd/mandoc/tbl_term.c
Update to 1.12.3.
*****
1 /* $Id: tbl_term.c,v 1.25 2013/05/31 21:37:17 schwarze Exp $ */
1 /* $Id: tbl_term.c,v 1.21 2011/09/20 23:05:49 schwarze Exp $ */
2 /*
3  * Copyright (c) 2009, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011, 2012 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <stdio.h>
24 #include <stdlib.h>
25 #include <string.h>
26
27 #include "mandoc.h"
28 #include "out.h"
29 #include "term.h"
30
31 static size_t term_tbl_len(size_t, void *);
32 static size_t term_tbl_strlen(const char *, void *);
33 static void tbl_char(struct term *p, char, size_t);
34 static void tbl_data(struct term *p, const struct tbl_opts *,
35 static void tbl_data(struct term *p, const struct tbl *,
36 const struct tbl_dat *,
37 const struct roffcol *);
37 static size_t tbl_rulewidth(struct term *p, const struct tbl_head *);
38 static void tbl_hframe(struct term *p, const struct tbl_span *, int);
39 static void tbl_literal(struct term *p, const struct tbl_dat *,
40 const struct roffcol *);
41 static void tbl_number(struct term *p, const struct tbl_opts *,
41 static void tbl_number(struct term *p, const struct tbl *,
42 const struct tbl_dat *,
43 const struct roffcol *);
44 static void tbl_hrule(struct term *p, const struct tbl_span *);
45 static void tbl_vrule(struct term *p, const struct tbl_head *);
46
47
48 static size_t
49 term_tbl_strlen(const char *p, void *arg)
50 {
51
52 return(term_strlen((const struct term *)arg, p));
53 }
54
55 unchanged portion omitted
56
57 void
58 term_tbl(struct term *p, const struct tbl_span *sp)

```

```

64 {
65     const struct tbl_head *hp;
66     const struct tbl_dat *dp;
67     struct roffcol *col;
68     int spans;
69     size_t rmargin, maxrmargin;
70
71     rmargin = tp->rmargin;
72     maxrmargin = tp->maxrmargin;
73
74     tp->rmargin = tp->maxrmargin = TERM_MAXMARGIN;
75
76     /* Inhibit printing of spaces: we do padding ourselves. */
77
78     tp->flags |= TERMP_NONOSPACE;
79     tp->flags |= TERMP_NOSPACE;
80
81     /*
82      * The first time we're invoked for a given table block,
83      * calculate the table widths and decimal positions.
84      */
85
86     if (TBL_SPAN_FIRST & sp->flags) {
87         term_flushln(tp);
88
89         tp->tbl.len = term_tbl_len;
90         tp->tbl.slen = term_tbl_strlen;
91         tp->tbl.arg = tp;
92
93         tblcalc(&tp->tbl, sp);
94     }
95
96     /* Horizontal frame at the start of boxed tables. */
97
98     if (TBL_SPAN_FIRST & sp->flags) {
99         if (TBL_OPT_DBOX & sp->opts->opts)
100             if (TBL_OPT_DBOX & sp->tbl->opts)
101                 tbl_hframe(tp, sp, 1);
102         if (TBL_OPT_DBOX & sp->opts->opts ||
103             TBL_OPT_BOX & sp->opts->opts)
104             if (TBL_OPT_DBOX & sp->tbl->opts ||
105                 TBL_OPT_BOX & sp->tbl->opts)
106                 tbl_hframe(tp, sp, 0);
107     }
108
109     /* Vertical frame at the start of each row. */
110
111     if (TBL_OPT_BOX & sp->opts->opts || TBL_OPT_DBOX & sp->opts->opts)
112         if (TBL_OPT_BOX & sp->tbl->opts || TBL_OPT_DBOX & sp->tbl->opts)
113             term_word(tp, TBL_SPAN_HORIZ == sp->pos ||
114                 TBL_SPAN_DHORIZ == sp->pos ? "+" : "|");
115
116     /*
117      * Now print the actual data itself depending on the span type.
118      * Spanner spans get a horizontal rule; data spanners have their
119      * data printed by matching data to header.
120      */
121
122     switch (sp->pos) {
123     case (TBL_SPAN_HORIZ):
124         /* FALLTHROUGH */
125     case (TBL_SPAN_DHORIZ):
126         tbl_hrule(tp, sp);
127         break;
128     case (TBL_SPAN_DATA):
129         /* Iterate over template headers. */

```



```

126         dp = sp->first;
127         spans = 0;
128         for (hp = sp->head; hp; hp = hp->next) {
129
130             /*
131              * If the current data header is invoked during
132              * a spanner ("spans" > 0), don't emit anything
133              * at all.
134              */
135             switch (hp->pos) {
136                 case (TBL_HEAD_VERT):
137                     /* FALLTHROUGH */
138                     case (TBL_HEAD_DVERT):
139                         if (spans <= 0)
140                             tbl_vruler(tp, hp);
141                         continue;
142                     case (TBL_HEAD_DATA):
143                         break;
144             }
145
146             if (--spans >= 0)
147                 continue;
148
149             /* Separate columns. */
150             /*
151              * All cells get a leading blank, except the
152              * first one and those after double rulers.
153              */
154
155             if (NULL != hp->prev)
156                 tbl_vruler(tp, hp);
157             if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
158                 tbl_char(tp, ASCII_NBRSP, 1);
159
160             col = &tp->tbl.cols[hp->ident];
161             tbl_data(tp, sp->opts, dp, col);
162             tbl_data(tp, sp->tbl, dp, col);
163
164             /* No trailing blanks. */
165
166             if (NULL == hp->next)
167                 break;
168
169             /*
170              * Add another blank between cells,
171              * or two when there is no vertical ruler.
172              */
173             tbl_char(tp, ASCII_NBRSP,
174                 TBL_HEAD_VERT == hp->next->pos ||
175                 TBL_HEAD_DVERT == hp->next->pos ? 1 : 2);
176
177             /*
178              * Go to the next data cell and assign the
179              * number of subsequent spans, if applicable.
180              */
181
182             if (dp) {
183                 spans = dp->spans;
184                 dp = dp->next;
185             }
186             break;
187         }
188
189         /* Vertical frame at the end of each row. */

```

