****
1094 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/Makefile.common
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
****

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2012 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

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

preconv_OBJS =  preconv.o

CFLAGS +=       $(CC_VERBOSE)

CPPFLAGS +=     -DHAVE_CONFIG_H -DUSE_WCHAR     \
                -DOSNAME="\"illumos\""
                -DOSNAME="\"illumos\""  \
                -DVERSION="\"1.12.1\""
```

```
**********************************************************
    3308 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/arch.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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  */

 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  */

 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")
 43 LINE("beagle",          "Beagle")
 44 LINE("bebox",           "BeBox")
 45 LINE("cats",            "cats")
 46 LINE("cesfic",          "CESFIC")
 47 LINE("cobalt",          "Cobalt")
 48 LINE("dreamcast",       "Dreamcast")
 49 LINE("emips",           "EMIPS")
 50 LINE("evbarm",          "evbARM")
 51 LINE("evbmips",         "evbMIPS")
 52 LINE("evbppc",          "evbPPC")
 53 LINE("evbsh3",          "evbSH3")
 54 LINE("ews4800mips",     "EWS4800MIPS")
 55 LINE("hp300",           "HP300")
 56 LINE("hp700",           "HP700")
 57 LINE("hpcarm",          "HPCARM")
```



```
 58 LINE("hpcmips",         "HPCMIPS")
 59 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("newsmips",        "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",          "SOCPPC")
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 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/chars.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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

 22 #include <assert.h>
 23 #include <ctype.h>
 24 #include <stdlib.h>
 25 #include <string.h>

 27 #include "mandoc.h"
 28 #include "libmandoc.h"

 30 #define PRINT_HI        126
 31 #define PRINT_LO        32

 33 struct  ln {
 34         struct ln       *next;
 35         const char      *code;
 36         const char      *ascii;
 37         int              unicode;
 38 };

 40 #define LINES_MAX        329
 40 #define LINES_MAX        328

 42 #define CHAR(in, ch, code) \
 43         { NULL, (in), (ch), (code) },

 45 #define CHAR_TBL_START    static struct ln lines[LINES_MAX] = {
 46 #define CHAR_TBL_END      };

 48 #include "chars.in"

 50 struct  mchars {
 51         struct ln       **htab;
 52 };
```

*_____unchanged_portion_omitted_*

```
 65 struct mchars *
 66 mchars_alloc(void)
 67 {
```



```
 68         struct mchars    *tab;
 69         struct ln       **htab;
 70         struct ln        *pp;
 71         int               i, hash;

 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          */

 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 **));

 82         for (i = 0; i < LINES_MAX; i++) {
 83                 hash = (int)lines[i].code[0] - PRINT_LO;

 85                 if (NULL == (pp = htab[hash])) {
 86                         htab[hash] = &lines[i];
 87                         continue;
 88                 }

 90                 for ( ; pp->next; pp = pp->next)
 91                         /* Scan ahead. */ ;
 92                 pp->next = &lines[i];
 93         }

 95         tab->htab = htab;
 96         return(tab);
 97 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   10052 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/chars.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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("\'",                      "\'",           769)
53 CHAR("aa",                      "\'",           769)
54 CHAR("ga",                      "`",            768)
55 CHAR("`",                       "`",            768)
56 CHAR("ab",                      "˘",            774)
57 CHAR("ac",                      "¸",            807)
58 CHAR("ad",                      "\"",           776)
```



```
59 CHAR("ah",                      "v",            711)
60 CHAR("ao",                      "o",            730)
61 CHAR("a~",                      "~",            771)
62 CHAR("ho",                      ",",            808)
63 CHAR("ha",                      "^",            94)
64 CHAR("ti",                      "~",            126)

66 /* Quotes. */
67 CHAR("Bq",                      ",,",           8222)
68 CHAR("bq",                      "‚",            8218)
69 CHAR("lq",                      "“",            8220)
70 CHAR("rq",                      "\'\'",         8221)
71 CHAR("oq",                      "‘",            8216)
72 CHAR("cq",                      "\'",           8217)
73 CHAR("aq",                      "\'",           39)
74 CHAR("dq",                      "\"",           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",                      "|",            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",                      ",-",           9127)
96 CHAR("bracelefttp",            ",-",           9127)
97 CHAR("lk",                      "{",            9128)
98 CHAR("braceleftmid",           "{",            9128)
99 CHAR("lb",                      ",-",           9129)
100 CHAR("braceleftbp",           "˙-",           9129)
101 CHAR("braceleftex",           "|",            9130)
102 CHAR("rt",                     "-.",           9131)
103 CHAR("bracerighttp",          "-.",           9131)
104 CHAR("rk",                     "}",            9132)
105 CHAR("bracerightmid",         "}",            9132)
106 CHAR("rb",                     "-\'",          9133)
107 CHAR("bracerightbp",          "-\'",          9133)
108 CHAR("bracerightex",          "|",            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",                     "|",            915)
120 CHAR("*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",                      "0",            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",                      "0",            977)
166 CHAR("+f",                      "o",            966)
167 CHAR("+p",                      "w",            982)
168 CHAR("+e",                      "e",            1013)
169 CHAR("ts",                      "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",                      "^",            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",                      "^",            8657)
240 CHAR("vA",                      "^=v",          8661)

242 /* Logic. */
243 CHAR("AN",                      "^",            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 /* Mathematicals. */
255 CHAR("pl",                      "+",            43)
256 CHAR("mi",                      "-",            8722)
```

```
257 CHAR("-",                   "-",            45)
258 CHAR("-+",                  "-+",           8723)
259 CHAR("+-",                  "+-",           177)
260 CHAR("t+-",                 "+-",           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",                  "!E",           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("rf",                  "_|",           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
```

```
**********************************************************
    1243 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/config.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
#ifndef MANDOC_CONFIG_H
#define MANDOC_CONFIG_H

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

#include <stdio.h>

#define VERSION "1.12.3"
#define HAVE_STRPTIME
#define HAVE_GETSUBOPT
#define HAVE_STRLCAT
#define HAVE_STRLCPY
#define HAVE_MMAP

#include <sys/types.h>

#if !defined(__BEGIN_DECLS)
#  ifdef __cplusplus
#  define       __BEGIN_DECLS           extern "C" {
#  else
#  define       __BEGIN_DECLS
#  endif
#endif
#if !defined(__END_DECLS)
#  ifdef __cplusplus
#  define       __END_DECLS             }
#  else
#  define       __END_DECLS
#  endif
#endif

#ifndef HAVE_BETOH64
#  if defined(__APPLE__)
#    define betoh64(x) OSSwapBigToHostInt64(x)
#if defined(__APPLE__)
# define htobe32(x) OSSwapHostToBigInt32(x)
# define betoh32(x) OSSwapBigToHostInt32(x)
#    define htobe64(x) OSSwapHostToBigInt64(x)
#  elif defined(__sun)
#    define betoh64(x) BE_64(x)
#    define htobe64(x) BE_64(x)
#  else
# define betoh64(x) OSSwapBigToHostInt64(x)
#elif defined(__linux__)
# define betoh32(x) be32toh(x)
#    define betoh64(x) be64toh(x)
#  endif
#endif

#ifndef HAVE_STRLCAT
extern  size_t    strlcat(char *, const char *, size_t);
#endif
#ifndef HAVE_STRLCPY
extern  size_t    strlcpy(char *, const char *, size_t);
#endif
#ifndef HAVE_GETSUBOPT
extern  int       getsubopt(char **, char * const *, char **);
```



```
extern  char     *suboptarg;
#endif
#ifndef HAVE_FGETLN
extern  char     *fgetln(FILE *, size_t *);
#endif

#endif /* MANDOC_CONFIG_H */
```

```
**********************************************************
   15031 Wed Jul 30 20:55:06 2014
new/usr/src/cmd/mandoc/html.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

22 #include <sys/types.h>

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>

33 #include "mandoc.h"
34 #include "libmandoc.h"
35 #include "out.h"
36 #include "html.h"
37 #include "main.h"

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 };
```

*unchanged_portion_omitted_*

```
223 static void
224 print_metaf(struct html *h, enum mandoc_esc deco)
225 {
226         enum htmlfont    font;

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         }

251         if (h->metaf) {
252                 print_tagq(h, h->metaf);
253                 h->metaf = NULL;
254         }

256         h->metal = h->metac;
257         h->metac = font;

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);
256         if (HTMLFONT_NONE != font)
257                 h->metaf = HTMLFONT_BOLD == font ?
258                         print_otag(h, TAG_B, 0, NULL) :
268                 print_otag(h, TAG_I, 0, NULL);
269                 break;
270         default:
271                 break;
272         }
273 }

275 int
276 html_strlen(const char *cp)
277 {
278         size_t           rsz;
279         int              skip, sz;
265         int              ssz, sz;
266         const char      *seq, *p;

281         /*
282          * Account for escaped sequences within string length
283          * calculations.  This follows the logic in term_strlen() as we
284          * must calculate the width of produced strings.
285          * Assume that characters are always width of "1".  This is
286          * hacky, but it gets the job done for approximation of widths.
287          */

289         sz = 0;
290         skip = 0;
291         while (1) {
292                 rsz = strcspn(cp, "\\");
293                 if (rsz) {
```

```
294                         cp += rsz;
295                         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 norecurse)
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 && '\\' != *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_FONTITALIC):
382                         /* FALLTHROUGH */
383                 case (ESCAPE_FONTBI):
384                         /* FALLTHROUGH */
385                 case (ESCAPE_FONTROMAN):
386                         if (0 == norecurse)
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;
 359                 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(" ");
 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);
 514         if (HTMLFONT_NONE != h->metac)
 515                 h->metaf = HTMLFONT_BOLD == h->metac ?
 516                         print_otag(h, TAG_B, 0, NULL) :
 571                 print_otag(h, TAG_I, 0, NULL);
 572                 break;
 573         default:
 574                 break;
 575         }

 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 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/html.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
   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

  20 __BEGIN_DECLS

  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 };
______unchanged_portion_omitted_

  74 enum    htmlfont {
  75         HTMLFONT_NONE = 0,
```



```
  76         HTMLFONT_BOLD,
  77         HTMLFONT_ITALIC,
  78         HTMLFONT_BI,
  79         HTMLFONT_MAX
  80 };
______unchanged_portion_omitted_

 113 struct  html {
 114         int              flags;
 115 #define HTML_NOSPACE     (1 << 0) /* suppress next space */
 116 #define HTML_IGNDELIM    (1 << 1)
 117 #define HTML_KEEP        (1 << 2)
 118 #define HTML_PREKEEP     (1 << 3)
 119 #define HTML_NONOSPACE   (1 << 4) /* never add spaces */
 120 #define HTML_LITERAL     (1 << 5) /* literal (e.g., <PRE>) context */
 121 #define HTML_SKIPCHAR    (1 << 6) /* skip the next character */
 122         struct tagq      tags; /* stack of open tags */
 123         struct rofftbl   tbl; /* current table */
 124         struct tag      *tblt; /* current open table scope */
 125         struct mchars   *symtab; /* character-escapes */
 126         char            *base_man; /* base for manpage href */
 127         char            *base_includes; /* base for include href */
 128         char            *style; /* style-sheet URI */
 129         char             buf[BUFSIZ]; /* see bufcat and friends */
 130         size_t           buflen;
 131         struct tag      *metaf; /* current open font scope */
 132         enum htmlfont    metal; /* last used font */
 133         enum htmlfont    metac; /* current font mode */
 134         enum htmltype    type; /* output media type */
 135         int              oflags; /* output options */
 136 #define HTML_FRAGMENT    (1 << 0) /* don't emit HTML/HEAD/BODY */
 137 };
______unchanged_portion_omitted_
```

```
**********************************************************
    4877 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/lib.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

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

/*
 * Note that we don't document "legacy" libraries that have moved into
 * libc.  While there will be section 3lib man pages for them, they
 * won't be referenced in other man pages.
 * TBD
 */
LINE("libadm",          "General Administrative Library (libadm, \\-ladm)")
LINE("libbsdmalloc",    "BSD Memory Allocation Library (libbsdmalloc, -lbsdmallo
LINE("libbsm",          "Security and Auditing Library (libbsm, \\lbsm)")
LINE("libc",            "Standard C Library (libc, \\-lc)")
LINE("libc_db",         "Threads Debugging Library (libc_db, \\-lc_db)")
LINE("libcfgadm",       "Configuration Adminstration Library (libcfgadm, \\-lcfg
LINE("libcommputil",    "Communication Protocol Parser Utilities Library (libpco
LINE("libcontract",     "Contract Management Library (libcontract, \\-lcontract)
LINE("libcpc",          "CPU Performance Counters Library (libcpc, \\-lcpc)")
LINE("libcurses",       "Curses Library (libcurses, \\-lcurses)")
LINE("libdat",          "Direct Access Transport Library (libdat, \\-ldat)")
LINE("libdevid",        "Device ID Library (libdevid, \\-ldevid)")
LINE("libdevinfo",      "Device Information Library (libdevinfo, \\-ldevinfo)")
LINE("libdlpi",         "Data Link Provider Interface (DLPI) Library (libdlp, \\
LINE("libdns_sd",       "DNS Service Discovery Library (libdns_sd, \\-ldns_sd)")
LINE("libelf",          "ELF Access Library (libelf, \\-lelf)")
LINE("libexacct",       "Extended Accounting File Access Library (libexacct, \\-
LINE("libfcoe",         "FCoE Port Management Library (libfcoe, \\-lfcoe)")
LINE("libfstyp",        "File System Type Identification Library (libfstyp, \\-l
LINE("libgen",          "String Pattern Matching Library (libgen, \\-lgen)")
LINE("libgss",          "Generic Security Services Library (libgss, \\-lgss)")
LINE("libiscsit",       "iSCSI Management Library (libiscsit, \\-liscsit)")
LINE("libkstat",        "Kernel Statistics Library (libkstat, \\-lkstat)")
LINE("libkvm",          "Kernel VM Library (libkvm, \\-lkvm)")
LINE("libldap",         "LDAP Library (libldap, \\-lldap)")
LINE("liblgrp",         "Locality Group Library (liblgrp, -llgrp)")
LINE("libm",            "Mathematical Library (libm, -lm)")
LINE("libmail",         "User Mailbox Library (libmail, -lmail)")
LINE("libmalloc",       "Memory Allocation Library (libmalloc, -lmalloc)")
LINE("libmd",           "Message Digest Library (libmd, -lmd)")
LINE("libmp",           "Multiple Precision Library (libmp, -lmp)")
LINE("libmpapi",        "Common Multipath Management Library (libmpapi, -lMPAPI)
LINE("libnsl",          "Network Services Library (libnsl, \\-lnsl)")
LINE("libnvpair",       "Name-Value Pair Library (libnvpair, \\-lnvpair)")
LINE("libpam",          "PAM (Pluggable Authentication Module) Library (libpam,
LINE("libpicl",         "PICL Library (libpicl, \\-lpicl)")
LINE("libpicltree",     "PICL Plug-In Library (libpicltree, \\-lpicltree)")
```



```
LINE("libpkcs11",       "PKCS#11 Cryptographic Framework Library (libpkcs11, \\-
LINE("libpool",         "Pool Configuration Manipulation Library (libpool, \\-lp
LINE("libproc",         "Process Control Library (libproc, \\-lproc)")
LINE("libproject",      "Project Database Access Library (libproject, \\-lprojec
LINE("libresolv",       "Resolver Library (libresolv, \\-lresolv \\-lsocket \\-l
LINE("librpc",          "RPC Service Library (librpcsvc, \\-lrpc)")
LINE("librsm",          "Remote Shared Memory Interface Library (librsm, \\-lrsm
LINE("libsasl",         "Simple Authentication and Security Library (libsasl, \\
LINE("libscf",          "Service Configuration Facility Library (libscf, \\-lscf
LINE("libsec",          "File Access Control Library (libsec, \\-lsec)")
LINE("libsecdb",        "Security Attributes Database Library (libsecdb, \\-lsec
LINE("libsip",          "Session Initiation Protocol Library (libsip, \\-lsip)")
LINE("libslp",          "Service Location Protocol Library (libslp, \\-lslp)")
LINE("libsocket",       "Sockets Library (libsocket, \\-lsocket)")
LINE("libstmf",         "SCSI Target Mode Framework Library (libstmf, \\-lstmf)"
LINE("libsysevet",      "System Event Inteface Library (libsysevent, \\-lsyseven
LINE("libtecla",        "Interactive Command Line Input Library (libtecla, \\-lt
LINE("libtnfctl",       "TNF Probe Control Library (libtnfctl, \\-ltnfctl)")
LINE("libtsol",         "Trusted Extensions Library (libtsol, \\-ltsol)")
LINE("libuuid",         "UUID Library (libuuid, \\-luuid)")
LINE("libvolmgt",       "Volume Management Library (libvolmgt, \\-lvolmgt)")
LINE("libxcurses",      "X/Open Curses Library (libxcurses, \\-lxcurses)")
LINE("libxnet",         "X/Open Networking Library (libxnet, \\-lxnet)")
```

```
**********************************************************
    3112 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/libman.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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

 20 enum    man_next {
 21         MAN_NEXT_SIBLING = 0,
 22         MAN_NEXT_CHILD
 23 };
______unchanged_portion_omitted_

 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

 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_NSCOPED      (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 };

 60 extern  const struct man_macro *const man_macros;

 62 __BEGIN_DECLS

 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)
 64 #define           man_pmsg(m, l, p, t) \
 65                   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);
 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);

 83 __END_DECLS

 85 #endif /*!LIBMAN_H*/
```

```
**********************************************************
    3355 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/libmandoc.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
/*      $Id: libmandoc.h,v 1.35 2013/12/15 21:23:52 schwarze Exp $ */
/*      $Id: libmandoc.h,v 1.29 2011/12/02 01:37:14 schwarze Exp $ */
/*
 * Copyright (c) 2009, 2010, 2011, 2012 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifndef LIBMANDOC_H
#define LIBMANDOC_H

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

enum    regs {
        REG_nS = 0, /* nS register */
        REG__MAX
};

__BEGIN_DECLS

struct  roff;
struct  mdoc;
struct  man;

void             mandoc_msg(enum mandocerr, struct mparse *,
                        int, int, const char *);
void             mandoc_vmsg(enum mandocerr, struct mparse *,
                        int, int, const char *, ...);
char            *mandoc_getarg(struct mparse *, char **, int, int *);
char            *mandoc_normdate(struct mparse *, char *, int, int);
int              mandoc_eos(const char *, size_t, int);
int              mandoc_getcontrol(const char *, int *);
int              mandoc_strntoi(const char *, size_t, int);
const char      *mandoc_a2msec(const char*);

void             mdoc_free(struct mdoc *);
struct  mdoc    *mdoc_alloc(struct roff *, struct mparse *, char *);
struct  mdoc    *mdoc_alloc(struct roff *, struct mparse *);
```



```
void             mdoc_reset(struct mdoc *);
int              mdoc_parseln(struct mdoc *, int, char *, int);
int              mdoc_endparse(struct mdoc *);
int              mdoc_addspan(struct mdoc *, const struct tbl_span *);
int              mdoc_addeqn(struct mdoc *, const struct eqn *);

void             man_free(struct man *);
struct  man     *man_alloc(struct roff *, struct mparse *);
void             man_reset(struct man *);
int              man_parseln(struct man *, int, char *, int);
int              man_endparse(struct man *);
int              man_addspan(struct man *, const struct tbl_span *);
int              man_addeqn(struct man *, const struct eqn *);

void             roff_free(struct roff *);
struct roff     *roff_alloc(enum mparset, struct mparse *);
struct roff     *roff_alloc(struct mparse *);
void             roff_reset(struct roff *);
enum rofferr     roff_parseln(struct roff *, int,
                        char **, size_t *, int, int *);
void             roff_endparse(struct roff *);
void             roff_setreg(struct roff *, const char *, int, char sign);
int              roff_getreg(const struct roff *, const char *);
int              roff_regisset(const struct roff *, enum regs);
unsigned int     roff_regget(const struct roff *, enum regs);
void             roff_regunset(struct roff *, enum regs);
char            *roff_strdup(const struct roff *, const char *);
int              roff_getcontrol(const struct roff *,
                        const char *, int *);
#if 0
char             roff_eqndelim(const struct roff *);
void             roff_openeqn(struct roff *, const char *,
                        int, int, const char *);
int              roff_closeeqn(struct roff *);
#endif

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

__END_DECLS

#endif /*!LIBMANDOC_H*/
```

```
**********************************************************
    5162 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/libmdoc.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
/*      $Id: libmdoc.h,v 1.82 2013/10/21 23:47:58 schwarze Exp $ */
/*      $Id: libmdoc.h,v 1.78 2011/12/02 01:37:14 schwarze Exp $ */
/*
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2013 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifndef LIBMDOC_H
#define LIBMDOC_H

enum    mdoc_next {
        MDOC_NEXT_SIBLING = 0,
        MDOC_NEXT_CHILD
};

struct  mdoc {
        struct mparse    *parse; /* parse pointer */
        char             *defos; /* default argument for .Os */
        int               flags; /* parse flags */
#define MDOC_HALT        (1 << 0) /* error in parse: halt */
#define MDOC_LITERAL     (1 << 1) /* in a literal scope */
#define MDOC_PBODY       (1 << 2) /* in the document body */
#define MDOC_NEWLINE     (1 << 3) /* first macro/text in a line */
#define MDOC_PHRASELIT   (1 << 4) /* literal within a partila phrase */
#define MDOC_PPHRASE     (1 << 5) /* within a partial phrase */
#define MDOC_FREECOL     (1 << 6) /* ‘It’ invocation should close */
#define MDOC_SYNOPSIS    (1 << 7) /* SYNOPSIS-style formatting */
#define MDOC_KEEP        (1 << 8) /* in a word keep */
#define MDOC_SMOFF       (1 << 9) /* spacing is off */
        enum mdoc_next    next; /* where to put the next node */
        struct mdoc_node *last; /* the last node parsed */
        struct mdoc_node *first; /* the first node parsed */
        struct mdoc_meta  meta; /* document meta-data */
        enum mdoc_sec     lastnamed;
        enum mdoc_sec     lastsec;
        struct roff      *roff;
};

#define MACRO_PROT_ARGS struct mdoc *mdoc, \
#define MACRO_PROT_ARGS struct mdoc *m, \
                        enum mdoct tok, \
                        int line, \
                        int ppos, \
                        int *pos, \
                        char *buf

struct  mdoc_macro {
        int             (*fp)(MACRO_PROT_ARGS);
```



```
        int              flags;
#define MDOC_CALLABLE    (1 << 0)
#define MDOC_PARSED      (1 << 1)
#define MDOC_EXPLICIT    (1 << 2)
#define MDOC_PROLOGUE    (1 << 3)
#define MDOC_IGNDELIM    (1 << 4)
#define MDOC_JOIN        (1 << 5)
        /* Reserved words in arguments treated as text. */
};
______unchanged_portion_omitted_

extern  const struct mdoc_macro *const mdoc_macros;

__BEGIN_DECLS

#define           mdoc_pmsg(mdoc, l, p, t) \
                  mandoc_msg((t), (mdoc)->parse, (l), (p), NULL)
#define           mdoc_nmsg(mdoc, n, t) \
                  mandoc_msg((t), (mdoc)->parse, (n)->line, (n)->pos, NULL)
#define           mdoc_pmsg(m, l, p, t) \
                  mandoc_msg((t), (m)->parse, (l), (p), NULL)
#define           mdoc_nmsg(m, n, t) \
                  mandoc_msg((t), (m)->parse, (n)->line, (n)->pos, NULL)
int               mdoc_macro(MACRO_PROT_ARGS);
int               mdoc_word_alloc(struct mdoc *,
                        int, int, const char *);
void              mdoc_word_append(struct mdoc *, const char *);
int               mdoc_elem_alloc(struct mdoc *, int, int,
                        enum mdoct, struct mdoc_arg *);
int               mdoc_block_alloc(struct mdoc *, int, int,
                        enum mdoct, struct mdoc_arg *);
int               mdoc_head_alloc(struct mdoc *, int, int, enum mdoct);
int               mdoc_tail_alloc(struct mdoc *, int, int, enum mdoct);
int               mdoc_body_alloc(struct mdoc *, int, int, enum mdoct);
int               mdoc_endbody_alloc(struct mdoc *, int, int, enum mdoct,
                        struct mdoc_node *, enum mdoc_endbody);
int               mdoc_endbody_alloc(struct mdoc *m, int line, int pos,
                        enum mdoct tok, struct mdoc_node *body,
                        enum mdoc_endbody end);
void              mdoc_node_delete(struct mdoc *, struct mdoc_node *);
int               mdoc_node_relink(struct mdoc *, struct mdoc_node *);
void              mdoc_hash_init(void);
enum mdoct        mdoc_hash_find(const char *);
const char       *mdoc_a2att(const char *);
const char       *mdoc_a2lib(const char *);
const char       *mdoc_a2st(const char *);
const char       *mdoc_a2arch(const char *);
const char       *mdoc_a2vol(const char *);
int               mdoc_valid_pre(struct mdoc *, struct mdoc_node *);
int               mdoc_valid_post(struct mdoc *);
enum margverr     mdoc_argv(struct mdoc *, int, enum mdoct,
                        struct mdoc_arg **, int *, char *);
void              mdoc_argv_free(struct mdoc_arg *);
enum margserr     mdoc_args(struct mdoc *, int,
                        int *, char *, enum mdoct, char **);
enum margserr     mdoc_zargs(struct mdoc *, int,
                        int *, char *, char **);
int               mdoc_macroend(struct mdoc *);
enum mdelim       mdoc_isdelim(const char *);

__END_DECLS

#endif /*!LIBMDOC_H*/
```

```
**********************************************************
    2647 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/libroff.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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

 20 __BEGIN_DECLS

 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 };

 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;
 34         struct tbl        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 };
```

_____unchanged_portion_omitted_

```
**********************************************************
    9264 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/main.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  1 /*      $Id: main.c,v 1.167 2012/11/19 17:22:26 schwarze Exp $ */
  1 /*      $Id: main.c,v 1.165 2011/10/06 22:29:12 kristaps Exp $ */
  2 /*
  3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
  4  * Copyright (c) 2010, 2011, 2012 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

 22 #include <assert.h>
 23 #include <stdio.h>
 24 #include <stdint.h>
 25 #include <stdlib.h>
 26 #include <string.h>
 27 #include <unistd.h>

 29 #include "mandoc.h"
 30 #include "main.h"
 31 #include "mdoc.h"
 32 #include "man.h"

 34 #if !defined(__GNUC__) || (__GNUC__ < 2)
 35 # if !defined(lint)
 36 #  define __attribute__(x)
 37 # endif
 38 #endif /* !defined(__GNUC__) || (__GNUC__ < 2) */

 40 typedef void            (*out_mdoc)(void *, const struct mdoc *);
 41 typedef void            (*out_man)(void *, const struct man *);
 42 typedef void            (*out_free)(void *);

 44 enum    outt {
 45         OUTT_ASCII = 0, /* -Tascii */
 46         OUTT_LOCALE,    /* -Tlocale */
 47         OUTT_UTF8,      /* -Tutf8 */
 48         OUTT_TREE,      /* -Ttree */
 49         OUTT_MAN,       /* -Tman */
 50         OUTT_HTML,      /* -Thtml */
 51         OUTT_XHTML,     /* -Txhtml */
 52         OUTT_LINT,      /* -Tlint */
 53         OUTT_PS,        /* -Tps */
 54         OUTT_PDF        /* -Tpdf */
 55 };
______unchanged_portion_omitted_
```



```
 69 static  int               moptions(enum mparset *, char *);
 70 static  void              mmsg(enum mandocerr, enum mandoclevel,
 71                                 const char *, int, int, const char *);
 72 static  void              parse(struct curparse *, int,
 73                                 const char *, enum mandoclevel *);
 74 static  int               toptions(struct curparse *, char *);
 75 static  void              usage(void) __attribute__((noreturn));
 76 static  void              version(void) __attribute__((noreturn));
 77 static  int               woptions(struct curparse *, char *);

 79 static  const char       *progname;

 81 int
 82 main(int argc, char *argv[])
 83 {
 84         int              c;
 85         struct curparse  curp;
 86         enum mparset     type;
 87         enum mandoclevel rc;
 88         char            *defos;

 90         progname = strrchr(argv[0], '/');
 91         if (progname == NULL)
 92                 progname = argv[0];
 93         else
 94                 ++progname;

 96         memset(&curp, 0, sizeof(struct curparse));

 98         type = MPARSE_AUTO;
 99         curp.outtype = OUTT_ASCII;
100         curp.wlevel  = MANDOCLEVEL_FATAL;
101         defos = NULL;

103         /* LINTED */
104         while (-1 != (c = getopt(argc, argv, "I:m:O:T:VW:")))
102         while (-1 != (c = getopt(argc, argv, "m:O:T:VW:")))
105                 switch (c) {
106                 case ('I'):
107                         if (strncmp(optarg, "os=", 3)) {
108                                 fprintf(stderr, "-I%s: Bad argument\n",
109                                         optarg);
110                                 return((int)MANDOCLEVEL_BADARG);
111                         }
112                         if (defos) {
113                                 fprintf(stderr, "-I%s: Duplicate argument\n",
114                                         optarg);
115                                 return((int)MANDOCLEVEL_BADARG);
116                         }
117                         defos = mandoc_strdup(optarg + 3);
118                         break;
119                 case ('m'):
120                         if ( ! moptions(&type, optarg))
121                                 return((int)MANDOCLEVEL_BADARG);
122                         break;
123                 case ('O'):
124                         (void)strlcat(curp.outopts, optarg, BUFSIZ);
125                         (void)strlcat(curp.outopts, ",", BUFSIZ);
126                         break;
127                 case ('T'):
128                         if ( ! toptions(&curp, optarg))
129                                 return((int)MANDOCLEVEL_BADARG);
130                         break;
131                 case ('W'):
132                         if ( ! woptions(&curp, optarg))
133                                 return((int)MANDOCLEVEL_BADARG);
```

```
134                         break;
135                 case ('V'):
136                         version();
137                         /* NOTREACHED */
138                 default:
139                         usage();
140                         /* NOTREACHED */
141                 }

143         curp.mp = mparse_alloc(type, curp.wlevel, mmsg, &curp, defos);
128         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] "
173                         "[-foption] "
190                         "[-mformat] "
191                         "[-Ooption] "
192                         "[-Toutput] "
193                         "[-Wlevel]\n"
194                         "\t      [file ...]\n",
177                         "[-Wlevel] "
178                         "[file...]\n",
195                         progname);

197         exit((int)MANDOCLEVEL_BADARG);
198 }
______unchanged_portion_omitted_
```

```
**********************************************************
   14223 Wed Jul 30 20:55:07 2014
new/usr/src/cmd/mandoc/man.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: man.c,v 1.121 2013/11/10 22:54:40 schwarze Exp $ */
/*      $Id: man.c,v 1.115 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

#include <assert.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "man.h"
#include "mandoc.h"
#include "libman.h"
#include "libmandoc.h"

const   char *const __man_macronames[MAN_MAX] = {
        "br",           "TH",           "SH",           "SS",
        "TP",           "LP",           "PP",           "P",
        "IP",           "HP",           "SM",           "SB",
        "BI",           "IB",           "BR",           "RB",
        "R",            "B",            "I",            "IR",
        "RI",           "na",           "sp",           "nf",
        "fi",           "RE",           "RS",           "DT",
        "UC",           "PD",           "AT",           "in",
        "ft",           "OP",           "EX",           "EE",
        "UR",           "UE"
        "ft",           "OP"
        };

const   char * const *man_macronames = __man_macronames;

static  struct man_node *man_node_alloc(struct man *, int, int,
                                enum man_type, enum mant);
static  int              man_node_append(struct man *,
                                struct man_node *);
static  void             man_node_free(struct man_node *);
static  void             man_node_unlink(struct man *,
                                struct man_node *);
static  int              man_ptext(struct man *, int, char *, int);
static  int              man_pmacro(struct man *, int, char *, int);
```



```
static  void             man_free1(struct man *);
static  void             man_alloc1(struct man *);
static  int              man_descope(struct man *, int, int);


const struct man_node *
man_node(const struct man *man)
man_node(const struct man *m)
{

        assert( ! (MAN_HALT & man->flags));
        return(man->first);
        assert( ! (MAN_HALT & m->flags));
        return(m->first);
}


const struct man_meta *
man_meta(const struct man *man)
man_meta(const struct man *m)
{

        assert( ! (MAN_HALT & man->flags));
        return(&man->meta);
        assert( ! (MAN_HALT & m->flags));
        return(&m->meta);
}
______unchanged_portion_omitted_


int
man_endparse(struct man *man)
man_endparse(struct man *m)
{

        assert( ! (MAN_HALT & man->flags));
        if (man_macroend(man))
        assert( ! (MAN_HALT & m->flags));
        if (man_macroend(m))
                return(1);
        man->flags |= MAN_HALT;
        m->flags |= MAN_HALT;
        return(0);
}


int
man_parseln(struct man *man, int ln, char *buf, int offs)
man_parseln(struct man *m, int ln, char *buf, int offs)
{

        man->flags |= MAN_NEWLINE;
        m->flags |= MAN_NEWLINE;

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

        return (roff_getcontrol(man->roff, buf, &offs) ?
                        man_pmacro(man, ln, buf, offs) :
                        man_ptext(man, ln, buf, offs));
        return (mandoc_getcontrol(buf, &offs) ?
                        man_pmacro(m, ln, buf, offs) :
                        man_ptext(m, ln, buf, offs));
}
______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))
438                 if ( ! man_word_alloc(m, line, offs, ""))
445                                 return(0);
446                         man->next = MAN_NEXT_SIBLING;
440                 return(man_descope(m, line, offs));
447                 }
448                 return(1);
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);
453                         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))
464         if ( ! man_word_alloc(m, line, offs, buf + offs))
473                 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;
475                 m->last->flags |= MAN_EOS;

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

488 static int
489 man_pmacro(struct man *man, int ln, char *buf, int offs)
481 man_pmacro(struct man *m, 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);
489                 man_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
498                 return(1);
499         } else if ('\0' == buf[offs])
500                 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,
511                 mandoc_vmsg(MANDOCERR_MACRO, m->parse, ln,
520                                 ppos, "%s", buf + ppos - 1);
521                 return(1);
522         }

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);
527                 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;
536                         m->flags & MAN_ELINE) {
537                 n = m->last;
546                 assert(MAN_TEXT != n->type);

548                 /* 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,
545                 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;
549                 man_node_delete(m, n);
550                 m->flags &= ~MAN_ELINE;
559         }

561         /*
562          * Remove prior BLINE macro that is being clobbered.
563          */
564         if ((man->flags & MAN_BLINE) &&
556         if ((m->flags & MAN_BLINE) &&
565             (MAN_BSCOPE & man_macros[tok].flags)) {
566                 n = man->last;
558                 n = m->last;

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

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

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

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

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          */

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

601         /* Call to handler... */

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

607         /*
```



```
608          * We weren't in a block-line scope when entering the
609          * above-parsed macro, so return.
610          */

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

618         /*
619          * If we're in a block scope, then allow this macro to slip by
620          * without closing scope around it.
621          */

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

628         /*
629          * If we've opened a new next-line element scope, then return
630          * now, as the next line will close out the block scope.
631          */

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

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

638         assert(MAN_BLINE & man->flags);
639         man->flags &= ~MAN_BLINE;
630         assert(MAN_BLINE & m->flags);
631         m->flags &= ~MAN_BLINE;

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

645 err:    /* Error out. */

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

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

659         /* Adjust siblings. */
```

```
        if (n->prev)
                n->prev->next = n->next;
        if (n->next)
                n->next->prev = n->prev;

        /* Adjust parent. */

        if (n->parent) {
                n->parent->nchild--;
                if (n->parent->child == n)
                        n->parent->child = n->prev ? n->prev : n->next;
        }

        /* Adjust parse point, if applicable. */

        if (man && man->last == n) {
        if (m && m->last == n) {
                /*XXX: this can occur when bailing from validation. */
                /*assert(NULL == n->next);*/
                if (n->prev) {
                        man->last = n->prev;
                        man->next = MAN_NEXT_SIBLING;
                        m->last = n->prev;
                        m->next = MAN_NEXT_SIBLING;
                } else {
                        man->last = n->parent;
                        man->next = MAN_NEXT_CHILD;
                        m->last = n->parent;
                        m->next = MAN_NEXT_CHILD;
                }
        }

        if (man && man->first == n)
                man->first = NULL;
        if (m && m->first == n)
                m->first = NULL;
}

const struct mparse *
man_mparse(const struct man *man)
man_mparse(const struct man *m)
{

        assert(man && man->parse);
        return(man->parse);
        assert(m && m->parse);
        return(m->parse);
}
_____unchanged_portion_omitted_
```

****
2731 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/man.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
****

```
 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

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 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/man_html.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

22 #include <sys/types.h>

24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>

30 #include "mandoc.h"
31 #include "out.h"
32 #include "html.h"
33 #include "man.h"
34 #include "main.h"

36 /* TODO: preserve ident widths. */
37 /* FIXME: have PD set the default vspace width. */

39 #define INDENT            5

41 #define MAN_ARGS          const struct man_meta *man, \
40 #define MAN_ARGS          const struct man_meta *m, \
42                           const struct man_node *n, \
43                           struct mhtml *mh, \
44                           struct html *h

46 struct  mhtml {
47         int               fl;
48 #define MANH_LITERAL     (1 << 0) /* literal context */
49 };
```

_____*unchanged_portion_omitted_*

```
56 static  void             print_bvspace(struct html *,
57                                 const struct man_node *);
58 static  void             print_man(MAN_ARGS);
59 static  void             print_man_head(MAN_ARGS);
60 static  void             print_man_nodelist(MAN_ARGS);
```



```
61 static  void             print_man_node(MAN_ARGS);
62 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);

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 };
```

_____*unchanged_portion_omitted_*

```
147 void
148 html_man(void *arg, const struct man *man)
```

```
142 html_man(void *arg, const struct man *m)
149 {
150         struct mhtml     mh;

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

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);
163                 print_man_head(m, n, mh, h);
170                 print_tagq(h, tt);
171                 print_otag(h, TAG_BODY, 0, NULL);
172                 print_otag(h, TAG_DIV, 1, &tag);
173         } else
174                 t = print_otag(h, TAG_DIV, 1, &tag);

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


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);
181         assert(m->title);
182         assert(m->msec);
183         bufcat_fmt(h, "%s(%s)", m->title, m->msec);
190         print_otag(h, TAG_TITLE, 0, NULL);
191         print_text(h, h->buf);
192 }


195 static void
196 print_man_nodelist(MAN_ARGS)
197 {

199         print_man_node(man, n, mh, h);
193         print_man_node(m, n, mh, h);
200         if (n->next)
201                 print_man_nodelist(man, n->next, mh, h);
195                 print_man_nodelist(m, n->next, mh, h);
202 }


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);
210                 man_root_pre(m, n, mh, h);
217                 break;
218         case (MAN_TEXT):
219                 /*
220                  * If we have a blank line, output a vertical space.
221                  * If we have a space as the first character, break
222                  * before printing the line's data.
223                  */
224                 if ('\0' == *n->string) {
225                         print_otag(h, TAG_P, 0, NULL);
226                         return;
227                 }

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);

234                 print_text(h, n->string);
235                 return;
236         case (MAN_EQN):
237                 print_eqn(h, n->eqn);
238                 break;
239         case (MAN_TBL):
240                 /*
241                  * This will take care of initialising all of the table
242                  * state data for the first table, then tearing it down
243                  * for the last one.
244                  */
245                 print_tbl(h, n->span);
246                 return;
247         default:
248                 /*
249                  * Close out scope of font prior to opening a macro
250                  * scope.
251                  */
252                 if (HTMLFONT_NONE != h->metac) {
253                         h->metal = h->metac;
254                         h->metac = HTMLFONT_NONE;
255                 }

257                 /*
258                  * Close out the current table, if it's open, and unset
259                  * the "meta" table state.  This will be reopened on the
260                  * next table element.
261                  */
262                 if (h->tblt) {
263                         print_tblclose(h);
264                         t = h->tags.head;
265                 }
266                 if (mans[n->tok].pre)
267                         child = (*mans[n->tok].pre)(man, n, mh, h);
261                         child = (*mans[n->tok].pre)(m, n, mh, h);
268                 break;
269         }
```

```
271         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);
471                 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);
```



```
546                 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);
553                                 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) {
641         if (MAN_nf != n->tok) {
648                 print_otag(h, TAG_BR, 0, NULL);
649                 mh->fl &= ~MANH_LITERAL;
650         } else
651                 mh->fl |= MANH_LITERAL;