```

162         if (TBL_OPT_BOX & sp->opts->opts || TBL_OPT_DBOX & sp->opts->opts)
163             if (TBL_OPT_BOX & sp->tbl->opts || TBL_OPT_DBOX & sp->tbl->opts)
164                 term_word(tp, TBL_SPAN_HORIZ == sp->pos ||
165                     TBL_SPAN_DHORIZ == sp->pos ? "+" : " |");
166         term_flushln(tp);
167
168         /*
169          * If we're the last row, clean up after ourselves: clear the
170          * existing table configuration and set it to NULL.
171          */
172         if (TBL_SPAN_LAST & sp->flags) {
173             if (TBL_OPT_DBOX & sp->opts->opts ||
174                 TBL_OPT_BOX & sp->opts->opts) {
175                 if (TBL_OPT_DBOX & sp->tbl->opts ||
176                     TBL_OPT_BOX & sp->tbl->opts)
177                     tbl_hframe(tp, sp, 0);
178                 tp->skipvsp = 1;
179             }
180             if (TBL_OPT_DBOX & sp->opts->opts) {
181                 if (TBL_OPT_DBOX & sp->tbl->opts)
182                     tbl_hframe(tp, sp, 1);
183                 tp->skipvsp = 2;
184             }
185             assert(tp->tbl.cols);
186             free(tp->tbl.cols);
187             tp->tbl.cols = NULL;
188         }
189
190         tp->flags &= ~TERMP_NONOSPACE;
191         tp->rmargin = rmargin;
192         tp->maxrmargin = maxrmargin;
193     }
194
195     /*
196     * Horizontal rules extend across the entire table.
197     * Calculate the width by iterating over columns.
198     */
199     static size_t
200     tbl_rulewidth(struct term *tp, const struct tbl_head *hp)
201     {
202         size_t width;
203
204         width = tp->tbl.cols[hp->ident].width;
205
206         if (TBL_HEAD_DATA == hp->pos) {
207             /* Account for leading blanks. */
208             if (hp->prev)
209                 width += 2 - hp->vert;
210
211             /* Account for trailing blank. */
212             if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
213                 width++;
214
215             /* Account for trailing blanks. */
216             width++;
217             if (hp->next &&
218                 TBL_HEAD_VERT != hp->next->pos &&
219                 TBL_HEAD_DVERT != hp->next->pos)
220                 width++;
221         }
222         return(width);
223     }

```

```

214 /*
215  * Rules inside the table can be single or double
216  * and have crossings with vertical rules marked with pluses.
217  */
218 static void
219 tbl_hrule(struct term *tp, const struct tbl_span *sp)
220 {
221     const struct tbl_head *hp;
222     char c;
223
224     c = '-';
225     if (TBL_SPAN_DHORIZ == sp->pos)
226         c = '=';
227
228     for (hp = sp->head; hp; hp = hp->next) {
229         if (hp->prev && hp->vert)
230             tbl_char(tp, '+', hp->vert);
231         tbl_char(tp, c, tbl_rulewidth(tp, hp));
232     }
233     for (hp = sp->head; hp; hp = hp->next)
234         tbl_char(tp,
235                 TBL_HEAD_DATA == hp->pos ? c : '+',
236                 tbl_rulewidth(tp, hp));
237 }
238
239 /*
240  * Rules above and below the table are always single
241  * and have an additional plus at the beginning and end.
242  * For double frames, this function is called twice,
243  * and the outer one does not have crossings.
244  */
245 static void
246 tbl_hframe(struct term *tp, const struct tbl_span *sp, int outer)
247 {
248     const struct tbl_head *hp;
249
250     term_word(tp, "+");
251     for (hp = sp->head; hp; hp = hp->next) {
252         if (hp->prev && hp->vert)
253             tbl_char(tp, (outer ? '-' : '+'), hp->vert);
254         tbl_char(tp, '-', tbl_rulewidth(tp, hp));
255     }
256     for (hp = sp->head; hp; hp = hp->next)
257         tbl_char(tp,
258                 outer || TBL_HEAD_DATA == hp->pos ? '-' : '+',
259                 tbl_rulewidth(tp, hp));
260     term_word(tp, "+");
261     term_flushln(tp);
262 }
263
264 static void
265 tbl_data(struct term *tp, const struct tbl_opts *opts,
266          const struct tbl_dat *dp,
267          const struct tbl_dat *tbl,
268          const struct tbl_dat *dp,
269          const struct roffcol *col)
270 {
271     if (NULL == dp) {
272         tbl_char(tp, ASCII_NBRSP, col->width);
273         return;
274     }
275     assert(dp->layout);
276
277     switch (dp->pos) {
278     case (TBL_DATA_NONE):
279         tbl_char(tp, ASCII_NBRSP, col->width);

```

```

271         return;
272     case (TBL_DATA_HORIZ):
273         /* FALLTHROUGH */
274     case (TBL_DATA_NHORIZ):
275         tbl_char(tp, '-', col->width);
276         return;
277     case (TBL_DATA_NDHORIZ):
278         /* FALLTHROUGH */
279     case (TBL_DATA_DHORIZ):
280         tbl_char(tp, '=', col->width);
281         return;
282     default:
283         break;
284 }
285
286 switch (dp->layout->pos) {
287 case (TBL_CELL_HORIZ):
288     tbl_char(tp, '-', col->width);
289     break;
290 case (TBL_CELL_DHORIZ):
291     tbl_char(tp, '=', col->width);
292     break;
293 case (TBL_CELL_LONG):
294     /* FALLTHROUGH */
295 case (TBL_CELL_CENTRE):
296     /* FALLTHROUGH */
297 case (TBL_CELL_LEFT):
298     /* FALLTHROUGH */
299 case (TBL_CELL_RIGHT):
300     tbl_literal(tp, dp, col);
301     break;
302 case (TBL_CELL_NUMBER):
303     tbl_number(tp, opts, dp, col);
304     tbl_number(tp, tbl, dp, col);
305     break;
306 case (TBL_CELL_DOWN):
307     tbl_char(tp, ASCII_NBRSP, col->width);
308     break;
309 default:
310     abort();
311     /* NOTREACHED */
312 }
313
314 static void
315 tbl_vrule(struct term *tp, const struct tbl_head *hp)
316 {
317     tbl_char(tp, ASCII_NBRSP, 1);
318     if (0 < hp->vert)
319         tbl_char(tp, '|', hp->vert);
320     if (2 > hp->vert)
321         tbl_char(tp, ASCII_NBRSP, 2 - hp->vert);
322     switch (hp->pos) {
323     case (TBL_HEAD_VERT):
324         term_word(tp, "|");
325         break;
326     case (TBL_HEAD_DVERT):
327         term_word(tp, "||");
328         break;
329     default:
330         break;
331     }
332 }

```

unchanged portion omitted

```

340 static void
341 tbl_literal(struct term *tp, const struct tbl_dat *dp,
342             const struct roffcol *col)
343 {
344     struct tbl_head *hp;
345     size_t width, len, padl, padr;
346     int spans;
347     size_t len, padl, padr;
348
349     assert(dp->string);
350     len = term_strlen(tp, dp->string);
351
352     hp = dp->layout->head->next;
353     width = col->width;
354     for (spans = dp->spans; spans--; hp = hp->next)
355         width += tp->tbl.cols[hp->ident].width + 3;
356
357     padr = width > len ? width - len : 0;
358     padr = col->width > len ? col->width - len : 0;
359     padl = 0;
360
361     switch (dp->layout->pos) {
362     case (TBL_CELL_LONG):
363         padl = term_len(tp, 1);
364         padr = padr > padl ? padr - padl : 0;
365         break;
366     case (TBL_CELL_CENTRE):
367         if (2 > padr)
368             break;
369         padl = padr / 2;
370         padr -= padl;
371         break;
372     case (TBL_CELL_RIGHT):
373         padl = padr;
374         padr = 0;
375         break;
376     default:
377         break;
378     }
379
380     tbl_char(tp, ASCII_NBRSP, padl);
381     term_word(tp, dp->string);
382     tbl_char(tp, ASCII_NBRSP, padr);
383 }
384
385 static void
386 tbl_number(struct term *tp, const struct tbl_opts *opts,
387            const struct tbl_dat *dp,
388            const struct roffcol *col)
389 {
390     char *cp;
391     char buf[2];
392     size_t sz, psz, ssz, d, padl;
393     int i;
394
395     /*
396     * See calc_data_number(). Left-pad by taking the offset of our
397     * and the maximum decimal; right-pad by the remaining amount.
398     */
399
400     assert(dp->string);
401
402     sz = term_strlen(tp, dp->string);
403
404     buf[0] = opts->decimal;

```

```

422     buf[0] = tbl->decimal;
423     buf[1] = '\0';
424
425     psz = term_strlen(tp, buf);
426
427     if (NULL != (cp = strchr(dp->string, opts->decimal))) {
428     if (NULL != (cp = strchr(dp->string, tbl->decimal))) {
429         buf[1] = '\0';
430         for (ssz = 0, i = 0; cp != &dp->string[i]; i++) {
431             buf[0] = dp->string[i];
432             ssz += term_strlen(tp, buf);
433         }
434         d = ssz + psz;
435     } else
436         d = sz + psz;
437
438     padl = col->decimal - d;
439
440     tbl_char(tp, ASCII_NBRSP, padl);
441     term_word(tp, dp->string);
442     if (col->width > sz + padl)
443         tbl_char(tp, ASCII_NBRSP, col->width - sz - padl);
444 }

```

unchanged portion omitted

```

*****
16799 Fri Jul 25 15:00:17 2014
new/usr/src/cmd/mandoc/term.c
Update to 1.12.3.
*****
1 /* $Id: term.c,v 1.214 2013/12/25 00:39:31 schwarze Exp $ */
1 /* $Id: term.c,v 1.201 2011/09/21 09:57:13 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
4  * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <sys/types.h>
23
24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdint.h>
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <string.h>
30
31 #include "mandoc.h"
32 #include "out.h"
33 #include "term.h"
34 #include "main.h"
35
36 static size_t cond_width(const struct term *p, int, int *);
37 static void adjbuf(struct term *p, size_t);
38 static void adjbuf(struct term *p, int);
39 static void bufferc(struct term *p, char);
40 static void encode(struct term *p, const char *, size_t);
41 static void encodel(struct term *p, int);
42
43 void term_free(struct term *p)
44 {
45     if (p->buf)
46         free(p->buf);
47     if (p->syntab)
48         mchars_free(p->syntab);
49     free(p);
50 }
51
52 unchanged_portion_omitted
53
54 /*
55  * Flush a line of text. A "line" is loosely defined as being something
56  * that should be followed by a newline, regardless of whether it's
57  * broken apart by newlines getting there. A line can also be a

```

```

78 * fragment of a columnar list ('Bl -tag' or 'Bl -column'), which does
79 * not have a trailing newline.
80 *
81 * The following flags may be specified:
82 *
83 * - TERMP_NOBREAK: this is the most important and is used when making
84 * columns. In short: don't print a newline and instead expect the
85 * next call to do the padding up to the start of the next column.
86 * p->trailspace may be set to 0, 1, or 2, depending on how many
87 * space characters are required at the end of the column.
88 *
89 * - TERMP_TWOSPACE: make sure there is room for at least two space
90 * characters of padding. Otherwise, rather break the line.
91 *
92 * - TERMP_DANGLE: don't newline when TERMP_NOBREAK is specified and
93 * the line is overrun, and don't pad-right if it's underrun.
94 *
95 * - TERMP_HANG: like TERMP_DANGLE, but doesn't newline when
96 * overrunning, instead save the position and continue at that point
97 * when the next invocation.
98 *
99 * In-line line breaking:
100 *
101 * If TERMP_NOBREAK is specified and the line overruns the right
102 * margin, it will break and pad-right to the right margin after
103 * writing. If maxrmargin is violated, it will break and continue
104 * writing from the right-margin, which will lead to the above scenario
105 * upon exit. Otherwise, the line will break at the right margin.
106 */
107 void term_flushln(struct term *p)
108 {
109     size_t i; /* current input position in p->buf */
110     int ntab; /* number of tabs to prepend */
111     size_t i; /* current input position in p->buf */
112     size_t vis; /* current visual position on output */
113     size_t vbl; /* number of blanks to prepend to output */
114     size_t vend; /* end of word visual position on output */
115     size_t bp; /* visual right border position */
116     size_t dv; /* temporary for visual pos calculations */
117     size_t j; /* temporary loop index for p->buf */
118     size_t jhy; /* last hyph before overflow w/r/t j */
119     int j; /* temporary loop index for p->buf */
120     int jhy; /* last hyph before overflow w/r/t j */
121     size_t maxvis; /* output position of visible boundary */
122     size_t mmax; /* used in calculating bp */
123
124     /*
125      * First, establish the maximum columns of "visible" content.
126      * This is usually the difference between the right-margin and
127      * an indentation, but can be, for tagged lists or columns, a
128      * small set of values.
129      *
130      * The following unsigned-signed subtractions look strange,
131      * but they are actually correct. If the int p->overstep
132      * is negative, it gets sign extended. Subtracting that
133      * very large size_t effectively adds a small number to dv.
134      */
135     assert (p->rmargin >= p->offset);
136     dv = p->rmargin - p->offset;
137     maxvis = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;
138     dv = p->maxrmargin - p->offset;
139     mmax = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;
140
141     bp = TERMP_NOBREAK & p->flags ? mmax : maxvis;

```