653         return(0);
654 }
```

*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 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/man_macro.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: man_macro.c,v 1.79 2013/12/25 00:50:05 schwarze Exp $ */
/*      $Id: man_macro.c,v 1.71 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2013 Franco Fichtner <franco@lastsummer.de>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#include "man.h"
#include "mandoc.h"
#include "libmandoc.h"
#include "libman.h"

enum    rew {
        REW_REWIND,
        REW_NOHALT,
        REW_HALT
};

static  int              blk_close(MACRO_PROT_ARGS);
static  int              blk_exp(MACRO_PROT_ARGS);
static  int              blk_imp(MACRO_PROT_ARGS);
static  int              in_line_eoln(MACRO_PROT_ARGS);
static  int              man_args(struct man *, int,
                                 int *, char *, char **);

static  int              rew_scope(enum man_type,
                                 struct man *, enum mant);
static  enum rew         rew_dohalt(enum mant, enum man_type,
                                 const struct man_node *);
static  enum rew         rew_block(enum mant, enum man_type,
                                 const struct man_node *);
static  void             rew_warn(struct man *,
                                 struct man_node *, enum mandocerr);

const   struct man_macro __man_macros[MAN_MAX] = {
        { in_line_eoln, MAN_NSCOPED }, /* br */
        { in_line_eoln, MAN_BSCOPE }, /* TH */
        { blk_imp, MAN_BSCOPE | MAN_SCOPED }, /* SH */
```



```
        { blk_imp, MAN_BSCOPE | MAN_SCOPED }, /* SS */
        { blk_imp, MAN_BSCOPE | MAN_SCOPED | MAN_FSCOPED }, /* TP */
        { blk_imp, MAN_BSCOPE }, /* LP */
        { blk_imp, MAN_BSCOPE }, /* PP */
        { blk_imp, MAN_BSCOPE }, /* P */
        { blk_imp, MAN_BSCOPE }, /* IP */
        { blk_imp, MAN_BSCOPE }, /* HP */
        { in_line_eoln, MAN_SCOPED }, /* SM */
        { in_line_eoln, MAN_SCOPED }, /* SB */
        { in_line_eoln, 0 }, /* BI */
        { in_line_eoln, 0 }, /* IB */
        { in_line_eoln, 0 }, /* BR */
        { in_line_eoln, 0 }, /* RB */
        { in_line_eoln, MAN_SCOPED }, /* R */
        { in_line_eoln, MAN_SCOPED }, /* B */
        { in_line_eoln, MAN_SCOPED }, /* I */
        { in_line_eoln, 0 }, /* IR */
        { in_line_eoln, 0 }, /* RI */
        { in_line_eoln, MAN_NSCOPED }, /* na */
        { in_line_eoln, MAN_NSCOPED }, /* sp */
        { in_line_eoln, MAN_BSCOPE }, /* nf */
        { in_line_eoln, MAN_BSCOPE }, /* fi */
        { blk_close, 0 }, /* RE */
        { blk_exp, MAN_BSCOPE | MAN_EXPLICIT }, /* RS */
        { blk_exp, MAN_EXPLICIT }, /* RS */
        { in_line_eoln, 0 }, /* DT */
        { in_line_eoln, 0 }, /* UC */
        { in_line_eoln, 0 }, /* PD */
        { in_line_eoln, 0 }, /* AT */
        { in_line_eoln, 0 }, /* in */
        { in_line_eoln, 0 }, /* ft */
        { in_line_eoln, 0 }, /* OP */
        { in_line_eoln, MAN_BSCOPE }, /* EX */
        { in_line_eoln, MAN_BSCOPE }, /* EE */
        { blk_exp, MAN_BSCOPE | MAN_EXPLICIT }, /* UR */
        { blk_close, 0 }, /* UE */
};

const   struct man_macro * const man_macros = __man_macros;


/*
 * Warn when "n" is an explicit non-roff macro.
 */
static void
rew_warn(struct man *man, struct man_node *n, enum mandocerr er)
rew_warn(struct man *m, struct man_node *n, enum mandocerr er)
{

        if (er == MANDOCERR_MAX || MAN_BLOCK != n->type)
                return;
        if (MAN_VALID & n->flags)
                return;
        if ( ! (MAN_EXPLICIT & man_macros[n->tok].flags))
                return;

        assert(er < MANDOCERR_FATAL);
        man_nmsg(man, n, er);
        man_nmsg(m, n, er);
}


/*
 * Rewind scope.  If a code "er" != MANDOCERR_MAX has been provided, it
 * will be used if an explicit block scope is being closed out.
 */
```

```
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;

128         assert(to);

130         man->next = MAN_NEXT_SIBLING;
124         m->next = MAN_NEXT_SIBLING;

132         /* LINTED */
133         while (man->last != to) {
127         while (m->last != to) {
134                 /*
135                  * Save the parent here, because we may delete the
136                  * man->last node in the post-validation phase and reset
137                  * it to man->last->parent, causing a step in the closing
130                  * m->last node in the post-validation phase and reset
131                  * it to m->last->parent, causing a step in the closing
138                  * out to be lost.
139                  */
140                 n = man->last->parent;
141                 rew_warn(man, man->last, er);
142                 if ( ! man_valid_post(man))
134                 n = m->last->parent;
135                 rew_warn(m, m->last, er);
136                 if ( ! man_valid_post(m))
143                         return(0);
144                 man->last = n;
145                 assert(man->last);
138                 m->last = n;
139                 assert(m->last);
146         }

148         rew_warn(man, man->last, er);
149         if ( ! man_valid_post(man))
142         rew_warn(m, m->last, er);
143         if ( ! man_valid_post(m))
150                 return(0);

152         return(1);
153 }
______unchanged_portion_omitted_


167 /*
168  * There are three scope levels: scoped to the root (all), scoped to the
169  * section (all less sections), and scoped to subsections (all less
170  * sections and subsections).
171  */
172 static enum rew
173 rew_dohalt(enum mant tok, enum man_type type, const struct man_node *n)
174 {
175         enum rew         c;

177         /* We cannot progress beyond the root ever. */
178         if (MAN_ROOT == n->type)
179                 return(REW_HALT);

181         assert(n->parent);

183         /* Normal nodes shouldn't go to the level of the root. */
184         if (MAN_ROOT == n->parent->type)
185                 return(REW_REWIND);
```



```
187         /* Already-validated nodes should be closed out. */
188         if (MAN_VALID & n->flags)
189                 return(REW_NOHALT);

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
186         if (type == n->type && tok == n->tok)
196                         return(REW_REWIND);
197         }

199         /*
200          * Next follow the implicit scope-smashings as defined by man.7:
201          * section, sub-section, etc.
202          */

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         }

237         return(REW_NOHALT);
238 }


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)
233 rew_scope(enum man_type type, struct man *m, enum mant tok)
248 {
249         struct man_node *n;
```

```
250         enum rew         c;

252         /* LINTED */
253         for (n = man->last; n; n = n->parent) {
239         for (n = m->last; n; n = n->parent) {
254                 /*
255                  * Whether we should stop immediately (REW_HALT), stop
256                  * and rewind until this point (REW_REWIND), or keep
257                  * rewinding (REW_NOHALT).
258                  */
259                 c = rew_dohalt(tok, type, n);
260                 if (REW_HALT == c)
261                         return(1);
262                 if (REW_REWIND == c)
263                         break;
264         }

266         /*
267          * Rewind until the current point.  Warn if we're a roff
268          * instruction that's mowing over explicit scopes.
269          */
270         assert(n);

272         return(man_unscope(man, n, MANDOCERR_MAX));
258         return(man_unscope(m, n, MANDOCERR_MAX));
273 }


276 /*
277  * Close out a generic explicit macro.
278  */
279 /* ARGSUSED */
280 int
281 blk_close(MACRO_PROT_ARGS)
282 {
283         enum mant               ntok;
284         const struct man_node   *nn;

286         switch (tok) {
287         case (MAN_RE):
288                 ntok = MAN_RS;
289                 break;
290         case (MAN_UE):
291                 ntok = MAN_UR;
292                 break;
293         default:
294                 abort();
295                 /* NOTREACHED */
296         }

298         for (nn = man->last->parent; nn; nn = nn->parent)
299                 if (ntok == nn->tok && MAN_BLOCK == nn->type)
281         for (nn = m->last->parent; nn; nn = nn->parent)
282                 if (ntok == nn->tok)
300                         break;

302         if (NULL == nn) {
303                 man_pmsg(man, line, ppos, MANDOCERR_NOSCOPE);
304                 if ( ! rew_scope(MAN_BLOCK, man, MAN_PP))
285         if (NULL == nn)
286                 man_pmsg(m, line, ppos, MANDOCERR_NOSCOPE);

288         if ( ! rew_scope(MAN_BODY, m, ntok))
305                         return(0);
306         } else
307                 man_unscope(man, nn, MANDOCERR_MAX);
```



```
290         if ( ! rew_scope(MAN_BLOCK, m, ntok))
291                 return(0);

309         return(1);
310 }


313 /* ARGSUSED */
314 int
315 blk_exp(MACRO_PROT_ARGS)
316 {
317         struct man_node *n;
318         int              la;
319         char            *p;

321         /* Close out prior implicit scopes. */
304         /*
305          * Close out prior scopes.  "Regular" explicit macros cannot be
306          * nested, but we allow roff macros to be placed just about
307          * anywhere.
308          */

323         if ( ! rew_scope(MAN_BLOCK, man, tok))
310         if ( ! man_block_alloc(m, line, ppos, tok))
324                 return(0);

326         if ( ! man_block_alloc(man, line, ppos, tok))
312         if ( ! man_head_alloc(m, line, ppos, tok))
327                 return(0);
328         if ( ! man_head_alloc(man, line, ppos, tok))
329                 return(0);

331         for (;;) {
332                 la = *pos;
333                 if ( ! man_args(man, line, pos, buf, &p))
317                 if ( ! man_args(m, line, pos, buf, &p))
334                         break;
335                 if ( ! man_word_alloc(man, line, la, p))
319                 if ( ! man_word_alloc(m, line, la, p))
336                         return(0);
337         }

339         assert(man);
323         assert(m);
340         assert(tok != MAN_MAX);

342         for (n = man->last; n; n = n->parent) {
343                 if (n->tok != tok)
344                         continue;
345                 assert(MAN_HEAD == n->type);
346                 man_unscope(man, n, MANDOCERR_MAX);
347                 break;
348         }

350         return(man_body_alloc(man, line, ppos, tok));
326         if ( ! rew_scope(MAN_HEAD, m, tok))
327                 return(0);
328         return(man_body_alloc(m, line, ppos, tok));
351 }



355 /*
356  * Parse an implicit-block macro.  These contain a MAN_HEAD and a
357  * MAN_BODY contained within a MAN_BLOCK.  Rules for closing out other
358  * scopes, such as 'SH' closing out an 'SS', are defined in the rew
```

```
 359  * routines.
 360  */
 361 /* ARGSUSED */
 362 int
 363 blk_imp(MACRO_PROT_ARGS)
 364 {
 365         int              la;
 366         char            *p;
 367         struct man_node *n;

 369         /* Close out prior scopes. */

 371         if ( ! rew_scope(MAN_BODY, man, tok))
 349         if ( ! rew_scope(MAN_BODY, m, tok))
 372                 return(0);
 373         if ( ! rew_scope(MAN_BLOCK, man, tok))
 351         if ( ! rew_scope(MAN_BLOCK, m, tok))
 374                 return(0);

 376         /* Allocate new block & head scope. */

 378         if ( ! man_block_alloc(man, line, ppos, tok))
 356         if ( ! man_block_alloc(m, line, ppos, tok))
 379                 return(0);
 380         if ( ! man_head_alloc(man, line, ppos, tok))
 358         if ( ! man_head_alloc(m, line, ppos, tok))
 381                 return(0);

 383         n = man->last;
 361         n = m->last;

 385         /* Add line arguments. */

 387         for (;;) {
 388                 la = *pos;
 389                 if ( ! man_args(man, line, pos, buf, &p))
 367                 if ( ! man_args(m, line, pos, buf, &p))
 390                         break;
 391                 if ( ! man_word_alloc(man, line, la, p))
 369                 if ( ! man_word_alloc(m, line, la, p))
 392                         return(0);
 393         }

 395         /* Close out head and open body (unless MAN_SCOPE). */

 397         if (MAN_SCOPED & man_macros[tok].flags) {
 398                 /* If we're forcing scope ('TP'), keep it open. */
 399                 if (MAN_FSCOPED & man_macros[tok].flags) {
 400                         man->flags |= MAN_BLINE;
 378                         m->flags |= MAN_BLINE;
 401                         return(1);
 402                 } else if (n == man->last) {
 403                         man->flags |= MAN_BLINE;
 380                 } else if (n == m->last) {
 381                         m->flags |= MAN_BLINE;
 404                         return(1);
 405                 }
 406         }

 408         if ( ! rew_scope(MAN_HEAD, man, tok))
 386         if ( ! rew_scope(MAN_HEAD, m, tok))
 409                 return(0);
 410         return(man_body_alloc(man, line, ppos, tok));
 388         return(man_body_alloc(m, line, ppos, tok));
 411 }
```



```
 414 /* ARGSUSED */
 415 int
 416 in_line_eoln(MACRO_PROT_ARGS)
 417 {
 418         int              la;
 419         char            *p;
 420         struct man_node *n;

 422         if ( ! man_elem_alloc(man, line, ppos, tok))
 400         if ( ! man_elem_alloc(m, line, ppos, tok))
 423                 return(0);

 425         n = man->last;
 403         n = m->last;

 427         for (;;) {
 428                 la = *pos;
 429                 if ( ! man_args(man, line, pos, buf, &p))
 407                 if ( ! man_args(m, line, pos, buf, &p))
 430                         break;
 431                 if ( ! man_word_alloc(man, line, la, p))
 409                 if ( ! man_word_alloc(m, line, la, p))
 432                         return(0);
 433         }

 435         /*
 436          * Append MAN_EOS in case the last snipped argument
 437          * ends with a dot, e.g. '.IR syslog (3).'
 438          */

 440         if (n != man->last &&
 441             mandoc_eos(man->last->string, strlen(man->last->string), 0))
 442                 man->last->flags |= MAN_EOS;

 444         /*
 445          * If no arguments are specified and this is MAN_SCOPED (i.e.,
 446          * next-line scoped), then set our mode to indicate that we're
 447          * waiting for terms to load into our context.
 448          */

 450         if (n == man->last && MAN_SCOPED & man_macros[tok].flags) {
 419         if (n == m->last && MAN_SCOPED & man_macros[tok].flags) {
 451                 assert( ! (MAN_NSCOPED & man_macros[tok].flags));
 452                 man->flags |= MAN_ELINE;
 421                 m->flags |= MAN_ELINE;
 453                 return(1);
 454         }

 456         /* Set ignorable context, if applicable. */

 458         if (MAN_NSCOPED & man_macros[tok].flags) {
 459                 assert( ! (MAN_SCOPED & man_macros[tok].flags));
 460                 man->flags |= MAN_ILINE;
 429                 m->flags |= MAN_ILINE;
 461         }

 463         assert(MAN_ROOT != man->last->type);
 464         man->next = MAN_NEXT_SIBLING;
 432         assert(MAN_ROOT != m->last->type);
 433         m->next = MAN_NEXT_SIBLING;
 465
 466         /*
 467          * Rewind our element scope.  Note that when TH is pruned, we'll
 468          * be back at the root, so make sure that we don't clobber as
 469          * its sibling.
```

```
 470          */

 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 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/man_term.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: man_term.c,v 1.139 2013/12/22 23:34:13 schwarze Exp $ */
/*      $Id: man_term.c,v 1.127 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2010, 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mandoc.h"
#include "out.h"
#include "man.h"
#include "term.h"
#include "main.h"

#define MAXMARGINS        64 /* maximum number of indented scopes */

/* FIXME: have PD set the default vspace width. */

struct  mtermp {
        int              fl;
#define MANT_LITERAL     (1 << 0)
        size_t           lmargin[MAXMARGINS]; /* margins (incl. visible page) *
        int              lmargincur; /* index of current margin */
        int              lmarginsz; /* actual number of nested margins */
        size_t           offset; /* default offset to visible page */
        int              pardist; /* vert. space before par., unit: [v] */
};

#define DECL_ARGS         struct termp *p, \
                          struct mtermp *mt, \
                          const struct man_node *n, \
                          const struct man_meta *meta
                          const struct man_meta *m

struct  termact {
```



```
        int             (*pre)(DECL_ARGS);
        void            (*post)(DECL_ARGS);
        int               flags;
#define MAN_NOTEXT       (1 << 0) /* Never has text children. */
};

static  int               a2width(const struct termp *, const char *);
static  size_t            a2height(const struct termp *, const char *);

static  void              print_man_nodelist(DECL_ARGS);
static  void              print_man_node(DECL_ARGS);
static  void              print_man_head(struct termp *, const void *);
static  void              print_man_foot(struct termp *, const void *);
static  void              print_bvspace(struct termp *,
                                  const struct man_node *, int);
                                  const struct man_node *);

static  int               pre_B(DECL_ARGS);
static  int               pre_HP(DECL_ARGS);
static  int               pre_I(DECL_ARGS);
static  int               pre_IP(DECL_ARGS);
static  int               pre_OP(DECL_ARGS);
static  int               pre_PD(DECL_ARGS);
static  int               pre_PP(DECL_ARGS);
static  int               pre_RS(DECL_ARGS);
static  int               pre_SH(DECL_ARGS);
static  int               pre_SS(DECL_ARGS);
static  int               pre_TP(DECL_ARGS);
static  int               pre_UR(DECL_ARGS);
static  int               pre_alternate(DECL_ARGS);
static  int               pre_ft(DECL_ARGS);
static  int               pre_ign(DECL_ARGS);
static  int               pre_in(DECL_ARGS);
static  int               pre_literal(DECL_ARGS);
static  int               pre_sp(DECL_ARGS);

static  void              post_IP(DECL_ARGS);
static  void              post_HP(DECL_ARGS);
static  void              post_RS(DECL_ARGS);
static  void              post_SH(DECL_ARGS);
static  void              post_SS(DECL_ARGS);
static  void              post_TP(DECL_ARGS);
static  void              post_UR(DECL_ARGS);

static  const struct termact termacts[MAN_MAX] = {
        { pre_sp, NULL, MAN_NOTEXT }, /* br */
        { NULL, NULL, 0 }, /* TH */
        { pre_SH, post_SH, 0 }, /* SH */
        { pre_SS, post_SS, 0 }, /* SS */
        { pre_TP, post_TP, 0 }, /* TP */
        { pre_PP, NULL, 0 }, /* LP */
        { pre_PP, NULL, 0 }, /* PP */
        { pre_PP, NULL, 0 }, /* P */
        { pre_IP, post_IP, 0 }, /* IP */
        { pre_HP, post_HP, 0 }, /* HP */
        { NULL, NULL, 0 }, /* SM */
        { pre_B, NULL, 0 }, /* SB */
        { pre_alternate, NULL, 0 }, /* BI */
        { pre_alternate, NULL, 0 }, /* IB */
        { pre_alternate, NULL, 0 }, /* BR */
        { pre_alternate, NULL, 0 }, /* RB */
        { NULL, NULL, 0 }, /* R */
        { pre_B, NULL, 0 }, /* B */
        { pre_I, NULL, 0 }, /* I */
        { pre_alternate, NULL, 0 }, /* IR */
        { pre_alternate, NULL, 0 }, /* RI */
```

```
119         { pre_ign, NULL, MAN_NOTEXT }, /* na */
120         { 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 termp            *p;
144         const struct man_node   *n;
145         const struct man_meta   *meta;
139         const struct man_meta   *m;
146         struct mtermp            mt;

148         p = (struct termp *)arg;

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

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

157         if (NULL == p->symtab)
158                 p->symtab = mchars_alloc();

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

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

166         memset(&mt, 0, sizeof(struct mtermp));

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

172         if (n->child)
173                 print_man_nodelist(p, &mt, n->child, meta);
166                 print_man_nodelist(p, &mt, n->child, m);

175         term_end(p);
176 }
______unchanged_portion_omitted_

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 termp *p, const struct man_node *n, int pardist)
204 print_bvspace(struct termp *p, const struct man_node *n)
212 {
213         int      i;

215         term_newln(p);

217         if (n->body && n->body->child)
218                 if (MAN_TBL == n->body->child->type)
219                         return;

221         if (MAN_ROOT == n->parent->type || MAN_RS != n->parent->tok)
222                 if (NULL == n->prev)
223                         return;

225         for (i = 0; i < pardist; i++)
226                 term_vspace(p);
227 }
______unchanged_portion_omitted_


248 /* ARGSUSED */
249 static int
250 pre_literal(DECL_ARGS)
251 {

253         term_newln(p);

255         if (MAN_nf == n->tok || MAN_EX == n->tok)
246         if (MAN_nf == n->tok)
256                 mt->fl |= MANT_LITERAL;
257         else
258                 mt->fl &= ~MANT_LITERAL;

260         /*
261          * Unlike .IP and .TP, .HP does not have a HEAD.
262          * So in case a second call to term_flushln() is needed,
263          * indentation has to be set up explicitly.
264          */
265         if (MAN_HP == n->parent->tok && p->rmargin < p->maxrmargin) {
266                 p->offset = p->rmargin;
267                 p->rmargin = p->maxrmargin;
268                 p->trailspace = 0;
269                 p->flags &= ~TERMP_NOBREAK;
259                 p->flags &= ~(TERMP_NOBREAK | TERMP_TWOSPACE);
270                 p->flags |= TERMP_NOSPACE;
271         }

273         return(0);
274 }

276 /* ARGSUSED */
277 static int
278 pre_PD(DECL_ARGS)
279 {

281         n = n->child;
282         if (0 == n) {
```

```
283                 mt->pardist = 1;
284                 return(0);
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);
310                 print_man_node(p, mt, nn, m);
336                 if (nn->next)
337                         p->flags |= TERMP_NOSPACE;
338         }

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 */
444                 if (MAN_SS == n->parent->tok)
482                         return(0);
483                 default:
484                         break;
446                 if (MAN_SH == n->parent->tok)
447                         return(0);
485                 }
486         }

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

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

515         return(0);
516 }


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) {
517         case (MAN_BLOCK):
518                 term_flushln(p);
519                 break;
571         case (MAN_BODY):
572                 term_newln(p);
521                 term_flushln(p);
573                 p->flags &= ~TERMP_NOBREAK;
574                 p->trailspace = 0;
523                 p->flags &= ~TERMP_TWOSPACE;
575                 p->offset = mt->offset;
576                 p->rmargin = p->maxrmargin;
577                 break;
578         default:
579                 break;
580         }
581 }


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);
541                 print_bvspace(p, n);
593                 break;
594         default:
595                 p->offset = mt->offset;
596                 break;
597         }

599         return(MAN_HEAD != n->type);
600 }


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);
568                 print_bvspace(p, n);
621                 /* FALLTHROUGH */
622         default:
623                 return(1);
624         }

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);
601                         print_man_node(p, mt, n->child, m);

655                 if (savelit)
656                         mt->fl |= MANT_LITERAL;

658                 return(0);
659         case (MAN_BODY):
660                 p->offset = mt->offset + len;
661                 p->rmargin = p->maxrmargin;
662                 break;
663         default:
664                 break;
665         }

667         return(1);
668 }


671 /* ARGSUSED */
672 static void
673 post_IP(DECL_ARGS)
674 {

676         switch (n->type) {
677         case (MAN_HEAD):
678                 term_flushln(p);
679                 p->flags &= ~TERMP_NOBREAK;
680                 p->trailspace = 0;
681                 p->rmargin = p->maxrmargin;
682                 break;
683         case (MAN_BODY):
684                 term_newln(p);
685                 p->offset = mt->offset;
686                 break;
687         default:
688                 break;
689         }
690 }


693 /* ARGSUSED */
694 static int
695 pre_TP(DECL_ARGS)
696 {
697         const struct man_node   *nn;
698         size_t                   len;
699         int                      savelit, ival;

701         switch (n->type) {
702         case (MAN_HEAD):
703                 p->flags |= TERMP_NOBREAK;
704                 p->trailspace = 1;
705                 break;
706         case (MAN_BODY):
707                 p->flags |= TERMP_NOSPACE;
708                 break;
709         case (MAN_BLOCK):
710                 print_bvspace(p, n, mt->pardist);
655                 print_bvspace(p, n);
711                 /* FALLTHROUGH */
712         default:
```



```
713                 return(1);
714         }

716         len = (size_t)mt->lmargin[mt->lmargincur];
717         ival = -1;

719         /* Calculate offset. */

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;

726         switch (n->type) {
727         case (MAN_HEAD):
728                 /* Handle zero-length properly. */
729                 if (0 == len)
730                         len = term_len(p, 1);

732                 p->offset = mt->offset;
733                 p->rmargin = mt->offset + len;

735                 savelit = MANT_LITERAL & mt->fl;
736                 mt->fl &= ~MANT_LITERAL;

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);
686                                 print_man_node(p, mt, nn, m);

743                 if (savelit)
744                         mt->fl |= MANT_LITERAL;
745                 if (ival >= 0)
746                         mt->lmargin[mt->lmargincur] = (size_t)ival;

748                 return(0);
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         }

759         return(1);
760 }


763 /* ARGSUSED */
764 static void
765 post_TP(DECL_ARGS)
766 {

768         switch (n->type) {
769         case (MAN_HEAD):
770                 term_flushln(p);
714                 p->flags &= ~TERMP_NOBREAK;
715                 p->flags &= ~TERMP_TWOSPACE;
716                 p->rmargin = p->maxrmargin;
771                 break;
772         case (MAN_BODY):
773                 term_newln(p);
774                 p->offset = mt->offset;
```

```
                break;
        default:
                break;
        }
}


/* ARGSUSED */
static int
pre_SS(DECL_ARGS)
{
        int      i;

        switch (n->type) {
        case (MAN_BLOCK):
                mt->fl &= ~MANT_LITERAL;
                mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
                mt->offset = term_len(p, p->defindent);
                /* If following a prior empty 'SS', no vspace. */
                if (n->prev && MAN_SS == n->prev->tok)
                        if (NULL == n->prev->body->child)
                                break;
                if (NULL == n->prev)
                        break;
                for (i = 0; i < mt->pardist; i++)
                        term_vspace(p);
                break;
        case (MAN_HEAD):
                term_fontrepl(p, TERMFONT_BOLD);
                p->offset = term_len(p, 3);
                p->offset = term_len(p, p->defindent/2);
                break;
        case (MAN_BODY):
                p->offset = mt->offset;
                break;
        default:
                break;
        }

        return(1);
}
```