```

138 /*
139  * Calculate the required amount of padding.
140  */
141 vbl = p->offset + p->overstep > p->viscol ?
142     p->offset + p->overstep - p->viscol : 0;

144 vis = vend = 0;
145 i = 0;

147 while (i < p->col) {
148     /*
149     * Handle literal tab characters: collapse all
150     * subsequent tabs into a single huge set of spaces.
151     */
152     ntab = 0;
153     while (i < p->col && '\t' == p->buf[i]) {
154         vend = (vis / p->tabwidth + 1) * p->tabwidth;
155         vbl += vend - vis;
156         vis = vend;
157         ntab++;
158         i++;
159     }

161     /*
162     * Count up visible word characters. Control sequences
163     * (starting with the CSI) aren't counted. A space
164     * generates a non-printing word, which is valid (the
165     * space is printed according to regular spacing rules).
166     */

168     for (j = i, jhy = 0; j < p->col; j++) {
169         if (' ' == p->buf[j] || '\t' == p->buf[j])
170             if ((j && ' ' == p->buf[j]) || '\t' == p->buf[j])
171                 break;

172         /* Back over the the last printed character. */
173         if (8 == p->buf[j]) {
174             assert(j);
175             vend -= (*p->width)(p, p->buf[j - 1]);
176             continue;
177         }

179         /* Regular word. */
180         /* Break at the hyphen point if we overrun. */
181         if (vend > vis && vend < bp &&
182             ASCII_HYPH == p->buf[j])
183             jhy = j;

185         vend += (*p->width)(p, p->buf[j]);
186     }

188     /*
189     * Find out whether we would exceed the right margin.
190     * If so, break to the next line.
191     */
192     if (vend > bp && 0 == jhy && vis > 0) {
193         vend -= vis;
194         (*p->endline)(p);
195         p->viscol = 0;
196         if (TERMP_NOBREAK & p->flags) {
197             vbl = p->rmargin;
198             vend += p->rmargin - p->offset;
199         } else
200             vbl = p->offset;

202     /* use pending tabs on the new line */

```

```

194     /* Remove the p->overstep width. */

204     if (0 < ntab)
205         vbl += ntab * p->tabwidth;

207     /*
208     * Remove the p->overstep width.
209     * Again, if p->overstep is negative,
210     * sign extension does the right thing.
211     */

213     bp += (size_t)p->overstep;
214     p->overstep = 0;
215 }

217 /* Write out the [remaining] word. */
218 for (; i < p->col; i++) {
219     if (vend > bp && jhy > 0 && i > jhy)
220         break;
221     if ('\t' == p->buf[i])
222         break;
223     if (' ' == p->buf[i]) {
224         j = i;
225         while (' ' == p->buf[i])
226             i++;
227         dv = (i - j) * (*p->width)(p, ' ');
228         dv = (size_t)(i - j) * (*p->width)(p, ' ');
229         vbl += dv;
230         vend += dv;
231         break;
232     }
233     if (ASCII_NBRSP == p->buf[i]) {
234         vbl += (*p->width)(p, ' ');
235         continue;
236     }

237     /*
238     * Now we definitely know there will be
239     * printable characters to output,
240     * so write preceding white space now.
241     */
242     if (vbl) {
243         (*p->advance)(p, vbl);
244         p->viscol += vbl;
245         vbl = 0;
246     }

248     if (ASCII_HYPH == p->buf[i]) {
249         (*p->letter)(p, '-');
250         p->viscol += (*p->width)(p, '-');
251         continue;
252     }

254     (*p->letter)(p, p->buf[i]);
255     if (8 == p->buf[i])
256         p->viscol -= (*p->width)(p, p->buf[i-1]);
257     else
258         p->viscol += (*p->width)(p, p->buf[i]);
259 }
260 vis = vend;
261 }

263 /*
264 * If there was trailing white space, it was not printed;
265 * so reset the cursor position accordingly.
266 */

```

```

267     if (vis)
268         vis -= vbl;

270     p->col = 0;
271     p->overstep = 0;

273     if ( ! (TERMP_NOBREAK & p->flags) ) {
274         p->viscol = 0;
275         (*p->endline)(p);
276         return;
277     }

279     if (TERMP_HANG & p->flags) {
280         p->overstep = (int)(vis - maxvis +
281             p->trailspace * (*p->width)(p, ' '));
282         /* We need one blank after the tag. */
283         p->overstep = (int)(vis - maxvis + (*p->width)(p, ' '));
284     }

286     /*
287     * Behave exactly the same way as groff:
288     * If we have overstepped the margin, temporarily move
289     * it to the right and flag the rest of the line to be
290     * shorter.
291     * If there is a request to keep the columns together,
292     * allow negative overstep when the column is not full.
293     * If we landed right at the margin, be happy.
294     * If we are one step before the margin, temporarily
295     * move it one step LEFT and flag the rest of the line
296     * to be longer.
297     */
298     if (p->trailspace && p->overstep < 0)
299         if (p->overstep < -1)
300             p->overstep = 0;
301     return;

303     } else if (TERMP_DANGLE & p->flags)
304         return;

306     /* If the column was overrun, break the line. */
307     if (maxvis < vis + p->trailspace * (*p->width)(p, ' ')) {
308         if (maxvis <= vis +
309             ((TERMP_TWOSPACE & p->flags) ? (*p->width)(p, ' ') : 0)) {
310             (*p->endline)(p);
311             p->viscol = 0;
312         }
313     }
314 }
315 unchanged_portion_omitted

```

```

320 /*
321 * Asserts a vertical space (a full, empty line-break between lines).
322 * Note that if used twice, this will cause two blank spaces and so on.
323 * All data in the output buffer is flushed prior to the newline
324 * assertion.
325 */
326 void
327 term_vspace(struct term *p)
328 {
329     term_newln(p);
330     p->viscol = 0;
331     if (0 < p->skipvsp)
332         p->skipvsp--;
333     else
334         (*p->endline)(p);
335 }
336 unchanged_portion_omitted

```