*unchanged_portion_omitted_*

```
/* ARGSUSED */
static int
pre_SH(DECL_ARGS)
{
        int      i;

        switch (n->type) {
        case (MAN_BLOCK):
                mt->fl &= ~MANT_LITERAL;
                mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
                mt->offset = term_len(p, p->defindent);
                /* If following a prior empty 'SH', no vspace. */
                if (n->prev && MAN_SH == n->prev->tok)
                        if (NULL == n->prev->body->child)
                                break;
                /* If the first macro, no vspae. */
                if (NULL == n->prev)
                        break;
                for (i = 0; i < mt->pardist; i++)
                        term_vspace(p);
                break;
        case (MAN_HEAD):
```



```
                term_fontrepl(p, TERMFONT_BOLD);
                p->offset = 0;
                break;
        case (MAN_BODY):
                p->offset = mt->offset;
                break;
        default:
                break;
        }

        return(1);
}
```

*unchanged_portion_omitted_*

```
/* ARGSUSED */
static int
pre_UR(DECL_ARGS)
{

        return (MAN_HEAD != n->type);
}

/* ARGSUSED */
static void
post_UR(DECL_ARGS)
{

        if (MAN_BLOCK != n->type)
                return;

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

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

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

static void
print_man_node(DECL_ARGS)
{
        size_t           rm, rmax;
        int              c;

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

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

                /*
                 * If we're in a literal context, make sure that words
                 * togehter on the same line stay together.  This is a
                 * POST-printing call, so we check the NEXT word.  Since
```

```
 918                  * -man doesn't have nested macros, we don't need to be
 919                  * 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 termp *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 termp *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                 strlcpy(buf, meta->vol, BUFSIZ);
1060         if (m->vol)
1061                 strlcpy(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 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/man_validate.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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 */
91         { NULL, NULL }, /* IP */
97         { NULL, NULL }, /* HP */
98         { NULL, NULL }, /* SM */
99         { NULL, NULL }, /* SB */
100        { NULL, NULL }, /* BI */
101        { NULL, NULL }, /* IB */
102        { NULL, NULL }, /* BR */
103        { NULL, NULL }, /* RB */
104        { NULL, NULL }, /* R */
105        { NULL, NULL }, /* B */
106        { NULL, NULL }, /* I */
107        { NULL, NULL }, /* IR */
108        { NULL, NULL }, /* RI */
109        { NULL, posts_eq0 }, /* na */
110        { NULL, posts_sp }, /* sp */
111        { NULL, posts_nf }, /* nf */
112        { NULL, posts_fi }, /* fi */
113        { NULL, NULL }, /* RE */
114        { NULL, posts_part }, /* RS */
115        { NULL, NULL }, /* DT */
116        { NULL, posts_uc }, /* UC */
117        { NULL, posts_le1 }, /* PD */
112        { NULL, NULL }, /* PD */
118        { NULL, posts_at }, /* AT */
119        { NULL, NULL }, /* in */
120        { NULL, posts_ft }, /* ft */
```

```
121         { NULL, posts_eq2 }, /* OP */
122         { 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)
121 man_valid_pre(struct man *m, struct man_node *n)
131 {
132         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))
141                 if ( ! (*cp)(m, n))
151                         return(0);
152         return(1);
153 }


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

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

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



```
179         if (NULL == (cp = man_valids[man->last->tok].posts))
170         if (NULL == (cp = man_valids[m->last->tok].posts))
180                 return(1);
181         for ( ; *cp; cp++)
182                 if ( ! (*cp)(man, man->last))
173                 if ( ! (*cp)(m, m->last))
183                         return(0);

185         return(1);
186 }


189 static int
190 check_root(CHKARGS)
191 {

193         if (MAN_BLINE & man->flags)
194                 man_nmsg(man, n, MANDOCERR_SCOPEEXIT);
195         else if (MAN_ELINE & man->flags)
196                 man_nmsg(man, n, MANDOCERR_SCOPEEXIT);
184         if (MAN_BLINE & m->flags)
185                 man_nmsg(m, n, MANDOCERR_SCOPEEXIT);
186         else if (MAN_ELINE & m->flags)
187                 man_nmsg(m, n, MANDOCERR_SCOPEEXIT);

198         man->flags &= ~MAN_BLINE;
199         man->flags &= ~MAN_ELINE;
189         m->flags &= ~MAN_BLINE;
190         m->flags &= ~MAN_ELINE;

201         if (NULL == man->first->child) {
202                 man_nmsg(man, n, MANDOCERR_NODOCBODY);
192         if (NULL == m->first->child) {
193                 man_nmsg(m, n, MANDOCERR_NODOCBODY);
203                 return(0);
204         } else if (NULL == man->meta.title) {
205                 man_nmsg(man, n, MANDOCERR_NOTITLE);
195         } else if (NULL == m->meta.title) {
196                 man_nmsg(m, n, MANDOCERR_NOTITLE);

207                 /*
208                  * If a title hasn't been set, do so now (by
209                  * implication, date and section also aren't set).
210                  */

212                 man->meta.title = mandoc_strdup("unknown");
213                 man->meta.msec = mandoc_strdup("1");
214                 man->meta.date = mandoc_normdate
215                         (man->parse, NULL, n->line, n->pos);
203                 m->meta.title = mandoc_strdup("unknown");
204                 m->meta.msec = mandoc_strdup("1");
205                 m->meta.date = mandoc_normdate
206                         (m->parse, NULL, n->line, n->pos);
216         }

218         return(1);
219 }

221 static void
222 check_text(CHKARGS)
223 {
224         char            *cp, *p;

226         if (MAN_LITERAL & man->flags)
217         if (MAN_LITERAL & m->flags)
227                 return;
```

```
        cp = n->string;
        for (p = cp; NULL != (p = strchr(p, '\t')); p++)
                man_pmsg(man, n->line, (int)(p - cp), MANDOCERR_BADTAB);
                man_pmsg(m, n->line, (int)(p - cp), MANDOCERR_BADTAB);
}

#define INEQ_DEFINE(x, ineq, name) \
static int \
check_##name(CHKARGS) \
{ \
        if (n->nchild ineq (x)) \
                return(1); \
        mandoc_vmsg(MANDOCERR_ARGCOUNT, man->parse, n->line, n->pos, \
        mandoc_vmsg(MANDOCERR_ARGCOUNT, m->parse, n->line, n->pos, \
                        "line arguments %s %d (have %d)", \
                        #ineq, (x), n->nchild); \
        return(1); \
}

INEQ_DEFINE(0, ==, eq0)
INEQ_DEFINE(2, ==, eq2)
INEQ_DEFINE(1, <=, le1)
INEQ_DEFINE(2, >=, ge2)
INEQ_DEFINE(5, <=, le5)

static int
check_head1(CHKARGS)
{

        if (MAN_HEAD == n->type && 1 != n->nchild)
                mandoc_vmsg(MANDOCERR_ARGCOUNT, man->parse, n->line,
                    n->pos, "line arguments eq 1 (have %d)", n->nchild);

        return(1);
}

static int
post_ft(CHKARGS)
{
        char    *cp;
        int      ok;

        if (0 == n->nchild)
                return(1);

        ok = 0;
        cp = n->child->string;
        switch (*cp) {
        case ('1'):
                /* FALLTHROUGH */
        case ('2'):
                /* FALLTHROUGH */
        case ('3'):
                /* FALLTHROUGH */
        case ('4'):
                /* FALLTHROUGH */
        case ('I'):
                /* FALLTHROUGH */
        case ('P'):
                /* FALLTHROUGH */
        case ('R'):
                if ('\0' == cp[1])
                        ok = 1;
                break;
        case ('B'):
```



```
                if ('\0' == cp[1] || ('I' == cp[1] && '\0' == cp[2]))
                        ok = 1;
                break;
        case ('C'):
                if ('W' == cp[1] && '\0' == cp[2])
                        ok = 1;
                break;
        default:
                break;
        }

        if (0 == ok) {
                mandoc_vmsg
                        (MANDOCERR_BADFONT, man->parse,
                        (MANDOCERR_BADFONT, m->parse,
                         n->line, n->pos, "%s", cp);
                *cp = '\0';
        }

        if (1 < n->nchild)
                mandoc_vmsg
                        (MANDOCERR_ARGCOUNT, man->parse, n->line,
                        (MANDOCERR_ARGCOUNT, m->parse, n->line,
                         n->pos, "want one child (have %d)",
                         n->nchild);

        return(1);
}

static int
pre_sec(CHKARGS)
{

        if (MAN_BLOCK == n->type)
                man->flags &= ~MAN_LITERAL;
                m->flags &= ~MAN_LITERAL;
        return(1);
}

static int
post_sec(CHKARGS)
{

        if ( ! (MAN_HEAD == n->type && 0 == n->nchild))
                return(1);

        man_nmsg(man, n, MANDOCERR_SYNTARGCOUNT);
        man_nmsg(m, n, MANDOCERR_SYNTARGCOUNT);
        return(0);
}

static int
check_part(CHKARGS)
{

        if (MAN_BODY == n->type && 0 == n->nchild)
                mandoc_msg(MANDOCERR_ARGCWARN, man->parse, n->line,
                mandoc_msg(MANDOCERR_ARGCWARN, m->parse, n->line,
                                n->pos, "want children (have none)");

        return(1);
}


static int
check_par(CHKARGS)
```

```
353 {
355         switch (n->type) {
356         case (MAN_BLOCK):
357                 if (0 == n->body->nchild)
358                         man_node_delete(man, n);
338                         man_node_delete(m, n);
359                 break;
360         case (MAN_BODY):
361                 if (0 == n->nchild)
362                         man_nmsg(man, n, MANDOCERR_IGNPAR);
342                         man_nmsg(m, n, MANDOCERR_IGNPAR);
363                 break;
364         case (MAN_HEAD):
365                 if (n->nchild)
366                         man_nmsg(man, n, MANDOCERR_ARGSLOST);
346                         man_nmsg(m, n, MANDOCERR_ARGSLOST);
367                 break;
368         default:
369                 break;
370         }
372         return(1);
373 }
375 static int
376 post_IP(CHKARGS)
377 {
379         switch (n->type) {
380         case (MAN_BLOCK):
381                 if (0 == n->head->nchild && 0 == n->body->nchild)
382                         man_node_delete(man, n);
383                 break;
384         case (MAN_BODY):
385                 if (0 == n->parent->head->nchild && 0 == n->nchild)
386                         man_nmsg(man, n, MANDOCERR_IGNPAR);
387                 break;
388         default:
389                 break;
390         }
391         return(1);
392 }
394 static int
395 post_TH(CHKARGS)
396 {
397         const char      *p;
398         int              line, pos;
400         free(man->meta.title);
401         free(man->meta.vol);
402         free(man->meta.source);
403         free(man->meta.msec);
404         free(man->meta.date);
362         if (m->meta.title)
363                 free(m->meta.title);
364         if (m->meta.vol)
365                 free(m->meta.vol);
366         if (m->meta.source)
367                 free(m->meta.source);
368         if (m->meta.msec)
369                 free(m->meta.msec);
370         if (m->meta.date)
371                 free(m->meta.date);
```



```
406         line = n->line;
407         pos = n->pos;
408         man->meta.title = man->meta.vol = man->meta.date =
409                 man->meta.msec = man->meta.source = NULL;
375         m->meta.title = m->meta.vol = m->meta.date =
376                 m->meta.msec = m->meta.source = NULL;
411         /* ->TITLE<- MSEC DATE SOURCE VOL */
413         n = n->child;
414         if (n && n->string) {
415                 for (p = n->string; '\0' != *p; p++) {
416                         /* Only warn about this once... */
417                         if (isalpha((unsigned char)*p) &&
418                                         ! isupper((unsigned char)*p)) {
419                                 man_nmsg(man, n, MANDOCERR_UPPERCASE);
386                                 man_nmsg(m, n, MANDOCERR_UPPERCASE);
420                                 break;
421                         }
422                 }
423                 man->meta.title = mandoc_strdup(n->string);
390                 m->meta.title = mandoc_strdup(n->string);
424         } else
425                 man->meta.title = mandoc_strdup("");
392                 m->meta.title = mandoc_strdup("");
427         /* TITLE ->MSEC<- DATE SOURCE VOL */
429         if (n)
430                 n = n->next;
431         if (n && n->string)
432                 man->meta.msec = mandoc_strdup(n->string);
399                 m->meta.msec = mandoc_strdup(n->string);
433         else
434                 man->meta.msec = mandoc_strdup("");
401                 m->meta.msec = mandoc_strdup("");
436         /* TITLE MSEC ->DATE<- SOURCE VOL */
438         if (n)
439                 n = n->next;
440         if (n && n->string && '\0' != n->string[0]) {
441                 pos = n->pos;
442                 man->meta.date = mandoc_normdate
443                     (man->parse, n->string, line, pos);
409                 m->meta.date = mandoc_normdate
410                     (m->parse, n->string, line, pos);
444         } else
445                 man->meta.date = mandoc_strdup("");
412                 m->meta.date = mandoc_strdup("");
447         /* TITLE MSEC DATE ->SOURCE<- VOL */
449         if (n && (n = n->next))
450                 man->meta.source = mandoc_strdup(n->string);
417                 m->meta.source = mandoc_strdup(n->string);
452         /* TITLE MSEC DATE SOURCE ->VOL<- */
453         /* If missing, use the default VOL name for MSEC. */
455         if (n && (n = n->next))
456                 man->meta.vol = mandoc_strdup(n->string);
457         else if ('\0' != man->meta.msec[0] &&
458             (NULL != (p = mandoc_a2msec(man->meta.msec))))
459                 man->meta.vol = mandoc_strdup(p);
423                 m->meta.vol = mandoc_strdup(n->string);
```

```
424         else if ('\0' != m->meta.msec[0] &&
425             (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);
432         man_node_delete(m, m->last);
466         return(1);
467 }

469 static int
470 post_nf(CHKARGS)
471 {

473         if (MAN_LITERAL & man->flags)
474                 man_nmsg(man, n, MANDOCERR_SCOPEREP);
440         if (MAN_LITERAL & m->flags)
441                 man_nmsg(m, n, MANDOCERR_SCOPEREP);

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

480 static int
481 post_fi(CHKARGS)
482 {

484         if ( ! (MAN_LITERAL & man->flags))
485                 man_nmsg(man, n, MANDOCERR_WNOSCOPE);
451         if ( ! (MAN_LITERAL & m->flags))
452                 man_nmsg(m, n, MANDOCERR_WNOSCOPE);

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

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

502         const char      *p, *s;

504         n = n->child;