```

384 void
385 term_fontpopq(struct term *p, const void *key)
386 {
387     while (p->fonti >= 0 && key < (void *) (p->fontq + p->fonti))
388         while (p->fonti >= 0 && key != &p->fontq[p->fonti])
389             p->fonti--;
390     assert(p->fonti >= 0);
391 }
392 unchanged_portion_omitted
393
394 /*
395 * Handle pwords, partial words, which may be either a single word or a
396 * phrase that cannot be broken down (such as a literal string). This
397 * handles word styling.
398 */
399 void
400 term_word(struct term *p, const char *word)
401 {
402     const char     nbrsp[2] = { ASCII_NBRSP, 0 };
403     const char     *seq, *cp;
404     char           c;
405     int            sz, uc;
406     size_t         ssz;
407     enum mandoc_esc esc;
408
409     if ( ! (TERMP_NOSPACE & p->flags) ) {
410         if ( ! (TERMP_KEEP & p->flags) ) {
411             if (TERMP_PREKEEP & p->flags)
412                 p->flags |= TERMP_KEEP;
413             bufferc(p, ' ');
414             if (TERMP_SENTENCE & p->flags)
415                 bufferc(p, ' ');
416         } else
417             bufferc(p, ASCII_NBRSP);
418     }
419     if (TERMP_PREKEEP & p->flags)
420         p->flags |= TERMP_KEEP;
421
422     if ( ! (p->flags & TERMP_NONOSPACE) )
423         p->flags &= ~TERMP_NOSPACE;
424     else
425         p->flags |= TERMP_NOSPACE;
426
427     p->flags &= ~TERMP_SENTENCE;
428     p->flags &= ~(TERMP_SENTENCE | TERMP_IGNDELIM);
429
430     while ('0' != *word) {
431         if ('\' != *word) {
432             if (TERMP_SKIPCHAR & p->flags) {
433                 p->flags &= ~TERMP_SKIPCHAR;
434                 word++;
435                 continue;
436             }
437             if (TERMP_NBRWORD & p->flags) {
438                 if (' ' == *word) {
439                     encode(p, nbrsp, 1);
440                     word++;
441                     continue;
442                 }
443                 ssz = strcspn(word, "\\ ");
444             } else
445                 ssz = strcspn(word, "\\");
446             if ((ssz = strcspn(word, "\\")) > 0)

```

```

451         encode(p, word, ssz);
452         word += (int)ssz;
453         if ('\0' != *word)
454             continue;
455     }
456     word++;
457     esc = mandoc_escape(&word, &seq, &sz);
458     if (ESCAPE_ERROR == esc)
459         break;
460
461     if (TERMENC_ASCII != p->enc)
462         switch (esc) {
463         case (ESCAPE_UNICODE):
464             uc = mchars_num2uc(seq + 1, sz - 1);
465             if ('\0' == uc)
466                 break;
467             encode1(p, uc);
468             continue;
469         case (ESCAPE_SPECIAL):
470             uc = mchars_spec2cp(p->syntab, seq, sz);
471             if (uc <= 0)
472                 break;
473             encode1(p, uc);
474             continue;
475         default:
476             break;
477         }
478
479     switch (esc) {
480     case (ESCAPE_UNICODE):
481         encode1(p, '?');
482         break;
483     case (ESCAPE_NUMBERED):
484         c = mchars_num2char(seq, sz);
485         if ('\0' != c)
486             encode(p, &c, 1);
487         break;
488     case (ESCAPE_SPECIAL):
489         cp = mchars_spec2str(p->syntab, seq, sz, &ssz);
490         if (NULL != cp)
491             encode(p, cp, ssz);
492         else if (1 == ssz)
493             encode(p, seq, sz);
494         break;
495     case (ESCAPE_FONTBOLD):
496         term_fontrepl(p, TERMFONT_BOLD);
497         break;
498     case (ESCAPE_FONTITALIC):
499         term_fontrepl(p, TERMFONT_UNDER);
500         break;
501     case (ESCAPE_FONTBI):
502         term_fontrepl(p, TERMFONT_BI);
503         break;
504     case (ESCAPE_FONT):
505         /* FALLTHROUGH */
506     case (ESCAPE_FONTRoman):
507         term_fontrepl(p, TERMFONT_NONE);
508         break;
509     case (ESCAPE_FONTPREV):
510         term_fontlast(p);
511         break;
512     case (ESCAPE_NOSPACE):
513         if (TERMP_SKIPCHAR & p->flags)
514             p->flags &= ~TERMP_SKIPCHAR;

```

```

515         else if ('\0' == *word)
516             if ('\0' == *word)
517                 p->flags |= TERMP_NOSPACE;
518             break;
519         case (ESCAPE_SKIPCHAR):
520             p->flags |= TERMP_SKIPCHAR;
521             break;
522         default:
523             break;
524     }
525     p->flags &= ~TERMP_NBRWORD;
526 }
527
528 static void
529 adjbuf(struct term *p, size_t sz)
530 {
531     if (0 == p->maxcols)
532         p->maxcols = 1024;
533     while (sz >= p->maxcols)
534         p->maxcols <<= 2;
535
536     p->buf = mandoc_realloc(p->buf, sizeof(int) * p->maxcols);
537     p->buf = mandoc_realloc
538         (p->buf, sizeof(int) * (size_t)p->maxcols);
539 }
540
541 unchanged_portion_omitted_
542
543 /*
544  * See encode().
545  * Do this for a single (probably unicode) value.
546  * Does not check for non-decorated glyphs.
547  */
548 static void
549 encode1(struct term *p, int c)
550 {
551     enum termfont f;
552
553     if (TERMP_SKIPCHAR & p->flags) {
554         p->flags &= ~TERMP_SKIPCHAR;
555         return;
556     }
557     if (p->col + 4 >= p->maxcols)
558         adjbuf(p, p->col + 4);
559
560     if (p->col + 6 >= p->maxcols)
561         adjbuf(p, p->col + 6);
562
563     f = term_fonttop(p);
564
565     if (TERMFONT_UNDER == f || TERMFONT_BI == f) {
566         if (TERMFONT_NONE == f) {
567             p->buf[p->col++] = c;
568             return;
569         } else if (TERMFONT_UNDER == f) {
570             p->buf[p->col++] = '_';
571             p->buf[p->col++] = 8;
572         }
573     }
574     if (TERMFONT_BOLD == f || TERMFONT_BI == f) {
575         if (ASCII_HYPH == c)
576             p->buf[p->col++] = '-';
577         else
578             p->buf[p->col++] = c;

```

```

579         p->buf[p->col++] = 8;
580     }
581     p->buf[p->col++] = c;
582 }

584 static void
585 encode(struct term *p, const char *word, size_t sz)
586 {
587     size_t      i;
588     enum termfont f;
589     int         i, len;

590     if (TERMP_SKIPCHAR & p->flags) {
591         p->flags &= ~TERMP_SKIPCHAR;
592         return;
593     }
594     /* LINTED */
595     len = sz;

596     /*
597      * Encode and buffer a string of characters.  If the current
598      * font mode is unset, buffer directly, else encode then buffer
599      * character by character.
600      */

601     if (TERMFONT_NONE == term_fonttop(p)) {
602         if (p->col + sz >= p->maxcols)
603             adjbuf(p, p->col + sz);
604         for (i = 0; i < sz; i++)
605             if (TERMFONT_NONE == (f = term_fonttop(p))) {
606                 if (p->col + len >= p->maxcols)
607                     adjbuf(p, p->col + len);
608                 for (i = 0; i < len; i++)
609                     p->buf[p->col++] = word[i];
610                 return;
611             }
612     }

613     /* Pre-buffer, assuming worst-case. */

614     if (p->col + 1 + (sz * 5) >= p->maxcols)
615         adjbuf(p, p->col + 1 + (sz * 5));
616     if (p->col + 1 + (len * 3) >= p->maxcols)
617         adjbuf(p, p->col + 1 + (len * 3));

618     for (i = 0; i < sz; i++) {
619         if (ASCII_HYPH == word[i] ||
620             isgraph((unsigned char)word[i]))
621             encodel(p, word[i]);
622         for (i = 0; i < len; i++) {
623             if (ASCII_HYPH != word[i] &&
624                 ! isgraph((unsigned char)word[i])) {
625                 p->buf[p->col++] = word[i];
626                 continue;
627             }

628             if (TERMFONT_UNDER == f)
629                 p->buf[p->col++] = '_';
630             else if (ASCII_HYPH == word[i])
631                 p->buf[p->col++] = '-';
632             else
633                 p->buf[p->col++] = word[i];
634         }

635     }

636     p->buf[p->col++] = 8;
637     p->buf[p->col++] = word[i];
638 }

```

```

620 }
        unchanged_portion_omitted_

629 static size_t
630 cond_width(const struct term *p, int c, int *skip)
631 {
632     if (*skip) {
633         (*skip) = 0;
634         return(0);
635     } else
636         return((*p->width)(p, c));
637 }

638 size_t
639 term_strlen(const struct term *p, const char *cp)
640 {
641     size_t      sz, rsz, i;
642     int         ssz, skip, c;
643     int         sseq, c;
644     const char  *seq, *rhs;
645     enum mandoc_esc esc;
646     static const char rej[] = { '\\', ASCII_HYPH, ASCII_NBRSP, '\0' };

647     /*
648      * Account for escaped sequences within string length
649      * calculations.  This follows the logic in term_word() as we
650      * must calculate the width of produced strings.
651      */

652     sz = 0;
653     skip = 0;
654     while ('\0' != *cp) {
655         rsz = strcspn(cp, rej);
656         for (i = 0; i < rsz; i++)
657             sz += cond_width(p, *cp++, &skip);
658             sz += (*p->width)(p, *cp++);

659         c = 0;
660         switch (*cp) {
661             case '\\':
662                 cp++;
663                 esc = mandoc_escape(&cp, &seq, &ssz);
664                 if (ESCAPE_ERROR == esc)
665                     return(sz);

666                 if (TERMENC_ASCII != p->enc)
667                     switch (esc) {
668                         case (ESCAPE_UNICODE):
669                             c = mchars_num2uc
670                                 (seq + 1, ssz - 1);
671                             if ('\0' == c)
672                                 break;
673                             sz += cond_width(p, c, &skip);
674                             sz += (*p->width)(p, c);
675                             continue;
676                         case (ESCAPE_SPECIAL):
677                             c = mchars_spec2cp
678                                 (p->symtab, seq, ssz);
679                             if (c <= 0)
680                                 break;
681                             sz += cond_width(p, c, &skip);
682                             sz += (*p->width)(p, c);
683                             continue;
684                         default:
685                             break;
686                     }
687     }

```



```

688         }
690         rhs = NULL;
692         switch (esc) {
693         case (ESCAPE_UNICODE):
694             sz += cond_width(p, '?', &skip);
695             sz += (*p->width)(p, '?');
696             break;
697         case (ESCAPE_NUMBERED):
698             c = mchars_num2char(seq, ssz);
699             if ('\0' != c)
700                 sz += cond_width(p, c, &skip);
701             sz += (*p->width)(p, c);
702             break;
703         case (ESCAPE_SPECIAL):
704             rhs = mchars_spec2str
705                 (p->syntab, seq, ssz, &rsz);
706
707             if (ssz != 1 || rhs)
708                 break;
709
710             rhs = seq;
711             rsz = ssz;
712             break;
713         case (ESCAPE_SKIPCHAR):
714             skip = 1;
715             break;
716         default:
717             break;
718         }
719
720         if (NULL == rhs)
721             break;
722
723         if (skip) {
724             skip = 0;
725             break;
726         }
727
728         for (i = 0; i < rsz; i++)
729             sz += (*p->width)(p, *rhs++);
730         break;
731     case (ASCII_NBRSP):
732         sz += cond_width(p, ' ', &skip);
733         sz += (*p->width)(p, ' ');
734         cp++;
735         break;
736     case (ASCII_HYPH):
737         sz += cond_width(p, '-', &skip);
738         sz += (*p->width)(p, '-');
739         cp++;
740         break;
741     default:
742         break;
743     }
744 }
745
746     return(sz);
747 }

```

unchanged\_portion\_omitted

```

*****
4745 Fri Jul 25 15:00:17 2014
new/usr/src/cmd/mandoc/term.h
Update to 1.12.3.
*****
1 /* $Id: term.h,v 1.97 2013/12/25 00:39:31 schwarze Exp $ */
1 /* $Id: term.h,v 1.90 2011/12/04 23:10:52 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifndef TERM_H
19 #define TERM_H

21 __BEGIN_DECLS

23 struct termp;

25 enum termenc {
26     TERMENC_ASCII,
27     TERMENC_LOCALE,
28     TERMENC_UTF8
29 };
    unchanged_portion_omitted

37 enum termfont {
38     TERMFONT_NONE = 0,
39     TERMFONT_BOLD,
40     TERMFONT_UNDER,
41     TERMFONT_BI,
42     TERMFONT_MAX
43 };
    unchanged_portion_omitted