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



```
516                 else if (0 == strcmp(s, "6"))
517                         p = bsd_versions[3];
518                 else if (0 == strcmp(s, "7"))
519                         p = bsd_versions[4];
520                 else
521                         p = bsd_versions[0];
522         }

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

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

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         };

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

542         n = n->child;

544         if (NULL == n || MAN_TEXT != n->type)
545                 p = unix_versions[0];
546         else {
547                 s = n->string;
548                 if (0 == strcmp(s, "3"))
549                         p = unix_versions[0];
550                 else if (0 == strcmp(s, "4"))
551                         p = unix_versions[1];
552                 else if (0 == strcmp(s, "5")) {
553                         nn = n->next;
554                         if (nn && MAN_TEXT == nn->type && nn->string[0])
555                                 p = unix_versions[3];
556                         else
557                                 p = unix_versions[2];
558                 } else
559                         p = unix_versions[0];
560         }

562         free(man->meta.source);
563         man->meta.source = mandoc_strdup(p);
531         if (m->meta.source)
532                 free(m->meta.source);

534         m->meta.source = mandoc_strdup(p);
564         return(1);
565 }

567 static int
568 post_vs(CHKARGS)
569 {

571         if (NULL != n->prev)
572                 return(1);
```

```
574         switch (n->parent->tok) {
575         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.
543          * Don't warn about this because it occurs in pod2man and would
544          * cause considerable (unfixable) warnage.
584                  */
585                 man_node_delete(man, n);
586                 break;
587         default:
588                 break;
589         }
546         if (NULL == n->prev && MAN_ROOT == n->parent->type)
547                 man_node_delete(m, n);

591         return(1);
592 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12616 Wed Jul 30 20:55:08 2014
new/usr/src/cmd/mandoc/mandoc.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: mandoc.c,v 1.74 2013/12/30 18:30:32 schwarze Exp $ */
/*      $Id: mandoc.c,v 1.62 2011/12/03 16:08:51 schwarze Exp $ */
/*
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHORS DISCLAIM ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "mandoc.h"
#include "libmandoc.h"

#define DATESIZE 32

static  int      a2time(time_t *, const char *, const char *);
static  char    *time2a(time_t);
static  int      numescape(const char *);


enum mandoc_esc
mandoc_escape(const char **end, const char **start, int *sz)
/*
 * Pass over recursive numerical expressions.  This context of this
 * function is important: it's only called within character-terminating
 * escapes (e.g., \s[xxxyyy]), so all we need to do is handle initial
 * recursion: we don't care about what's in these blocks.
 * This returns the number of characters skipped or -1 if an error
 * occurs (the caller should bail).
 */
static int
numescape(const char *start)
{
        const char      *local_start;
        int              local_sz;
```



```
        char             term;
        enum mandoc_esc  gly;
        int              i;
        size_t           sz;
        const char      *cp;

        i = 0;

        /* The expression consists of a subexpression. */

        if ('\\' == start[i]) {
                cp = &start[++i];
        /*
         * When the caller doesn't provide return storage,
         * use local storage.
                 * Read past the end of the subexpression.
                 * Bail immediately on errors.
         */
                if (ESCAPE_ERROR == mandoc_escape(&cp, NULL, NULL))
                        return(-1);
                return(i + cp - &start[i]);
        }

        if (NULL == start)
                start = &local_start;
        if (NULL == sz)
                sz = &local_sz;
        if ('(' != start[i++])
                return(0);

        /*
         * Beyond the backslash, at least one input character
         * is part of the escape sequence.  With one exception
         * (see below), that character won't be returned.
         * A parenthesised subexpression.  Read until the closing
         * parenthesis, making sure to handle any nested subexpressions
         * that might ruin our parse.
         */

        while (')' != start[i]) {
                sz = strcspn(&start[i], ")\\");
                i += (int)sz;

                if ('\0' == start[i])
                        return(-1);
                else if ('\\' != start[i])
                        continue;

                cp = &start[++i];
                if (ESCAPE_ERROR == mandoc_escape(&cp, NULL, NULL))
                        return(-1);
                i += cp - &start[i];
        }

        /* Read past the terminating ')'. */
        return(++i);
}

enum mandoc_esc
mandoc_escape(const char **end, const char **start, int *sz)
{
        char             c, term, numeric;
        int              i, lim, ssz, rlim;
        const char      *cp, *rstart;
        enum mandoc_esc  gly;
```

```
        cp = *end;
        rstart = cp;
        if (start)
                *start = rstart;
        i = lim = 0;
        gly = ESCAPE_ERROR;
        *start = ++*end;
        *sz = 0;
        term = '\0';
        term = numeric = '\0';

        switch ((*start)[-1]) {
        switch ((c = cp[i++])) {
        /*
         * First the glyphs.  There are several different forms of
         * these, but each eventually returns a substring of the glyph
         * name.
         */
        case ('('):
                gly = ESCAPE_SPECIAL;
                *sz = 2;
                lim = 2;
                break;
        case ('['):
                gly = ESCAPE_SPECIAL;
                /*
                 * Unicode escapes are defined in groff as \[uXXXX] to
                 * \[u10FFFF], where the contained value must be a valid
                 * Unicode codepoint.  Here, however, only check whether
                 * it's not a zero-width escape.
                 */
                if ('u' == (*start)[0] && ']' != (*start)[1])
                if ('u' == cp[i] && ']' != cp[i + 1])
                        gly = ESCAPE_UNICODE;
                term = ']';
                break;
        case ('C'):
                if ('\'' != **start)
                if ('\'' != cp[i])
                        return(ESCAPE_ERROR);
                *start = ++*end;
                if ('u' == (*start)[0] && '\'' != (*start)[1])
                        gly = ESCAPE_UNICODE;
                else
                        gly = ESCAPE_SPECIAL;
                term = '\'';
                break;

        /*
         * Escapes taking no arguments at all.
         */
        case ('d'):
                /* FALLTHROUGH */
        case ('u'):
                return(ESCAPE_IGNORE);

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

        /*
```



```
         * Handle all triggers matching \X(xy, \Xx, and \X[xxxx], where
         * 'X' is the trigger.  These have opaque sub-strings.
         */
        case ('F'):
                /* FALLTHROUGH */
        case ('g'):
                /* FALLTHROUGH */
        case ('k'):
                /* FALLTHROUGH */
        case ('M'):
                /* FALLTHROUGH */
        case ('m'):
                /* FALLTHROUGH */
        case ('n'):
                /* FALLTHROUGH */
        case ('V'):
                /* FALLTHROUGH */
        case ('Y'):
                gly = ESCAPE_IGNORE;
                /* FALLTHROUGH */
        case ('f'):
                if (ESCAPE_ERROR == gly)
                        gly = ESCAPE_FONT;
                switch (**start) {

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

                switch (cp[i++]) {
                case ('('):
                        *start = ++*end;
                        *sz = 2;
                        lim = 2;
                        break;
                case ('['):
                        *start = ++*end;
                        term = ']';
                        break;
                default:
                        *sz = 1;
                        lim = 1;
                        i--;
                        break;
                }
                break;

        /*
         * These escapes are of the form \X'Y', where 'X' is the trigger
         * and 'Y' is any string.  These have opaque sub-strings.
         */
        case ('A'):
                /* FALLTHROUGH */
        case ('b'):
                /* FALLTHROUGH */
        case ('B'):
                /* FALLTHROUGH */
        case ('D'):
                /* FALLTHROUGH */
        case ('o'):
                /* FALLTHROUGH */
        case ('R'):
                /* FALLTHROUGH */
        case ('w'):
                /* FALLTHROUGH */
        case ('X'):
```

```
                /* FALLTHROUGH */
        case ('Z'):
                if ('\'' != **start)
                if ('\'' != cp[i++])
                        return(ESCAPE_ERROR);
                gly = ESCAPE_IGNORE;
                *start = ++*end;
                term = '\'';
                break;

        /*
         * These escapes are of the form \X'N', where 'X' is the trigger
         * and 'N' resolves to a numerical expression.
         */
        case ('B'):
                /* FALLTHROUGH */
        case ('h'):
                /* FALLTHROUGH */
        case ('H'):
                /* FALLTHROUGH */
        case ('L'):
                /* FALLTHROUGH */
        case ('l'):
                gly = ESCAPE_NUMBERED;
                /* FALLTHROUGH */
        case ('S'):
                /* FALLTHROUGH */
        case ('v'):
                /* FALLTHROUGH */
        case ('w'):
                /* FALLTHROUGH */
        case ('x'):
                if ('\'' != **start)
                        return(ESCAPE_ERROR);
                if (ESCAPE_ERROR == gly)
                gly = ESCAPE_IGNORE;
                *start = ++*end;
                term = '\'';
                if ('\'' != cp[i++])
                        return(ESCAPE_ERROR);
                term = numeric = '\'';
                break;

        /*
         * Special handling for the numbered character escape.
         * XXX Do any other escapes need similar handling?
         */
        case ('N'):
                if ('\0' == **start)
                if ('\0' == cp[i])
                        return(ESCAPE_ERROR);
                (*end)++;
                if (isdigit((unsigned char)**start)) {
                        *sz = 1;
                *end = &cp[++i];
                if (isdigit((unsigned char)cp[i-1]))
                        return(ESCAPE_IGNORE);
                }
                (*start)++;
                while (isdigit((unsigned char)**end))
                        (*end)++;
                *sz = *end - *start;
                if (start)
                        *start = &cp[i];
                if (sz)
                        *sz = *end - &cp[i];
```



```
                if ('\0' != **end)
                        (*end)++;
                return(ESCAPE_NUMBERED);

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

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

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

                switch (**end) {
                switch (cp[i++]) {
                case ('('):
                        *start = ++*end;
                        *sz = 2;
                        lim = 2;
                        break;
                case ('['):
                        *start = ++*end;
                        term = ']';
                        term = numeric = ']';
                        break;
                case ('\''):
                        *start = ++*end;
                        term = '\'';
                        term = numeric = '\'';
                        break;
                default:
                        *sz = 1;
                        lim = 1;
                        i--;
                        break;
                }

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

                break;

        /*
         * Anything else is assumed to be a glyph.
         * In this case, pass back the character after the backslash.
         */
        default:
                gly = ESCAPE_SPECIAL;
                *start = --*end;
                *sz = 1;
                lim = 1;
                i--;
                break;
        }

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

326         i += ssz;
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 ('\\'):

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.
373                  * Pretend that the constant-width font modes are the
374                  * same as the regular font modes.
313                                  */
314                                 (*start)++;
315                                 (*sz)--;
316                         } else {
317                                 if ('B' == (*start)[0] && 'I' == (*start)[1])
318                                         gly = ESCAPE_FONTBI;
376                 if (2 == rlim && 'C' == *rstart)
377                         rstart++;
378                 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.
487  * Null-terminate the argument in place.
429  * Collapse pairs of quotes inside quoted arguments.
430  * Advance the argument pointer to the next argument,
431  * or to the NUL byte terminating the argument line.
490  * or to the null byte terminating the argument line.
432  */
433 char *
434 mandoc_getarg(struct mparse *parse, char **cpp, int ln, int *pos)
435 {
436         char     *start, *cp;
437         int       quoted, pairs, white;

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

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

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

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



```
475                                 break;
476                         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         }

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

502         /* NUL-terminate this argument and move to the next one. */
543         /* 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;

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

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;

690         pos = *ppos;

692         if ('\\' == cp[pos] && '.' == cp[pos + 1])
693                 pos += 2;
694         else if ('.' == cp[pos] || '\'' == cp[pos])
695                 pos++;
696         else
697                 return(0);

699         while (' ' == cp[pos] || '\t' == cp[pos])
```

```
700                 pos++;

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 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mandoc.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

21 #define ASCII_NBRSP      31  /* non-breaking space */
22 #define ASCII_HYPH       30  /* breakable hyphen */

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 };

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,

48         MANDOCERR_WARNING, /* ===== start of warnings ===== */

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 */
64         MANDOCERR_NONAME, /* manual name not yet set */
66         MANDOCERR_SECOOO, /* sections out of conventional order */
67         MANDOCERR_SECREP, /* duplicate section name */
68         MANDOCERR_SECMSEC, /* section header suited to sections ... */
67         MANDOCERR_SECMSEC, /* section not in conventional manual section */

70         /* related to macros and nesting */
71         MANDOCERR_MACROOBS, /* skipping obsolete macro */
72         MANDOCERR_IGNPAR, /* skipping paragraph macro */
73         MANDOCERR_MOVEPAR, /* moving paragraph macro out of list */
74         MANDOCERR_IGNNS, /* skipping no-space macro */
75         MANDOCERR_SCOPENEST, /* blocks badly nested */
76         MANDOCERR_CHILD, /* child violates parent syntax */
77         MANDOCERR_NESTEDDISP, /* nested displays are not portable */
78         MANDOCERR_SCOPEREP, /* already in literal mode */
79         MANDOCERR_LINESCOPE, /* line scope broken */

81         /* related to missing macro arguments */
82         MANDOCERR_MACROEMPTY, /* skipping empty macro */
83         MANDOCERR_ARGCWARN, /* argument count wrong */
84         MANDOCERR_DISPTYPE, /* missing display type */
85         MANDOCERR_LISTFIRST, /* list type must come first */
86         MANDOCERR_NOWIDTHARG, /* tag lists require a width argument */
87         MANDOCERR_FONTTYPE, /* missing font type */
88         MANDOCERR_WNOSCOPE, /* skipping end of block that is not open */

90         /* related to bad macro arguments */
91         MANDOCERR_IGNARGV, /* skipping argument */
92         MANDOCERR_ARGVREP, /* duplicate argument */
93         MANDOCERR_DISPREP, /* duplicate display type */
94         MANDOCERR_LISTREP, /* duplicate list type */
95         MANDOCERR_BADATT, /* unknown AT&T UNIX version */
96         MANDOCERR_BADBOOL, /* bad Boolean value */
97         MANDOCERR_BADFONT, /* unknown font */
98         MANDOCERR_BADSTANDARD, /* unknown standard specifier */
99         MANDOCERR_BADWIDTH, /* bad width argument */

101        /* related to plain text */
102        MANDOCERR_NOBLANKLN, /* blank line in non-literal context */
103        MANDOCERR_BADTAB, /* tab in non-literal context */
104        MANDOCERR_EOLNSPACE, /* end of line whitespace */
105        MANDOCERR_BADCOMMENT, /* bad comment style */
106        MANDOCERR_BADESCAPE, /* unknown escape sequence */
107        MANDOCERR_BADQUOTE, /* unterminated quoted string */

109        /* related to equations */
110        MANDOCERR_EQNQUOTE, /* unexpected literal in equation */

112        MANDOCERR_ERROR, /* ===== start of errors ===== */

114        /* related to equations */
115        MANDOCERR_EQNNSCOPE, /* unexpected equation scope closure*/
116        MANDOCERR_EQNSCOPE, /* equation scope open on exit */
117        MANDOCERR_EQNBADSCOPE, /* overlapping equation scopes */
118        MANDOCERR_EQNEOF, /* unexpected end of equation */
119        MANDOCERR_EQNSYNT, /* equation syntax error */

121        /* related to tables */
122        MANDOCERR_TBL, /* bad table syntax */
```

```
123         MANDOCERR_TBLOPT, /* bad table option */
124         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_TBLBLOCK, /* 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_ARGSLOST, /* 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 {
163 struct  tbl {
169         char              tab; /* cell-separator */
170         char              decimal; /* decimal point */
171         int               linesize;
172         int               opts;
173 #define TBL_OPT_CENTRE   (1 << 0)
174 #define TBL_OPT_EXPAND   (1 << 1)
175 #define TBL_OPT_BOX      (1 << 2)
176 #define TBL_OPT_DBOX     (1 << 3)
177 #define TBL_OPT_ALLBOX   (1 << 4)
178 #define TBL_OPT_NOKEEP   (1 << 5)
179 #define TBL_OPT_NOSPACE  (1 << 6)
180         int               cols; /* number of columns */
181 };

178 enum    tbl_headt {
179         TBL_HEAD_DATA, /* plug in data from tbl_dat */
180         TBL_HEAD_VERT, /* vertical spacer */
181         TBL_HEAD_DVERT  /* double-vertical spacer */
182 };
```



```
183 /*
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 {
190         enum tbl_headt    pos;
189         int               ident; /* 0 <= unique id < cols */
190         int               vert; /* width of preceding vertical line */
191         struct tbl_head  *next;
192         struct tbl_head  *prev;
193 };

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, /* = */
206         TBL_CELL_VERT, /* | */
207         TBL_CELL_DVERT, /* || */
205         TBL_CELL_MAX
206 };

208 /*
209  * A cell in a layout row.
210  */
211 struct  tbl_cell {
212         struct tbl_cell  *next;
213         int               vert; /* width of preceding vertical line */
214         enum tbl_cellt    pos;
215         size_t            spacing;
216         int               flags;
217 #define TBL_CELL_TALIGN  (1 << 0) /* t, T */
218 #define TBL_CELL_BALIGN  (1 << 1) /* d, D */
219 #define TBL_CELL_BOLD    (1 << 2) /* fB, B, b */
220 #define TBL_CELL_ITALIC  (1 << 3) /* fI, I, i */
221 #define TBL_CELL_EQUAL   (1 << 4) /* e, E */
222 #define TBL_CELL_UP      (1 << 5) /* u, U */
223 #define TBL_CELL_WIGN    (1 << 6) /* z, Z */
224         struct tbl_head  *head;
225 };
______unchanged_portion_omitted_

263 /*
264  * A row of data in a table.
265  */
266 struct  tbl_span {
267         struct tbl_opts  *opts;
269         struct tbl       *tbl;
268         struct tbl_head  *head;
269         struct tbl_row   *layout; /* layout row */
270         struct tbl_dat   *first;
271         struct tbl_dat   *last;
272         int               line; /* parse line */
273         int               flags;
274 #define TBL_SPAN_FIRST   (1 << 0)
275 #define TBL_SPAN_LAST    (1 << 1)
276         enum tbl_spant    pos;
277         struct tbl_span  *next;
278 };
______unchanged_portion_omitted_
```

```
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 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mdoc.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

22 #include <sys/types.h>

24 #include <assert.h>
25 #include <stdarg.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29 #include <time.h>

31 #include "mdoc.h"
32 #include "mandoc.h"
33 #include "libmdoc.h"
34 #include "libmandoc.h"

36 const   char *const __mdoc_macronames[MDOC_MAX] = {
37         "Ap",           "Dd",           "Dt",           "Os",
38         "Sh",           "Ss",           "Pp",           "D1",
39         "Dl",           "Bd",           "Ed",           "Bl",
40         "El",           "It",           "Ad",           "An",
41         "Ar",           "Cd",           "Cm",           "Dv",
42         "Er",           "Ev",           "Ex",           "Fa",
43         "Fd",           "Fl",           "Fn",           "Ft",
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",           "Db",           "Dc",
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         };

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         };

88 const   char * const *mdoc_macronames = __mdoc_macronames;
89 const   char * const *mdoc_argnames = __mdoc_argnames;

91 static  void              mdoc_node_free(struct mdoc_node *);
92 static  void              mdoc_node_unlink(struct mdoc *,
93                                 struct mdoc_node *);
94 static  void              mdoc_free1(struct mdoc *);
95 static  void              mdoc_alloc1(struct mdoc *);
96 static  struct mdoc_node *node_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);

106 const struct mdoc_node *
107 mdoc_node(const struct mdoc *mdoc)
107 mdoc_node(const struct mdoc *m)
108 {

110         assert( ! (MDOC_HALT & mdoc->flags));
111         return(mdoc->first);
110         assert( ! (MDOC_HALT & m->flags));
111         return(m->first);
112 }


115 const struct mdoc_meta *
116 mdoc_meta(const struct mdoc *mdoc)
116 mdoc_meta(const struct mdoc *m)
117 {

119         assert( ! (MDOC_HALT & mdoc->flags));
```

```
120         return(&mdoc->meta);
119         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)
200 mdoc_alloc(struct roff *roff, struct mparse *parse)
201 {
202         struct mdoc     *p;

204         p = mandoc_calloc(1, sizeof(struct mdoc));

206         p->parse = parse;
207         p->defos = defos;
208         p->roff = roff;

210         mdoc_hash_init();
211         mdoc_alloc1(p);
212         return(p);
213 }


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)
220 mdoc_endparse(struct mdoc *m)
222 {

224         assert( ! (MDOC_HALT & mdoc->flags));
225         if (mdoc_macroend(mdoc))
223         assert( ! (MDOC_HALT & m->flags));
224         if (mdoc_macroend(m))
226                 return(1);
227         mdoc->flags |= MDOC_HALT;
226         m->flags |= MDOC_HALT;
228         return(0);
229 }

231 int
232 mdoc_addeqn(struct mdoc *mdoc, const struct eqn *ep)
231 mdoc_addeqn(struct mdoc *m, const struct eqn *ep)
233 {
234         struct mdoc_node *n;

236         assert( ! (MDOC_HALT & mdoc->flags));
235         assert( ! (MDOC_HALT & m->flags));

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

240         if (SEC_NONE == mdoc->lastnamed) {
241                 mdoc_pmsg(mdoc, ep->ln, ep->pos, MANDOCERR_NOTEXT);
239         if (SEC_NONE == m->lastnamed) {
240                 mdoc_pmsg(m, ep->ln, ep->pos, MANDOCERR_NOTEXT);
242                 return(1);
243         }

245         n = node_alloc(mdoc, ep->ln, ep->pos, MDOC_MAX, MDOC_EQN);
```



```
244         n = node_alloc(m, ep->ln, ep->pos, MDOC_MAX, MDOC_EQN);
246         n->eqn = ep;

248         if ( ! node_append(mdoc, n))
247         if ( ! node_append(m, n))
249                 return(0);

251         mdoc->next = MDOC_NEXT_SIBLING;
250         m->next = MDOC_NEXT_SIBLING;
252         return(1);
253 }

255 int
256 mdoc_addspan(struct mdoc *mdoc, const struct tbl_span *sp)
255 mdoc_addspan(struct mdoc *m, const struct tbl_span *sp)
257 {
258         struct mdoc_node *n;

260         assert( ! (MDOC_HALT & mdoc->flags));
259         assert( ! (MDOC_HALT & m->flags));

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

264         if (SEC_NONE == mdoc->lastnamed) {
265                 mdoc_pmsg(mdoc, sp->line, 0, MANDOCERR_NOTEXT);
263         if (SEC_NONE == m->lastnamed) {
264                 mdoc_pmsg(m, sp->line, 0, MANDOCERR_NOTEXT);
266                 return(1);
267         }

269         n = node_alloc(mdoc, sp->line, 0, MDOC_MAX, MDOC_TBL);
268         n = node_alloc(m, sp->line, 0, MDOC_MAX, MDOC_TBL);
270         n->span = sp;

272         if ( ! node_append(mdoc, n))
271         if ( ! node_append(m, n))
273                 return(0);

275         mdoc->next = MDOC_NEXT_SIBLING;
274         m->next = MDOC_NEXT_SIBLING;
276         return(1);
277 }


280 /*
281  * Main parse routine.  Parses a single line -- really just hands off to
282  * the macro (mdoc_pmacro()) or text parser (mdoc_ptext()).
283  */
284 int
285 mdoc_parseln(struct mdoc *mdoc, int ln, char *buf, int offs)
284 mdoc_parseln(struct mdoc *m, int ln, char *buf, int offs)
286 {

288         assert( ! (MDOC_HALT & mdoc->flags));
287         assert( ! (MDOC_HALT & m->flags));

290         mdoc->flags |= MDOC_NEWLINE;
289         m->flags |= MDOC_NEWLINE;

292         /*
293          * Let the roff nS register switch SYNOPSIS mode early,
294          * such that the parser knows at all times
295          * whether this mode is on or off.
296          * Note that this mode is also switched by the Sh macro.
297          */
298         if (roff_getreg(mdoc->roff, "nS"))
```

```
299                     mdoc->flags |= MDOC_SYNOPSIS;
297             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;
338         }

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

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

441         /* Flag analysis. */

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

451         return(p);
452 }


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

460         p = node_alloc(mdoc, line, pos, tok, MDOC_TAIL);
461         if ( ! node_append(mdoc, p))
458         p = node_alloc(m, line, pos, tok, MDOC_TAIL);
459         if ( ! node_append(m, p))
462                 return(0);
463         mdoc->next = MDOC_NEXT_CHILD;
```