54 struct termp {
55     enum termtype type;
56     struct rofftbl tbl; /* table configuration */
57     int mdocstyle; /* imitate mdoc(7) output */
58     size_t defindent; /* Default indent for text. */
59     size_t defrmargin; /* Right margin of the device. */
60     size_t rmargin; /* Current right margin. */
61     size_t maxrmargin; /* Max right margin. */
62     size_t maxcols; /* Max size of buf. */
63     int maxcols; /* Max size of buf. */
64     size_t offset; /* Margin offset. */
65     size_t tabwidth; /* Distance of tab positions. */
66     size_t col; /* Bytes in buf. */
67     int col; /* Bytes in buf. */
68     size_t viscol; /* Chars on current line. */
69     size_t trailspace; /* See term_flushln(). */
70     int overstep; /* See term_flushln(). */
71     int skipvsp; /* Vertical space to skip. */
72     int flags;
73 #define TERMP_SENTENCE (1 << 1) /* Space before a sentence. */

```

```

72 #define TERMP_NOSPACE (1 << 2) /* No space before words. */
73 #define TERMP_NONOSPACE (1 << 3) /* No space (no autounset). */
74 #define TERMP_NBRWORD (1 << 4) /* Make next word nonbreaking. */
75 #define TERMP_KEEP (1 << 5) /* Keep words together. */
76 #define TERMP_PREKEEP (1 << 6) /* ...starting with the next one. */
77 #define TERMP_SKIPCHAR (1 << 7) /* Skip the next character. */
78 #define TERMP_NOBREAK (1 << 8) /* See term_flushln(). */
79 #define TERMP_DANGLE (1 << 9) /* See term_flushln(). */
80 #define TERMP_HANG (1 << 10) /* See term_flushln(). */
81 #define TERMP_NOBREAK (1 << 4) /* See term_flushln(). */
82 #define TERMP_IGNDELIM (1 << 6) /* Delims like regulars. */
83 #define TERMP_NONOSPACE (1 << 7) /* No space (no autounset). */
84 #define TERMP_DANGLE (1 << 8) /* See term_flushln(). */
85 #define TERMP_HANG (1 << 9) /* See term_flushln(). */
86 #define TERMP_TWOSPACE (1 << 10) /* See term_flushln(). */
87 #define TERMP_NOSPLIT (1 << 11) /* See term_an_pre/post(). */
88 #define TERMP_SPLIT (1 << 12) /* See term_an_pre/post(). */
89 #define TERMP_ANPREC (1 << 13) /* See term_an_pre(). */
90 #define TERMP_KEEP (1 << 14) /* Keep words together. */
91 #define TERMP_PREKEEP (1 << 15) /* ...starting with the next one. */
92 int *buf; /* Output buffer. */
93 enum termenc enc; /* Type of encoding. */
94 struct mchars *symtab; /* Encoded-symbol table. */
95 enum termfont fontl; /* Last font set. */
96 enum termfont fontq[10]; /* Symmetric fonts. */
97 int fonti; /* Index of font stack. */
98 term_margin headf; /* invoked to print head */
99 term_margin footf; /* invoked to print foot */
100 void (*letter)(struct termp *, int);
101 void (*begin)(struct termp *);
102 void (*end)(struct termp *);
103 void (*newline)(struct termp *);
104 void (*advance)(struct termp *, size_t);
105 size_t (*width)(const struct termp *, int);
106 double (*hspan)(const struct termp *,
107     const struct roffsu *);
108 const void *argf; /* arg for headf/footf */
109 struct termp_ps *ps;
110 };
    unchanged_portion_omitted

```

```

*****
5394 Fri Jul 25 15:00:17 2014
new/usr/src/cmd/mandoc/term_ascii.c
Update to 1.12.3.
*****
1 /* $Id: term_ascii.c,v 1.21 2013/06/01 14:27:20 schwarze Exp $ */
1 /* $Id: term_ascii.c,v 1.20 2011/12/04 23:10:52 schwarze Exp $ */
2 /*
3  * Copyright (c) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  *
5  * Permission to use, copy, modify, and distribute this software for any
6  * purpose with or without fee is hereby granted, provided that the above
7  * copyright notice and this permission notice appear in all copies.
8  *
9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
10 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
11 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
12 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
13 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
14 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
15 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
16 */
17 #ifdef HAVE_CONFIG_H
18 #include "config.h"
19 #endif
20
21 #include <sys/types.h>
22
23 #include <assert.h>
24 #ifdef USE_WCHAR
25 #include <locale.h>
26 #endif
27 #include <stdint.h>
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <unistd.h>
31 #ifdef USE_WCHAR
32 #include <wchar.h>
33 #endif
34
35 #include "mandoc.h"
36 #include "out.h"
37 #include "term.h"
38 #include "main.h"
39
40 /*
41  * Sadly, this doesn't seem to be defined on systems even when they
42  * support it. For the time being, remove it and let those compiling
43  * the software decide for themselves what to use.
44  */
45 #if 0
46 #if ! defined(__STDC_ISO_10646__)
47 #undef USE_WCHAR
48 #endif
49 #endif
50
51 static struct termp      *ascii_init(enum termenc, char *);
52 static double            ascii_hspan(const struct termp *,
53                                     const struct roffsu *);
54 static size_t            ascii_width(const struct termp *, int);
55 static void              ascii_advance(struct termp *, size_t);
56 static void              ascii_begin(struct termp *);
57 static void              ascii_end(struct termp *);
58 static void              ascii_endline(struct termp *);
59 static void              ascii_letter(struct termp *, int);

```