```
461         m->next = MDOC_NEXT_CHILD;
464         return(1);
465 }


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

473         assert(mdoc->first);
474         assert(mdoc->last);
471         assert(m->first);
472         assert(m->last);

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


484 int
485 mdoc_body_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok)
483 mdoc_body_alloc(struct mdoc *m, int line, int pos, enum mdoct tok)
486 {
487         struct mdoc_node *p;

489         p = node_alloc(mdoc, line, pos, tok, MDOC_BODY);
490         if ( ! node_append(mdoc, p))
487         p = node_alloc(m, line, pos, tok, MDOC_BODY);
488         if ( ! node_append(m, p))
491                 return(0);
492         mdoc->next = MDOC_NEXT_CHILD;
490         m->next = MDOC_NEXT_CHILD;
493         return(1);
494 }


497 int
498 mdoc_endbody_alloc(struct mdoc *mdoc, int line, int pos, enum mdoct tok,
496 mdoc_endbody_alloc(struct mdoc *m, int line, int pos, enum mdoct tok,
499                 struct mdoc_node *body, enum mdoc_endbody end)
500 {
501         struct mdoc_node *p;

503         p = node_alloc(mdoc, line, pos, tok, MDOC_BODY);
501         p = node_alloc(m, line, pos, tok, MDOC_BODY);
504         p->pending = body;
505         p->norm = body->norm;
506         p->end = end;
507         if ( ! node_append(mdoc, p))
504         if ( ! node_append(m, p))
508                 return(0);
509         mdoc->next = MDOC_NEXT_SIBLING;
506         m->next = MDOC_NEXT_SIBLING;
510         return(1);
511 }


514 int
```

```
mdoc_block_alloc(struct mdoc *mdoc, int line, int pos,
mdoc_block_alloc(struct mdoc *m, int line, int pos,
                enum mdoct tok, struct mdoc_arg *args)
{
        struct mdoc_node *p;

        p = node_alloc(mdoc, line, pos, tok, MDOC_BLOCK);
        p = node_alloc(m, line, pos, tok, MDOC_BLOCK);
        p->args = args;
        if (p->args)
                (args->refcnt)++;

        switch (tok) {
        case (MDOC_Bd):
                /* FALLTHROUGH */
        case (MDOC_Bf):
                /* FALLTHROUGH */
        case (MDOC_Bl):
                /* FALLTHROUGH */
        case (MDOC_Rs):
                p->norm = mandoc_calloc(1, sizeof(union mdoc_data));
                break;
        default:
                break;
        }

        if ( ! node_append(mdoc, p))
        if ( ! node_append(m, p))
                return(0);
        mdoc->next = MDOC_NEXT_CHILD;
        m->next = MDOC_NEXT_CHILD;
        return(1);
}


int
mdoc_elem_alloc(struct mdoc *mdoc, int line, int pos,
mdoc_elem_alloc(struct mdoc *m, int line, int pos,
                enum mdoct tok, struct mdoc_arg *args)
{
        struct mdoc_node *p;

        p = node_alloc(mdoc, line, pos, tok, MDOC_ELEM);
        p = node_alloc(m, line, pos, tok, MDOC_ELEM);
        p->args = args;
        if (p->args)
                (args->refcnt)++;

        switch (tok) {
        case (MDOC_An):
                p->norm = mandoc_calloc(1, sizeof(union mdoc_data));
                break;
        default:
                break;
        }

        if ( ! node_append(mdoc, p))
        if ( ! node_append(m, p))
                return(0);
        mdoc->next = MDOC_NEXT_CHILD;
        m->next = MDOC_NEXT_CHILD;
        return(1);
}

int
mdoc_word_alloc(struct mdoc *mdoc, int line, int pos, const char *p)
```



```
mdoc_word_alloc(struct mdoc *m, int line, int pos, const char *p)
{
        struct mdoc_node *n;

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

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

        mdoc->next = MDOC_NEXT_SIBLING;
        m->next = MDOC_NEXT_SIBLING;
        return(1);
}

void
mdoc_word_append(struct mdoc *mdoc, const char *p)
{
        struct mdoc_node        *n;
        char                    *addstr, *newstr;

        n = mdoc->last;
        addstr = roff_strdup(mdoc->roff, p);
        if (-1 == asprintf(&newstr, "%s %s", n->string, addstr)) {
                perror(NULL);
                exit((int)MANDOCLEVEL_SYSERR);
        }
        free(addstr);
        free(n->string);
        n->string = newstr;
        mdoc->next = MDOC_NEXT_SIBLING;
}

static void
mdoc_node_free(struct mdoc_node *p)
{

        if (MDOC_BLOCK == p->type || MDOC_ELEM == p->type)
                free(p->norm);
        if (p->string)
                free(p->string);
        if (p->args)
                mdoc_argv_free(p->args);
        free(p);
}


static void
mdoc_node_unlink(struct mdoc *mdoc, struct mdoc_node *n)
mdoc_node_unlink(struct mdoc *m, struct mdoc_node *n)
{

        /* Adjust siblings. */

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

        /* Adjust parent. */

        if (n->parent) {
                n->parent->nchild--;
```

```
633                 if (n->parent->child == n)
634                         n->parent->child = n->prev ? n->prev : n->next;
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) {
621         if (m && m->last == n) {
642                 if (n->prev) {
643                         mdoc->last = n->prev;
644                         mdoc->next = MDOC_NEXT_SIBLING;
623                         m->last = n->prev;
624                         m->next = MDOC_NEXT_SIBLING;
645                 } else {
646                         mdoc->last = n->parent;
647                         mdoc->next = MDOC_NEXT_CHILD;
626                         m->last = n->parent;
627                         m->next = MDOC_NEXT_CHILD;
648                 }
649         }

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


656 void
657 mdoc_node_delete(struct mdoc *mdoc, struct mdoc_node *p)
637 mdoc_node_delete(struct mdoc *m, struct mdoc_node *p)
658 {

660         while (p->child) {
661                 assert(p->nchild);
662                 mdoc_node_delete(mdoc, p->child);
642                 mdoc_node_delete(m, p->child);
663         }
664         assert(0 == p->nchild);

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

670 int
671 mdoc_node_relink(struct mdoc *mdoc, struct mdoc_node *p)
672 {

674         mdoc_node_unlink(mdoc, p);
675         return(node_append(mdoc, p));
676 }

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)
661 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)))
669                 if ('\0' != (delim = roff_eqndelim(m->roff)))
698                         if (NULL != (start = strchr(buf + offs, delim)))
699                                 *start++ = '\0';

701                 /* Parse text as normal. */
702                 if ( ! mdoc_ptext(mdoc, line, buf, offs))
674                 if ( ! mdoc_ptext(m, line, buf, offs))
703                         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);
691                 roff_openeqn(m->roff, NULL, line, offs, buf);

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

728         return(1);
729 }
730 #endif

732 /*
733  * Parse free-form text, that is, a line that does not begin with the
734  * control character.
735  */
736 static int
737 mdoc_ptext(struct mdoc *mdoc, int line, char *buf, int offs)
709 mdoc_ptext(struct mdoc *m, int line, char *buf, int offs)
738 {
739         char             *c, *ws, *end;
740         struct mdoc_node *n;

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

744         if (SEC_NONE == mdoc->lastnamed) {
745                 mdoc_pmsg(mdoc, line, offs, MANDOCERR_NOTEXT);
716         if (SEC_NONE == m->lastnamed) {
717                 mdoc_pmsg(m, line, offs, MANDOCERR_NOTEXT);
```

```
746                 return(1);
747         }

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));
734                 m->flags |= MDOC_FREECOL;
735                 return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
764         }

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));
743                 m->flags |= MDOC_FREECOL;
744                 return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
773         }

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)
772                         if (MDOC_LITERAL & m->flags)
801                                 continue;
802                         break;
803                 case '\\':
804                         /* Skip the escaped character, too, if any. */
```



```
805                         if (c[1])
806                                 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);
842                 mdoc_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
872                 return(1);
873         } else if ('\0' == buf[offs])
874                 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,
864                 mandoc_vmsg(MANDOCERR_MACRO, m->parse,
894                                 ln, sv, "%s", buf + sv - 1);
895                 return(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);
885                 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))
892         if (NULL == m->last || MDOC_It == tok || MDOC_El == tok) {
```



```
893                 if ( ! mdoc_macro(m, tok, ln, sv, &offs, buf))
923                         goto err;
924                 return(1);
925         }

927         n = mdoc->last;
928         assert(mdoc->last);
898         n = m->last;
899         assert(m->last);

930         /*
931          * If the first macro of a 'Bl -column', open an 'It' block
932          * context around the parsed macro.
933          */

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))
908                 m->flags |= MDOC_FREECOL;
909                 if ( ! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
939                         goto err;
940                 return(1);
941         }

943         /*
944          * If we're following a block-level 'It' within a 'Bl -column'
945          * context (perhaps opened in the above block or in ptext()),
946          * then open an 'It' block context around the parsed macro.
947          */

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))
924                 m->flags |= MDOC_FREECOL;
925                 if ( ! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
955                         goto err;
956                 return(1);
957         }

959         /* Normal processing of a macro. */

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

964         return(1);

966 err:    /* Error out. */

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

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

976         if ('\0' == p[0])
977                 return(DELIM_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"))
 983         if (0 == strcmp(p + 1, "*(Ba"))
1013                 return(DELIM_MIDDLE);

1015         return(DELIM_NONE);
1016 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8568 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mdoc.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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_D1,
 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 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mdoc_argv.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
 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

22 #include <sys/types.h>

24 #include <assert.h>
25 #include <stdlib.h>
26 #include <stdio.h>
27 #include <string.h>

29 #include "mdoc.h"
30 #include "mandoc.h"
31 #include "libmdoc.h"
32 #include "libmandoc.h"

34 #define MULTI_STEP       5 /* pre-allocate argument values */
35 #define DELIMSZ          6 /* max possible size of a delimiter */

37 enum    argsflag {
38         ARGSFL_NONE = 0,
39         ARGSFL_DELIM, /* handle delimiters of [[::delim::][ ]+]+ */
40         ARGSFL_TABSEP /* handle tab/'Ta' separated phrases */
41 };

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) */
45         ARGV_MULTI, /* multiple args (e.g., -column xxx yyy) */
46         ARGV_OPT_SINGLE /* optional arg (e.g., -offset [xxx]) */
47 };
```

_____*unchanged_portion_omitted_*

```
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_checkpunct(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 *);
60 static  int              argv_single(struct mdoc *, int,
61                               struct mdoc_argv *, int *, char *);

63 static  const enum argvflag argvflags[MDOC_ARG_MAX] = {
64         ARGV_NONE,      /* MDOC_Split */
65         ARGV_NONE,      /* MDOC_Nosplit */
66         ARGV_NONE,      /* MDOC_Ragged */
67         ARGV_NONE,      /* MDOC_Unfilled */
68         ARGV_NONE,      /* MDOC_Literal */
69         ARGV_SINGLE,    /* MDOC_File */
70         ARGV_SINGLE,    /* MDOC_Offset */
72         ARGV_OPT_SINGLE, /* MDOC_Offset */
71         ARGV_NONE,      /* MDOC_Bullet */
72         ARGV_NONE,      /* MDOC_Dash */
73         ARGV_NONE,      /* MDOC_Hyphen */
74         ARGV_NONE,      /* MDOC_Item */
75         ARGV_NONE,      /* MDOC_Enum */
76         ARGV_NONE,      /* MDOC_Tag */
77         ARGV_NONE,      /* MDOC_Diag */
78         ARGV_NONE,      /* MDOC_Hang */
79         ARGV_NONE,      /* MDOC_Ohang */
80         ARGV_NONE,      /* MDOC_Inset */
81         ARGV_MULTI,     /* MDOC_Column */
82         ARGV_SINGLE,    /* MDOC_Width */
84         ARGV_OPT_SINGLE, /* MDOC_Width */
83         ARGV_NONE,      /* MDOC_Compact */
84         ARGV_NONE,      /* MDOC_Std */
85         ARGV_NONE,      /* MDOC_Filled */
86         ARGV_NONE,      /* MDOC_Words */
87         ARGV_NONE,      /* MDOC_Emphasis */
88         ARGV_NONE,      /* MDOC_Symbolic */
89         ARGV_NONE       /* MDOC_Symbolic */
90 };
```

_____*unchanged_portion_omitted_*

```
146 static  const struct mdocarg mdocargs[MDOC_MAX] = {
147         { ARGSFL_DELIM, NULL }, /* Ap */
149         { ARGSFL_NONE, NULL }, /* Ap */
148         { ARGSFL_NONE, NULL }, /* Dd */
149         { ARGSFL_NONE, NULL }, /* Dt */
150         { ARGSFL_NONE, NULL }, /* Os */
151         { ARGSFL_NONE, NULL }, /* Sh */
152         { ARGSFL_NONE, NULL }, /* Ss */
153         { ARGSFL_NONE, NULL }, /* Pp */
154         { ARGSFL_DELIM, NULL }, /* D1 */
155         { ARGSFL_DELIM, NULL }, /* Dl */
156         { ARGSFL_NONE, args_Bd }, /* Bd */
157         { ARGSFL_NONE, NULL }, /* Ed */
158         { ARGSFL_NONE, args_Bl }, /* Bl */
159         { ARGSFL_NONE, NULL }, /* El */
160         { ARGSFL_NONE, NULL }, /* It */
161         { ARGSFL_DELIM, NULL }, /* Ad */
162         { ARGSFL_DELIM, args_An }, /* An */
163         { ARGSFL_DELIM, NULL }, /* Ar */
164         { ARGSFL_DELIM, NULL }, /* Cd */
166         { ARGSFL_NONE, NULL }, /* Cd */
165         { ARGSFL_DELIM, NULL }, /* Cm */
166         { ARGSFL_DELIM, NULL }, /* Dv */
167         { ARGSFL_DELIM, NULL }, /* Er */
168         { ARGSFL_DELIM, NULL }, /* Ev */
169         { ARGSFL_NONE, args_Ex }, /* Ex */
170         { ARGSFL_DELIM, NULL }, /* Fa */
171         { ARGSFL_NONE, NULL }, /* Fd */
172         { ARGSFL_DELIM, NULL }, /* Fl */
173         { ARGSFL_DELIM, NULL }, /* Fn */
```

```
174         { ARGSFL_DELIM, NULL }, /* Ft */
175         { ARGSFL_DELIM, NULL }, /* Ic */
176         { ARGSFL_DELIM, NULL }, /* In */
178         { ARGSFL_NONE, NULL }, /* In */
177         { ARGSFL_DELIM, NULL }, /* Li */
178         { ARGSFL_NONE, NULL }, /* Nd */
179         { ARGSFL_DELIM, NULL }, /* Nm */
180         { ARGSFL_DELIM, NULL }, /* Op */
181         { ARGSFL_NONE, NULL }, /* Ot */
182         { ARGSFL_DELIM, NULL }, /* Pa */
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 */
246         { ARGSFL_NONE, NULL }, /* Fc */
245         { ARGSFL_NONE, NULL }, /* Oo */
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 */
255         { ARGSFL_NONE, NULL }, /* Lb */
254         { ARGSFL_NONE, NULL }, /* Lp */
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 */
265         { ARGSFL_NONE, NULL }, /* Dx */
264         { ARGSFL_NONE, NULL }, /* %Q */
265         { ARGSFL_NONE, NULL }, /* br */
266         { ARGSFL_NONE, NULL }, /* sp */
267         { ARGSFL_NONE, NULL }, /* %U */
268         { ARGSFL_NONE, NULL }, /* Ta */
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,
280 mdoc_argv(struct mdoc *m, int line, enum mdoct tok,
279                 struct mdoc_arg **v, int *pos, char *buf)
280 {
281         char             *p, sv;
282         struct mdoc_argv tmp;
283         struct mdoc_arg  *arg;
284         const enum mdocargt *ap;

286         if ('\0' == buf[*pos])
287                 return(ARGV_EOLN);
288         else if (NULL == (ap = mdocargs[tok].argvs))
289                 return(ARGV_WORD);
290         else if ('-' != buf[*pos])
291                 return(ARGV_WORD);

293         /* Seek to the first unescaped space. */

295         p = &buf[++(*pos)];

297         assert(*pos > 0);

299         for ( ; buf[*pos] ; (*pos)++)
300                 if (' ' == buf[*pos] && '\\' != buf[*pos - 1])
```

```
301                         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))
347                 if ( ! argv_single(m, line, &tmp, pos, buf))
346                         return(ARGV_ERROR);
347                 break;
348         case (ARGV_MULTI):
349                 if ( ! argv_multi(mdoc, line, &tmp, pos, buf))
351                 if ( ! argv_multi(m, line, &tmp, pos, buf))
350                         return(ARGV_ERROR);
351                 break;
354         case (ARGV_OPT_SINGLE):
355                 if ( ! argv_opt_single(m, line, &tmp, pos, buf))
356                         return(ARGV_ERROR);
357                 break;
352         case (ARGV_NONE):
353                 break;
354         }

356         if (NULL == (arg = *v))
357                 arg = *v = mandoc_calloc(1, sizeof(struct mdoc_arg));

359         arg->argc++;
360         arg->argv = mandoc_realloc
```



```
361                 (arg->argv, arg->argc * sizeof(struct mdoc_argv));

363         memcpy(&arg->argv[(int)arg->argc - 1],
364                         &tmp, sizeof(struct mdoc_argv));

366         return(ARGV_ARG);
367 }
_____unchanged_portion_omitted_

409 enum margserr
410 mdoc_zargs(struct mdoc *mdoc, int line, int *pos, char *buf, char **v)
416 mdoc_zargs(struct mdoc *m, int line, int *pos, char *buf, char **v)
411 {

413         return(args(mdoc, line, pos, buf, ARGSFL_NONE, v));
419         return(args(m, line, pos, buf, ARGSFL_NONE, v));
414 }

416 enum margserr
417 mdoc_args(struct mdoc *mdoc, int line, int *pos,
423 mdoc_args(struct mdoc *m, int line, int *pos,
418                 char *buf, enum mdoct tok, char **v)
419 {
420         enum argsflag     fl;
421         struct mdoc_node *n;

423         fl = mdocargs[tok].flags;

425         if (MDOC_It != tok)
426                 return(args(mdoc, line, pos, buf, fl, v));
432                 return(args(m, line, pos, buf, fl, v));

428         /*
429          * We know that we're in an 'It', so it's reasonable to expect
430          * us to be sitting in a 'Bl'.  Someday this may not be the case
431          * (if we allow random 'It's sitting out there), so provide a
432          * safe fall-back into the default behaviour.
433          */

435         for (n = mdoc->last; n; n = n->parent)
441         for (n = m->last; n; n = n->parent)
436                 if (MDOC_Bl == n->tok)
437                         if (LIST_column == n->norm->Bl.type) {
438                                 fl = ARGSFL_TABSEP;
439                                 break;
440                         }

442         return(args(mdoc, line, pos, buf, fl, v));
448         return(args(m, line, pos, buf, fl, v));
443 }

445 static enum margserr
446 args(struct mdoc *mdoc, int line, int *pos,
452 args(struct mdoc *m, int line, int *pos,
447                 char *buf, enum argsflag fl, char **v)
448 {
449         char             *p, *pp;
450         int               pairs;
451         enum margserr     rc;

453         if ('\0' == buf[*pos]) {
454                 if (MDOC_PPHRASE & mdoc->flags)
459                 if (MDOC_PPHRASE & m->flags)
455                         return(ARGS_EOLN);
456                 /*
457                  * If we're not in a partial phrase and the flag for
```

```
458                  * being a phrase literal is still set, the punctuation
459                  * 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 'Bl -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))
492                 if ( ! (MDOC_PHRASELIT & m->flags))
488                         for (pp = *v; ; pp++) {
489                                 if (NULL == (pp = strstr(pp, "Ta")))
490                                         break;
491                                 if (pp > *v && ' ' != *(pp - 1))
492                                         continue;
493                                 if (' ' == *(pp + 2) || '\0' == *(pp + 2))
494                                         break;
495                         }

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))
551                         *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)
568                         if (MDOC_PPHRASE & m->flags)
575                                 return(ARGS_QWORD);
576                         mdoc_pmsg(mdoc, line, *pos, MANDOCERR_BADQUOTE);
570                         mdoc_pmsg(m, line, *pos, MANDOCERR_BADQUOTE);
577                         return(ARGS_QWORD);
578                 }
```

```
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_*

```
**********************************************************
   43931 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mdoc_html.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
   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

  21 #include <sys/types.h>

  23 #include <assert.h>
  24 #include <ctype.h>
  25 #include <stdio.h>
  26 #include <stdlib.h>
  27 #include <string.h>
  28 #include <unistd.h>

  30 #include "mandoc.h"
  31 #include "out.h"
  32 #include "html.h"
  33 #include "mdoc.h"
  34 #include "main.h"

  36 #define INDENT           5

  38 #define MDOC_ARGS         const struct mdoc_meta *meta, \
  38 #define MDOC_ARGS         const struct mdoc_meta *m, \
  39                           const struct mdoc_node *n, \
  40                           struct html *h

  42 #ifndef MIN
  43 #define MIN(a,b)        ((/*CONSTCOND*/(a)<(b))?(a):(b))
  44 #endif

  46 struct  htmlmdoc {
  47         int             (*pre)(MDOC_ARGS);
  48         void            (*post)(MDOC_ARGS);
  49 };
_____unchanged_portion_omitted_

 262 void
 263 html_mdoc(void *arg, const struct mdoc *mdoc)
 263 html_mdoc(void *arg, const struct mdoc *m)
 264 {

 266         print_mdoc(mdoc_meta(mdoc), mdoc_node(mdoc),
```



```
 267                         (struct html *)arg);
 266         print_mdoc(mdoc_meta(m), mdoc_node(m), (struct html *)arg);
 268         putchar('\n');
 269 }
_____unchanged_portion_omitted_


 353 static void
 354 print_mdoc(MDOC_ARGS)
 355 {
 356         struct tag      *t, *tt;
 357         struct htmlpair  tag;

 359         PAIR_CLASS_INIT(&tag, "mandoc");

 361         if ( ! (HTML_FRAGMENT & h->oflags)) {
 362                 print_gen_decls(h);
 363                 t = print_otag(h, TAG_HTML, 0, NULL);
 364                 tt = print_otag(h, TAG_HEAD, 0, NULL);
 365                 print_mdoc_head(meta, n, h);
 364                 print_mdoc_head(m, n, h);
 366                 print_tagq(h, tt);
 367                 print_otag(h, TAG_BODY, 0, NULL);
 368                 print_otag(h, TAG_DIV, 1, &tag);
 369         } else
 370                 t = print_otag(h, TAG_DIV, 1, &tag);

 372         print_mdoc_nodelist(meta, n, h);
 371         print_mdoc_nodelist(m, n, h);
 373         print_tagq(h, t);
 374 }


 377 /* ARGSUSED */
 378 static void
 379 print_mdoc_head(MDOC_ARGS)
 380 {

 382         print_gen_head(h);
 383         bufinit(h);
 384         bufcat_fmt(h, "%s(%s)", meta->title, meta->msec);
 383         bufcat_fmt(h, "%s(%s)", m->title, m->msec);

 386         if (meta->arch)
 387                 bufcat_fmt(h, " (%s)", meta->arch);
 385         if (m->arch)
 386                 bufcat_fmt(h, " (%s)", m->arch);

 389         print_otag(h, TAG_TITLE, 0, NULL);
 390         print_text(h, h->buf);
 391 }


 394 static void
 395 print_mdoc_nodelist(MDOC_ARGS)
 396 {

 398         print_mdoc_node(meta, n, h);
 397         print_mdoc_node(m, n, h);
 399         if (n->next)
 400                 print_mdoc_nodelist(meta, n->next, h);
 399                 print_mdoc_nodelist(m, n->next, h);
 401 }


 404 static void
```

```
print_mdoc_node(MDOC_ARGS)
{
        int              child;
        struct tag      *t;

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

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

                /*
                 * Make sure that if we're in a literal mode already
                 * (i.e., within a <PRE>) don't print the newline.
                 */
                if (' ' == *n->string && MDOC_LINE & n->flags)
                        if ( ! (HTML_LITERAL & h->flags))
                                print_otag(h, TAG_BR, 0, NULL);
                if (MDOC_DELIMC & n->flags)
                        h->flags |= HTML_NOSPACE;
                print_text(h, n->string);
                if (MDOC_DELIMO & n->flags)
                        h->flags |= HTML_NOSPACE;
                return;
        case (MDOC_EQN):
                print_eqn(h, n->eqn);
                break;
        case (MDOC_TBL):
                /*
                 * This will take care of initialising all of the table
                 * state data for the first table, then tearing it down
                 * for the last one.
                 */
                print_tbl(h, n->span);
                return;
        default:
                /*
                 * Close out the current table, if it's open, and unset
                 * the "meta" table state.  This will be reopened on the
                 * next table element.
                 */
                if (h->tblt) {
                        print_tblclose(h);
                        t = h->tags.head;
                }

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

        if (HTML_KEEP & h->flags) {
                if (n->prev ? (n->prev->lastline != n->line) :
                    (n->parent && n->parent->line != n->line)) {
                if (n->prev && n->prev->line != n->line) {
                        h->flags &= ~HTML_KEEP;
                        h->flags |= HTML_PREKEEP;
                } else if (NULL == n->prev) {
```



```
                        if (n->parent && n->parent->line != n->line) {
                                h->flags &= ~HTML_KEEP;
                                h->flags |= HTML_PREKEEP;
                }
        }
        }

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

        print_stagq(h, t);

        switch (n->type) {
        case (MDOC_ROOT):
                mdoc_root_post(meta, n, h);
                mdoc_root_post(m, n, h);
                break;
        case (MDOC_EQN):
                break;
        default:
                if (mdocs[n->tok].post && ENDBODY_NOT == n->end)
                        (*mdocs[n->tok].post)(meta, n, h);
                        (*mdocs[n->tok].post)(m, n, h);
                break;
        }
}

/* ARGSUSED */
static void
mdoc_root_post(MDOC_ARGS)
{
        struct htmlpair  tag[3];
        struct tag      *t, *tt;

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

        print_otag(h, TAG_TBODY, 0, NULL);

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

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

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


/* ARGSUSED */
static int
mdoc_root_pre(MDOC_ARGS)
{
```

```
524         struct htmlpair  tag[3];
525         struct tag      *t, *tt;
526         char             b[BUFSIZ], title[BUFSIZ];

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

530         if (meta->arch) {
533         if (m->arch) {
531                 strlcat(b, " (", BUFSIZ);
532                 strlcat(b, meta->arch, BUFSIZ);
535                 strlcat(b, m->arch, BUFSIZ);
533                 strlcat(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_stagq(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_stagq(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         strlcpy(buf, "list ", BUFSIZ);
1020         strlcat(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.offs) {
1026                 a2offs(n->norm->Bl.offs, &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 }
______unchanged_portion_omitted_


1174 /* ARGSUSED */
1175 static int
1176 mdoc_bd_pre(MDOC_ARGS)
1177 {
1178         struct htmlpair          tag[2];
1179         int                      comp, sv;
1180         const struct mdoc_node  *nn;
1181         struct roffsu            su;

1183         if (MDOC_HEAD == n->type)
1184                 return(0);

1186         if (MDOC_BLOCK == n->type) {
1187                 comp = n->norm->Bd.comp;
1188                 for (nn = n; nn && ! comp; nn = nn->parent) {
1189                         if (MDOC_BLOCK != nn->type)
1190                                 continue;
1191                         if (MDOC_Ss == nn->tok || MDOC_Sh == nn->tok)
1192                                 comp = 1;
1193                         if (nn->prev)
1194                                 break;
1195                 }
1196                 if ( ! comp)
1197                         print_otag(h, TAG_P, 0, NULL);
1198                 return(1);
1199         }

1201         SCALE_HS_INIT(&su, 0);
1202         if (n->norm->Bd.offs)
1203                 a2offs(n->norm->Bd.offs, &su);
1204
1205         bufinit(h);
1206         bufcat_su(h, "margin-left", &su);
1207         PAIR_STYLE_INIT(&tag[0], h);

1209         if (DISP_unfilled != n->norm->Bd.type &&
1210                         DISP_literal != n->norm->Bd.type) {
1211                 PAIR_CLASS_INIT(&tag[1], "display");
1212                 print_otag(h, TAG_DIV, 2, tag);
1213                 return(1);
1214         }

1216         PAIR_CLASS_INIT(&tag[1], "lit display");
1217         print_otag(h, TAG_PRE, 2, tag);

1219         /* This can be recursive: save & set our literal state. */

1221         sv = h->flags & HTML_LITERAL;
1222         h->flags |= HTML_LITERAL;

1224         for (nn = n->child; nn; nn = nn->next) {
1225                 print_mdoc_node(meta, nn, h);
1228                 print_mdoc_node(m, nn, h);
1226                 /*
1227                  * If the printed node flushes its own line, then we
1228                  * needn't do it here as well.  This is hacky, but the
1229                  * notion of selective eoln whitespace is pretty dumb
1230                  * anyway, so don't sweat it.
1231                  */
```

```
1232                 switch (nn->tok) {
1233                 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_D1):
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");
2276                 print_text(h, "\\(aq");
2274                 break;
2275         default:
2276                 abort();
2277                 /* NOTREACHED */
2278         }
2279 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   45476 Wed Jul 30 20:55:09 2014
new/usr/src/cmd/mandoc/mdoc_macro.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
/*      $Id: mdoc_macro.c,v 1.125 2013/12/24 20:45:27 schwarze Exp $ */
/*      $Id: mdoc_macro.c,v 1.115 2012/01/05 00:43:51 schwarze Exp $ */
/*
 * Copyright (c) 2008-2012 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "mdoc.h"
#include "mandoc.h"
#include "libmdoc.h"
#include "libmandoc.h"

enum    rew {   /* see rew_dohalt() */
        REWIND_NONE,
        REWIND_THIS,
        REWIND_MORE,
        REWIND_FORCE,
        REWIND_LATER,
        REWIND_ERROR
};

static  int             blk_full(MACRO_PROT_ARGS);
static  int             blk_exp_close(MACRO_PROT_ARGS);
static  int             blk_part_exp(MACRO_PROT_ARGS);
static  int             blk_part_imp(MACRO_PROT_ARGS);
static  int             ctx_synopsis(MACRO_PROT_ARGS);
static  int             in_line_eoln(MACRO_PROT_ARGS);
static  int             in_line_argn(MACRO_PROT_ARGS);
static  int             in_line(MACRO_PROT_ARGS);
static  int             obsolete(MACRO_PROT_ARGS);
static  int             phrase_ta(MACRO_PROT_ARGS);

static  int             dword(struct mdoc *, int, int, const char *,
                                enum mdelim, int);
static  int             dword(struct mdoc *, int, int,
```



```
                                const char *, enum mdelim);
static  int             append_delims(struct mdoc *,
                                int, int *, char *);
static  enum mdoct      lookup(enum mdoct, const char *);
static  enum mdoct      lookup_raw(const char *);
static  int             make_pending(struct mdoc_node *, enum mdoct,
                                struct mdoc *, int, int);
static  int             phrase(struct mdoc *, int, int, char *);
static  enum mdoct      rew_alt(enum mdoct);
static  enum rew        rew_dohalt(enum mdoct, enum mdoc_type,
                                const struct mdoc_node *);
static  int             rew_elem(struct mdoc *, enum mdoct);
static  int             rew_last(struct mdoc *,
                                const struct mdoc_node *);
static  int             rew_sub(enum mdoc_type, struct mdoc *,
                                enum mdoct, int, int);

const   struct mdoc_macro __mdoc_macros[MDOC_MAX] = {
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ap */
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ap */
        { in_line_eoln, MDOC_PROLOGUE }, /* Dd */
        { in_line_eoln, MDOC_PROLOGUE }, /* Dt */
        { in_line_eoln, MDOC_PROLOGUE }, /* Os */
        { blk_full, MDOC_PARSED | MDOC_JOIN }, /* Sh */
        { blk_full, MDOC_PARSED | MDOC_JOIN }, /* Ss */
        { blk_full, MDOC_PARSED }, /* Sh */
        { blk_full, MDOC_PARSED }, /* Ss */
        { in_line_eoln, 0 }, /* Pp */
        { blk_part_imp, MDOC_PARSED | MDOC_JOIN }, /* D1 */
        { blk_part_imp, MDOC_PARSED | MDOC_JOIN }, /* Dl */
        { blk_part_imp, MDOC_PARSED }, /* D1 */
        { blk_part_imp, MDOC_PARSED }, /* Dl */
        { blk_full, MDOC_EXPLICIT }, /* Bd */
        { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ed */
        { blk_exp_close, MDOC_EXPLICIT }, /* Ed */
        { blk_full, MDOC_EXPLICIT }, /* Bl */
        { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* El */
        { blk_full, MDOC_PARSED | MDOC_JOIN }, /* It */
        { blk_exp_close, MDOC_EXPLICIT }, /* El */
        { blk_full, MDOC_PARSED }, /* It */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ad */
        { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* An */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* An */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ar */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Cd */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Cm */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Dv */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Er */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ev */
        { in_line_eoln, 0 }, /* Ex */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fa */
        { in_line_eoln, 0 }, /* Fd */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fl */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Fn */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ft */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ic */
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* In */
        { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Li */
        { blk_full, MDOC_JOIN }, /* Nd */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Li */
        { blk_full, 0 }, /* Nd */
        { ctx_synopsis, MDOC_CALLABLE | MDOC_PARSED }, /* Nm */
        { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Op */
        { obsolete, 0 }, /* Ot */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Pa */
        { 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 */
114         { in_line_eoln, 0 }, /* %A */
115         { in_line_eoln, 0 }, /* %B */
116         { in_line_eoln, 0 }, /* %D */
117         { in_line_eoln, 0 }, /* %I */
118         { in_line_eoln, 0 }, /* %J */
119         { in_line_eoln, 0 }, /* %N */
120         { in_line_eoln, MDOC_JOIN }, /* %O */
120         { in_line_eoln, 0 }, /* %O */
121         { in_line_eoln, 0 }, /* %P */
122         { in_line_eoln, MDOC_JOIN }, /* %R */
123         { in_line_eoln, MDOC_JOIN }, /* %T */
122         { in_line_eoln, 0 }, /* %R */
123         { in_line_eoln, 0 }, /* %T */
124         { in_line_eoln, 0 }, /* %V */
125         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
126                          MDOC_EXPLICIT | MDOC_JOIN }, /* Ac */
127         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
128                         MDOC_EXPLICIT | MDOC_JOIN }, /* Ao */
129         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Aq */
125         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Ac */
126         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Ao */
127         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Aq */
130         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* At */
131         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
132                          MDOC_EXPLICIT | MDOC_JOIN }, /* Bc */
129         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Bc */
133         { blk_full, MDOC_EXPLICIT }, /* Bf */
134         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
135                         MDOC_EXPLICIT | MDOC_JOIN }, /* Bo */
136         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Bq */
131         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bo */
132         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Bq */
137         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bsx */
138         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bx */
139         { in_line_eoln, 0 }, /* Db */
140         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
141                          MDOC_EXPLICIT | MDOC_JOIN }, /* Dc */
142         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
143                         MDOC_EXPLICIT | MDOC_JOIN }, /* Do */
144         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Dq */
145         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Ec */
146         { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ef */
147         { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Em */
136         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Dc */
137         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Do */
138         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Dq */
139         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Ec */
140         { blk_exp_close, MDOC_EXPLICIT }, /* Ef */
141         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Em */
148         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Eo */
149         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Fx */
150         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ms */
151         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED |
152                         MDOC_IGNDELIM | MDOC_JOIN }, /* No */
153         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED |
154                         MDOC_IGNDELIM | MDOC_JOIN }, /* Ns */
145         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* No */
```



```
146         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Ns */
155         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Nx */
156         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ox */
157         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
158                          MDOC_EXPLICIT | MDOC_JOIN }, /* Pc */
149         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Pc */
159         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Pf */
160         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
161                         MDOC_EXPLICIT | MDOC_JOIN }, /* Po */
162         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Pq */
163         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
164                          MDOC_EXPLICIT | MDOC_JOIN }, /* Qc */
165         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ql */
166         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
167                         MDOC_EXPLICIT | MDOC_JOIN }, /* Qo */
168         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Qq */
169         { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Re */
151         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Po */
152         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Pq */
153         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Qc */
154         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Ql */
155         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Qo */
156         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Qq */
157         { blk_exp_close, MDOC_EXPLICIT }, /* Re */
170         { blk_full, MDOC_EXPLICIT }, /* Rs */
171         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
172                          MDOC_EXPLICIT | MDOC_JOIN }, /* Sc */
173         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
174                         MDOC_EXPLICIT | MDOC_JOIN }, /* So */
175         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Sq */
159         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Sc */
160         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* So */
161         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Sq */
176         { in_line_eoln, 0 }, /* Sm */
177         { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Sx */
178         { in_line, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Sy */
163         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sx */
164         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sy */
179         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Tn */
180         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ux */
166         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ux */
181         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Xc */
182         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Xo */
183         { blk_full, MDOC_EXPLICIT | MDOC_CALLABLE }, /* Fo */
184         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
185                          MDOC_EXPLICIT | MDOC_JOIN }, /* Fc */
186         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
187                         MDOC_EXPLICIT | MDOC_JOIN }, /* Oo */
188         { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
189                          MDOC_EXPLICIT | MDOC_JOIN }, /* Oc */
170         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Fc */
171         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Oo */
172         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Oc */
190         { blk_full, MDOC_EXPLICIT }, /* Bk */
191         { blk_exp_close, MDOC_EXPLICIT | MDOC_JOIN }, /* Ek */
174         { blk_exp_close, MDOC_EXPLICIT }, /* Ek */
192         { in_line_eoln, 0 }, /* Bt */
193         { in_line_eoln, 0 }, /* Hf */
194         { obsolete, 0 }, /* Fr */
195         { in_line_eoln, 0 }, /* Ud */
196         { in_line, 0 }, /* Lb */
197         { in_line_eoln, 0 }, /* Lp */
198         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Lk */
199         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Mt */
200         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Brq */
201         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED |
```

```
                        MDOC_EXPLICIT | MDOC_JOIN }, /* Bro */
        { blk_exp_close, MDOC_CALLABLE | MDOC_PARSED |
                         MDOC_EXPLICIT | MDOC_JOIN }, /* Brc */
        { in_line_eoln, MDOC_JOIN }, /* %C */
        { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Brq */
        { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bro */
        { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Brc *
        { in_line_eoln, 0 }, /* %C */
        { obsolete, 0 }, /* Es */
        { obsolete, 0 }, /* En */
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Dx */
        { in_line_eoln, MDOC_JOIN }, /* %Q */
        { in_line_eoln, 0 }, /* %Q */
        { in_line_eoln, 0 }, /* br */
        { in_line_eoln, 0 }, /* sp */
        { in_line_eoln, 0 }, /* %U */
        { phrase_ta, MDOC_CALLABLE | MDOC_PARSED | MDOC_JOIN }, /* Ta */
        { phrase_ta, MDOC_CALLABLE | MDOC_PARSED }, /* Ta */
};

const   struct mdoc_macro * const mdoc_macros = __mdoc_macros;


/*
 * This is called at the end of parsing.  It must traverse up the tree,
 * closing out open [implicit] scopes.  Obviously, open explicit scopes
 * are errors.
 */
int
mdoc_macroend(struct mdoc *mdoc)
mdoc_macroend(struct mdoc *m)
{
        struct mdoc_node *n;

        /* Scan for open explicit scopes. */

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

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

        /* Rewind to the first. */

        return(rew_last(mdoc, mdoc->first));
        return(rew_last(m, m->first));
}
______unchanged_portion_omitted_


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

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

        /* LINTED */
        while (mdoc->last != to) {
                /*
                 * Save the parent here, because we may delete the
```



```
                 * mdoc->last node in the post-validation phase and reset
                 * it to mdoc->last->parent, causing a step in the closing
                 * m->last node in the post-validation phase and reset
                 * it to m->last->parent, causing a step in the closing
                 * out to be lost.
                 */
                np = mdoc->last->parent;
                if ( ! mdoc_valid_post(mdoc))
                        return(0);
                n = mdoc->last;
                mdoc->last = np;
                assert(mdoc->last);
                mdoc->last->last = n;
        }

        return(mdoc_valid_post(mdoc));
}
______unchanged_portion_omitted_


/*
 * We are trying to close a block identified by tok,
 * but the child block *broken is still open.
 * Thus, postpone closing the tok block
 * until the rew_sub call closing *broken.
 */
static int
make_pending(struct mdoc_node *broken, enum mdoct tok,
                struct mdoc *mdoc, int line, int ppos)
                struct mdoc *m, int line, int ppos)
{
        struct mdoc_node *breaker;

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

                /*
                 * If the *broken block had already been broken before
                 * and we encounter its breaker, make the tok block
                 * pending on the inner breaker.
                 * Graphically, "[A breaker=[B broken=[C->B B] tok=A] C]"
                 * becomes "[A broken=[B [C->B B] tok=A] C]"
                 * and finally "[A [B->A [C->B B] A] C]".
                 */
                if (breaker == broken->pending) {
                        broken = breaker;
                        continue;
                }

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

                /*
                 * Found the breaker.
                 * If another, outer breaker is already pending on
                 * the *broken block, we must not clobber the link
                 * to the outer breaker, but make it pending on the
                 * new, now inner breaker.
                 * Graphically, "[A breaker=[B broken=[C->A A] tok=B] C]"
                 * becomes "[A breaker=[B->A broken=[C A] tok=B] C]"
                 * 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,
518                 mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
539                                 "%s breaks %s", mdoc_macronames[tok],
540                                 mdoc_macronames[broken->tok]);
541                 return(1);
542         }

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

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



```
581                             MDOC_BLOCK != t)
582                                 return(1);
583                         /* FALLTHROUGH */
584                 case (REWIND_ERROR):
585                         mdoc_pmsg(mdoc, line, ppos, MANDOCERR_NOSCOPE);
560                         mdoc_pmsg(m, line, ppos, MANDOCERR_NOSCOPE);
586                         return(1);
587                 }
588                 break;
589         }