```

61 #ifdef USE_WCHAR
62 static void              locale_advance(struct termp *, size_t);
63 static void              locale_endline(struct termp *);
64 static void              locale_letter(struct termp *, int);
65 static size_t            locale_width(const struct termp *, int);
66 #endif
67
68 static struct termp *
69 ascii_init(enum termenc enc, char *outopts)
70 {
71     const char      *toks[4];
72     char            *v;
73     struct termp    *p;
74
75     p = mandoc_calloc(1, sizeof(struct termp));
76     p->enc = enc;
77
78     p->tabwidth = 5;
79     p->defrmargin = 78;
80
81     p->begin = ascii_begin;
82     p->end = ascii_end;
83     p->hspan = ascii_hspan;
84     p->type = TERMTYPE_CHAR;
85
86     p->enc = TERMENC_ASCII;
87     p->advance = ascii_advance;
88     p->endline = ascii_endline;
89     p->letter = ascii_letter;
90     p->width = ascii_width;
91
92 #ifdef USE_WCHAR
93     if (TERMENC_ASCII != enc) {
94         v = TERMENC_LOCALE == enc ?
95             setlocale(LC_ALL, "") :
96             setlocale(LC_CTYPE, "en_US.UTF-8");
97         if (NULL != v && MB_CUR_MAX > 1) {
98             p->enc = enc;
99             p->advance = locale_advance;
100            p->endline = locale_endline;
101            p->letter = locale_letter;
102            p->width = locale_width;
103        }
104     }
105 #endif
106
107     toks[0] = "indent";
108     toks[1] = "width";
109     toks[2] = "mdoc";
110     toks[3] = NULL;
111
112     while (outopts && *outopts)
113         switch (getsubopt(&outopts, UNCONST(toks), &v)) {
114             case (0):
115                 p->defindent = (size_t)atoi(v);
116                 break;
117             case (1):
118                 p->defrmargin = (size_t)atoi(v);
119                 break;
120             case (2):
121                 /*
122                  * Temporary, undocumented mode
123                  * to imitate mdoc(7) output style.
124                  */
125                 p->mdocstyle = 1;

```

`new/usr/src/cmd/mandoc/term_ascii.c`

3

```
125         p->defindent = 5;
126         break;
127     default:
128         break;
129     }

131     /* Enforce a lower boundary. */
132     if (p->defmargin < 58)
133         p->defmargin = 58;

135     return(p);
136 }
unchanged_portion_omitted
```

```

*****
6575 Fri Jul 25 15:00:17 2014
new/usr/src/cmd/mandoc/tree.c
Update to 1.12.3.
*****
1 /* $Id: tree.c,v 1.50 2013/12/24 19:11:46 schwarze Exp $ */
1 /* $Id: tree.c,v 1.47 2011/09/18 14:14:15 schwarze Exp $ */
2 /*
3  * Copyright (c) 2008, 2009, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
4  * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
5  *
6  * Permission to use, copy, modify, and distribute this software for any
7  * purpose with or without fee is hereby granted, provided that the above
8  * copyright notice and this permission notice appear in all copies.
9  *
10 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17 */
18 #ifdef HAVE_CONFIG_H
19 #include "config.h"
20 #endif
21
22 #include <assert.h>
23 #include <limits.h>
24 #include <stdio.h>
25 #include <stdlib.h>
26 #include <time.h>
27
28 #include "mandoc.h"
29 #include "mdoc.h"
30 #include "man.h"
31 #include "main.h"
32
33 static void print_box(const struct eqn_box *, int);
34 static void print_man(const struct man_node *, int);
35 static void print_mdoc(const struct mdoc_node *, int);
36 static void print_span(const struct tbl_span *, int);
37
38
39 /* ARGSUSED */
40 void
41 tree_mdoc(void *arg, const struct mdoc *mdoc)
42 {
43
44     print_mdoc(mdoc_node(mdoc), 0);
45 }
46
47 unchanged_portion_omitted
48
49
50
51 static void
52 print_mdoc(const struct mdoc_node *n, int indent)
53 {
54     const char *p, *t;
55     int i, j;
56     size_t argc;
57     size_t argsz;
58     char **params;
59     struct mdoc_argv *argv;
60
61     argv = NULL;
62     argc = 0;

```

```

66     argc = sz = 0;
67     params = NULL;
68     t = p = NULL;
69
70     switch (n->type) {
71     case (MDOC_ROOT):
72         t = "root";
73         break;
74     case (MDOC_BLOCK):
75         t = "block";
76         break;
77     case (MDOC_HEAD):
78         t = "block-head";
79         break;
80     case (MDOC_BODY):
81         if (n->end)
82             t = "body-end";
83         else
84             t = "block-body";
85         break;
86     case (MDOC_TAIL):
87         t = "block-tail";
88         break;
89     case (MDOC_ELEM):
90         t = "elem";
91         break;
92     case (MDOC_TEXT):
93         t = "text";
94         break;
95     case (MDOC_TBL):
96         /* FALLTHROUGH */
97     case (MDOC_EQN):
98         break;
99     default:
100         abort();
101         /* NOTREACHED */
102     }
103
104     switch (n->type) {
105     case (MDOC_TEXT):
106         p = n->string;
107         break;
108     case (MDOC_BODY):
109         p = mdoc_macronames[n->tok];
110         break;
111     case (MDOC_HEAD):
112         p = mdoc_macronames[n->tok];
113         break;
114     case (MDOC_TAIL):
115         p = mdoc_macronames[n->tok];
116         break;
117     case (MDOC_ELEM):
118         p = mdoc_macronames[n->tok];
119         if (n->args) {
120             argv = n->args->argv;
121             argsz = n->args->argsz;
122         }
123         break;
124     case (MDOC_BLOCK):
125         p = mdoc_macronames[n->tok];
126         if (n->args) {
127             argv = n->args->argv;
128             argsz = n->args->argsz;
129         }
130         break;
131     case (MDOC_TBL):

```

```

131         /* FALLTHROUGH */
132     case (MDOC_EQN):
133         break;
134     case (MDOC_ROOT):
135         p = "root";
136         break;
137     default:
138         abort();
139         /* NOTREACHED */
140     }

142     if (n->span) {
143         assert(NULL == p && NULL == t);
144         print_span(n->span, indent);
145     } else if (n->eqn) {
146         assert(NULL == p && NULL == t);
147         print_box(n->eqn->root, indent);
148     } else {
149         for (i = 0; i < indent; i++)
150             putchar('\t');

152         printf("%s (%s)", p, t);

154         for (i = 0; i < (int)argc; i++) {
155             printf(" -%s", mdoc_argnames[argv[i].arg]);
156             if (argv[i].sz > 0)
157                 printf(" [");
158             for (j = 0; j < (int)argv[i].sz; j++)
159                 printf(" [%s]", argv[i].value[j]);
160             if (argv[i].sz > 0)
161                 printf(" ]");
162         }

164         putchar(' ');
165         if (MDOC_LINE & n->flags)
166             putchar('*');
167         printf("%d:%d", n->line, n->pos);
168         if (n->lastline != n->line)
169             printf("--%d", n->lastline);
170         putchar('\n');
171         for (i = 0; i < (int)sz; i++)
172             printf(" [%s]", params[i]);

173         printf(" %d:%d\n", n->line, n->pos);

174     if (n->child)
175         print_mdoc(n->child, indent + 1);
176     if (n->next)
177         print_mdoc(n->next, indent);
178 }

```

unchanged\_portion\_omitted