591         assert(n);
592         if ( ! rew_last(mdoc, n))
567         if ( ! rew_last(m, n))
593                 return(0);

595         /*
596          * The current block extends an enclosing block.
597          * Now that the current block ends, close the enclosing block, too.
598          */
599         while (NULL != (n = n->pending)) {
600                 if ( ! rew_last(mdoc, n))
575                 if ( ! rew_last(m, n))
601                         return(0);
602                 if (MDOC_HEAD == n->type &&
603                     ! mdoc_body_alloc(mdoc, n->line, n->pos, n->tok))
578                     ! mdoc_body_alloc(m, n->line, n->pos, n->tok))
604                         return(0);
605         }

607         return(1);
608 }

610 /*
611  * Allocate a word and check whether it's punctuation or not.
612  * Punctuation consists of those tokens found in mdoc_isdelim().
613  */
614 static int
615 dword(struct mdoc *mdoc, int line, int col, const char *p,
616                 enum mdelim d, int may_append)
590 dword(struct mdoc *m, int line,
591                 int col, const char *p, enum mdelim d)
617 {
618
619         if (DELIM_MAX == d)
620                 d = mdoc_isdelim(p);

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

630         if ( ! mdoc_word_alloc(mdoc, line, col, p))
597         if ( ! mdoc_word_alloc(m, line, col, p))
631                 return(0);

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

636         /*
637          * Closing delimiters only suppress the preceding space
638          * when they follow something, not when they start a new
```

```
639          * block or element, and not when they follow 'No'.
640          *
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)
741                 if (MDOC_CALLABLE & mdoc_macros[n->tok].flags)
778                         later = n;
779         }

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);
748                         mdoc_pmsg(m, line, ppos, MANDOCERR_ARGSLOST);

786                 if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
750                 if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
787                         return(0);
788                 return(rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos));
752                 return(rew_sub(MDOC_BLOCK, m, tok, line, ppos));
789         }

791         if ( ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
755         if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
792                 return(0);

794         if (NULL == later && maxargs > 0)
795                 if ( ! mdoc_tail_alloc(mdoc, line, ppos, rew_alt(tok)))
759                 if ( ! mdoc_tail_alloc(m, line, ppos, rew_alt(tok)))
796                         return(0);

798         for (flushed = j = 0; ; j++) {
799                 lastarg = *pos;

801                 if (j == maxargs && ! flushed) {
802                         if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
766                         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
803                                 return(0);
804                         flushed = 1;
805                 }

807                 ac = mdoc_args(mdoc, line, pos, buf, tok, &p);
771                 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;

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



```
818                 if (MDOC_MAX == ntok) {
819                         if ( ! dword(mdoc, line, lastarg, p, DELIM_MAX,
820                             MDOC_JOIN & mdoc_macros[tok].flags))
783                         if ( ! dword(m, line, lastarg, p, DELIM_MAX))
821                                 return(0);
822                         continue;
823                 }

825                 if ( ! flushed) {
826                         if ( ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
789                         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
827                                 return(0);
828                         flushed = 1;
829                 }

831                 mdoc->flags &= ~MDOC_NEWLINE;

833                 if ( ! mdoc_macro(mdoc, ntok, line, lastarg, pos, buf))
793                 if ( ! mdoc_macro(m, ntok, line, lastarg, pos, buf))
834                         return(0);
835                 break;
836         }

838         if ( ! flushed && ! rew_sub(MDOC_BLOCK, mdoc, tok, line, ppos))
798         if ( ! flushed && ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
839                 return(0);

841         if ( ! nl)
842                 return(1);
843         return(append_delims(mdoc, line, pos, buf));
803         return(append_delims(m, line, pos, buf));
844 }


847 static int
848 in_line(MACRO_PROT_ARGS)
849 {
850         int              la, scope, cnt, nc, nl;
851         enum margverr    av;
852         enum mdoct       ntok;
853         enum margserr    ac;
854         enum mdelim      d;
855         struct mdoc_arg *arg;
856         char            *p;

858         nl = MDOC_NEWLINE & mdoc->flags;
818         nl = MDOC_NEWLINE & m->flags;

860         /*
861          * Whether we allow ignored elements (those without content,
862          * usually because of reserved words) to squeak by.
863          */

865         switch (tok) {
866         case (MDOC_An):
867                 /* FALLTHROUGH */
868         case (MDOC_Ar):
869                 /* FALLTHROUGH */
870         case (MDOC_Fl):
871                 /* FALLTHROUGH */
872         case (MDOC_Mt):
873                 /* FALLTHROUGH */
874         case (MDOC_Nm):
875                 /* FALLTHROUGH */
876         case (MDOC_Pa):
877                 nc = 1;
```

```
878                 break;
879         default:
880                 nc = 0;
881                 break;
882         }

884         for (arg = NULL;; ) {
885                 la = *pos;
886                 av = mdoc_argv(mdoc, line, tok, &arg, pos, buf);
846                 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);
863                 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))
882                         if (scope && ! rew_elem(m, tok))
923                                 return(0);
924                         if (nc && 0 == cnt) {
925                                 if ( ! mdoc_elem_alloc(mdoc, line,
926                                                 ppos, tok, arg))
885                                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
927                                         return(0);
928                                 if ( ! rew_last(mdoc, mdoc->last))
887                                 if ( ! rew_last(m, m->last))
929                                         return(0);
930                         } else if ( ! nc && 0 == cnt) {
931                                 mdoc_argv_free(arg);
932                                 mdoc_pmsg(mdoc, line, ppos,
933                                         MANDOCERR_MACROEMPTY);
891                                 mdoc_pmsg(m, line, ppos, MANDOCERR_MACROEMPTY);
934                         }

936                         if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
894                         if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
```



```
937                                 return(0);
938                         if ( ! nl)
939                                 return(1);
940                         return(append_delims(mdoc, line, pos, buf));
898                         return(append_delims(m, line, pos, buf));
941                 }

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))
920                                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
964                                         return(0);
965                                 if (MDOC_Ar == tok || MDOC_Li == tok ||
966                                                 MDOC_Fl == tok)
967                                         cnt++;
968                                 scope = 1;
969                         }
970                         /*
971                          * Close out our scope, if one is open, before
972                          * any punctuation.
973                          */
974                         if (scope && ! rew_elem(mdoc, tok))
931                         if (scope && ! rew_elem(m, tok))
975                                 return(0);
976                         scope = 0;
977                 } else if ( ! scope) {
978                         if ( ! mdoc_elem_alloc(mdoc, line, ppos, tok, arg))
935                         if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
979                                 return(0);
980                         scope = 1;
981                 }

983                 if (DELIM_NONE == d)
984                         cnt++;

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

990                 /*
991                  * 'Fl' macros have their scope re-opened with each new
992                  * word so that the '-' can be added to each one without
993                  * having to parse out spaces.
994                  */
995                 if (scope && MDOC_Fl == tok) {
996                         if ( ! rew_elem(mdoc, tok))
952                         if ( ! rew_elem(m, tok))
```

```
 997                                return(0);
 998                        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) {
1327         for (n = m->last; n && n != body && n != blk->parent; n = n->parent) {
1378                 if (MDOC_BLOCK == n->type &&
1379                     MDOC_EXPLICIT & mdoc_macros[n->tok].flags &&
1380                     ! (MDOC_VALID & n->flags)) {
1381                         make_pending(n, tok, mdoc, line, ppos);
1382                         if ( ! mdoc_endbody_alloc(mdoc, line, ppos,
1331                         make_pending(n, tok, m, line, ppos);
1332                         if ( ! mdoc_endbody_alloc(m, line, ppos,
1383                             tok, body, ENDBODY_NOSPACE))
1384                                 return(0);
1385                         return(1);
1386                 }
1387         }

1389         /*
1390          * If we can't rewind to our body, then our scope has already
1391          * been closed by another macro (like 'Oc' closing 'Op').  This
1392          * is ugly behaviour nodding its head to OpenBSD's overwhelming
1393          * crufty use of 'Op' breakage.
1394          */
1395         if (n != body)
1396                 mandoc_vmsg(MANDOCERR_SCOPENEST, mdoc->parse, line, ppos,
1346                 mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
1397                                 "%s broken", mdoc_macronames[tok]);
```



```
1399         if (n && ! rew_sub(MDOC_BODY, mdoc, tok, line, ppos))
1349         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                        continue;
1503                }

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);
1534                 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                 if (ARGS_ERROR == ac)
1703                         return(0);
1704                 if (ARGS_EOLN == ac)
1705                         break;

1707                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1709                 if (MDOC_MAX == ntok) {
1710                         if ( ! dword(mdoc, line, la, p, DELIM_MAX,
1711                             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 margserr    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_Bl != n->tok)
1818                 n = n->parent;
1819         if (NULL == n || LIST_column != n->norm->Bl.type) {
1820                 mdoc_pmsg(mdoc, line, ppos, MANDOCERR_STRAYTA);
1821                 return(1);
1822         }
1754         /*
1755          * FIXME: this is overly restrictive: if the ‘Ta’ is unexpected,
1756          * it should simply error out with ARGSLOST.
1757          */

1824         /* Advance to the next column. */
1825         if ( ! rew_sub(MDOC_BODY, mdoc, MDOC_It, line, ppos))
1759         if ( ! rew_sub(MDOC_BODY, m, MDOC_It, line, ppos))
1826                 return(0);
1827         if ( ! mdoc_body_alloc(mdoc, line, ppos, MDOC_It))
1761         if ( ! mdoc_body_alloc(m, line, ppos, MDOC_It))
1828                 return(0);

1830         for (;;) {
1831                 la = *pos;
1832                 ac = mdoc_zargs(mdoc, line, pos, buf, &p);
1766                 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[tok].flags))
1776                         if ( ! dword(m, line, la, p, DELIM_MAX))
1844                                 return(0);
1845                         continue;
1846                 }

1848                 if ( ! mdoc_macro(mdoc, ntok, line, la, pos, buf))
1781                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1849                         return(0);
1850                 return(append_delims(mdoc, line, pos, buf));
1783                 return(append_delims(m, line, pos, buf));
1851         }

1853         return(1);
1854 }
```

unchanged_portion_omitted_

```
**********************************************************
   33223 Wed Jul 30 20:55:10 2014
new/usr/src/cmd/mandoc/mdoc_man.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
/*      $Id: mdoc_man.c,v 1.57 2013/12/25 22:00:45 schwarze Exp $ */
/*      $Id: mdoc_man.c,v 1.9 2011/10/24 21:47:59 schwarze Exp $ */
/*
 * Copyright (c) 2011, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <stdio.h>
#include <string.h>

#include "mandoc.h"
#include "out.h"
#include "man.h"
#include "mdoc.h"
#include "main.h"

#define DECL_ARGS const struct mdoc_meta *meta, \
                  const struct mdoc_node *n
#define DECL_ARGS const struct mdoc_meta *m, \
                  const struct mdoc_node *n, \
                  struct mman *mm

struct  mman {
        int               need_space; /* next word needs prior ws */
        int               need_nl; /* next word needs prior nl */
};

struct  manact {
        int             (*cond)(DECL_ARGS); /* DON'T run actions */
        int             (*pre)(DECL_ARGS); /* pre-node action */
        void            (*post)(DECL_ARGS); /* post-node action */
        const char       *prefix; /* pre-node string constant */
        const char       *suffix; /* post-node string constant */
};

static  int       cond_body(DECL_ARGS);
static  int       cond_head(DECL_ARGS);
static  void      font_push(char);
static  void      font_pop(void);
static  void      mid_it(void);
static  void      post__t(DECL_ARGS);
static  void      post_bd(DECL_ARGS);
static  void      post_bf(DECL_ARGS);
```



```
static  void      post_bk(DECL_ARGS);
static  void      post_bl(DECL_ARGS);
static  void      post_dl(DECL_ARGS);
static  void      post_enc(DECL_ARGS);
static  void      post_eo(DECL_ARGS);
static  void      post_fa(DECL_ARGS);
static  void      post_fd(DECL_ARGS);
static  void      post_fl(DECL_ARGS);
static  void      post_fn(DECL_ARGS);
static  void      post_fo(DECL_ARGS);
static  void      post_font(DECL_ARGS);
static  void      post_in(DECL_ARGS);
static  void      post_it(DECL_ARGS);
static  void      post_lb(DECL_ARGS);
static  void      post_nm(DECL_ARGS);
static  void      post_percent(DECL_ARGS);
static  void      post_pf(DECL_ARGS);
static  void      post_sect(DECL_ARGS);
static  void      post_sp(DECL_ARGS);
static  void      post_vt(DECL_ARGS);
static  int       pre__t(DECL_ARGS);
static  int       pre_an(DECL_ARGS);
static  int       pre_ap(DECL_ARGS);
static  int       pre_bd(DECL_ARGS);
static  int       pre_bf(DECL_ARGS);
static  int       pre_bk(DECL_ARGS);
static  int       pre_bl(DECL_ARGS);
static  int       pre_br(DECL_ARGS);
static  int       pre_bx(DECL_ARGS);
static  int       pre_dl(DECL_ARGS);
static  int       pre_enc(DECL_ARGS);
static  int       pre_em(DECL_ARGS);
static  int       pre_fa(DECL_ARGS);
static  int       pre_fd(DECL_ARGS);
static  int       pre_fl(DECL_ARGS);
static  int       pre_fn(DECL_ARGS);
static  int       pre_fo(DECL_ARGS);
static  int       pre_ft(DECL_ARGS);
static  int       pre_in(DECL_ARGS);
static  int       pre_it(DECL_ARGS);
static  int       pre_lk(DECL_ARGS);
static  int       pre_li(DECL_ARGS);
static  int       pre_nm(DECL_ARGS);
static  int       pre_no(DECL_ARGS);
static  int       pre_ns(DECL_ARGS);
static  int       pre_pp(DECL_ARGS);
static  int       pre_rs(DECL_ARGS);
static  int       pre_sm(DECL_ARGS);
static  int       pre_sp(DECL_ARGS);
static  int       pre_sect(DECL_ARGS);
static  int       pre_sy(DECL_ARGS);
static  void      pre_syn(const struct mdoc_node *);
static  int       pre_vt(DECL_ARGS);
static  int       pre_ux(DECL_ARGS);
static  int       pre_xr(DECL_ARGS);
static  void      print_word(const char *);
static  void      print_line(const char *, int);
static  void      print_block(const char *, int);
static  void      print_offs(const char *);
static  void      print_width(const char *,
                              const struct mdoc_node *, size_t);
static  void      print_count(int *);
static  void      print_word(struct mman *, const char *);
static  void      print_node(DECL_ARGS);

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 }, /* Dl */
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 }, /* Bl */
127         { NULL, NULL, NULL, NULL, NULL }, /* El */
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 */
```

```
        { NULL, NULL, NULL, NULL, NULL }, /* Re */
        { cond_body, pre_rs, NULL, NULL, NULL }, /* Rs */
        { cond_body, pre_pp, NULL, NULL, NULL }, /* Rs */
        { NULL, NULL, NULL, NULL, NULL }, /* Sc */
        { cond_body, pre_enc, post_enc, "\\(oq", "\\(cq" }, /* So */
        { cond_body, pre_enc, post_enc, "\\(oq", "\\(cq" }, /* Sq */
        { NULL, pre_sm, NULL, NULL, NULL }, /* Sm */
        { NULL, pre_em, post_font, NULL, NULL }, /* Sx */
        { NULL, pre_sy, post_font, NULL, NULL }, /* Sy */
        { NULL, pre_li, post_font, NULL, NULL }, /* Tn */
        { cond_body, pre_enc, post_enc, "‘", "’" }, /* So */
        { cond_body, pre_enc, post_enc, "‘", "’" }, /* Sq */
        { NULL, NULL, NULL, NULL, NULL }, /* _Sm */
        { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Sx */
        { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Sy */
        { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Tn */
        { NULL, pre_ux, NULL, "UNIX", NULL }, /* Ux */
        { NULL, NULL, NULL, NULL, NULL }, /* Xc */
        { NULL, NULL, NULL, NULL, NULL }, /* Xo */
        { NULL, pre_fo, post_fo, NULL, NULL }, /* Fo */
        { NULL, NULL, NULL, NULL, NULL }, /* Fc */
        { NULL, NULL, NULL, NULL, NULL }, /* _Xc */
        { NULL, NULL, NULL, NULL, NULL }, /* _Xo */
        { NULL, NULL, NULL, NULL, NULL }, /* _Fo */
        { NULL, NULL, NULL, NULL, NULL }, /* _Fc */
        { cond_body, pre_enc, post_enc, "[", "]" }, /* Oo */
        { NULL, NULL, NULL, NULL, NULL }, /* Oc */
        { NULL, pre_bk, post_bk, NULL, NULL }, /* Bk */
        { NULL, NULL, NULL, NULL, NULL }, /* Ek */
        { NULL, NULL, NULL, NULL, NULL }, /* _Bk */
        { NULL, NULL, NULL, NULL, NULL }, /* _Ek */
        { NULL, pre_ux, NULL, "is currently in beta test.", NULL }, /* Bt */
        { NULL, NULL, NULL, NULL, NULL }, /* Hf */
        { NULL, NULL, NULL, NULL, NULL }, /* Fr */
        { NULL, pre_ux, NULL, "currently under development.", NULL }, /* Ud */
        { NULL, NULL, post_lb, NULL, NULL }, /* Lb */
        { NULL, NULL, NULL, NULL, NULL }, /* _Lb */
        { NULL, pre_pp, NULL, NULL, NULL }, /* Lp */
        { NULL, pre_lk, NULL, NULL, NULL }, /* Lk */
        { NULL, pre_em, post_font, NULL, NULL }, /* Mt */
        { NULL, NULL, NULL, NULL, NULL }, /* _Lk */
        { NULL, NULL, NULL, NULL, NULL }, /* _Mt */
        { cond_body, pre_enc, post_enc, "{", "}" }, /* Brq */
        { cond_body, pre_enc, post_enc, "{", "}" }, /* Bro */
        { NULL, NULL, NULL, NULL, NULL }, /* Brc */
        { NULL, NULL, post_percent, NULL, NULL }, /* %C */
        { NULL, NULL, NULL, NULL, NULL }, /* Es */
        { NULL, NULL, NULL, NULL, NULL }, /* En */
        { NULL, NULL, NULL, NULL, NULL }, /* _%C */
        { NULL, NULL, NULL, NULL, NULL }, /* _Es */
        { NULL, NULL, NULL, NULL, NULL }, /* _En */
        { NULL, pre_ux, NULL, "DragonFly", NULL }, /* Dx */
        { NULL, NULL, post_percent, NULL, NULL }, /* %Q */
        { NULL, NULL, NULL, NULL, NULL }, /* _%Q */
        { NULL, pre_br, NULL, NULL, NULL }, /* br */
        { NULL, pre_sp, post_sp, NULL, NULL }, /* sp */
        { NULL, NULL, post_percent, NULL, NULL }, /* %U */
        { NULL, NULL, NULL, NULL, NULL }, /* Ta */
        { NULL, NULL, NULL, NULL, NULL }, /* _%U */
        { NULL, NULL, NULL, NULL, NULL }, /* _Ta */
        { NULL, NULL, NULL, NULL, NULL }, /* ROOT */
};

static  int             outflags;
#define MMAN_spc        (1 << 0)  /* blank character before next word */
#define MMAN_spc_force  (1 << 1)  /* even before trailing punctuation */
```



```
#define MMAN_nl         (1 << 2)  /* break man(7) code line */
#define MMAN_br         (1 << 3)  /* break output line */
#define MMAN_sp         (1 << 4)  /* insert a blank output line */
#define MMAN_PP         (1 << 5)  /* reset indentation etc. */
#define MMAN_Sm         (1 << 6)  /* horizontal spacing mode */
#define MMAN_Bk         (1 << 7)  /* word keep mode */
#define MMAN_Bk_susp    (1 << 8)  /* suspend this (after a macro) */
#define MMAN_An_split   (1 << 9)  /* author mode is "split" */
#define MMAN_An_nosplit (1 << 10) /* author mode is "nosplit" */
#define MMAN_PD         (1 << 11) /* inter-paragraph spacing disabled */
#define MMAN_nbrword    (1 << 12) /* do not break the next word */

#define BL_STACK_MAX    32

static  size_t          Bl_stack[BL_STACK_MAX];  /* offsets [chars] */
static  int             Bl_stack_post[BL_STACK_MAX];  /* add final .RE */
static  int             Bl_stack_len;  /* number of nested Bl blocks */
static  int             TPremain;  /* characters before tag is full */

static  struct {
        char    *head;
        char    *tail;
        size_t   size;
}       fontqueue;

static void
font_push(char newfont)
print_word(struct mman *mm, const char *s)
{

        if (fontqueue.head + fontqueue.size <= ++fontqueue.tail) {
                fontqueue.size += 8;
                fontqueue.head = mandoc_realloc(fontqueue.head,
                                fontqueue.size);
        }
        *fontqueue.tail = newfont;
        print_word("");
        printf("\\f");
        putchar(newfont);
        outflags &= ~MMAN_spc;
}

static void
font_pop(void)
{

        if (fontqueue.tail > fontqueue.head)
                fontqueue.tail--;
        outflags &= ~MMAN_spc;
        print_word("");
        printf("\\f");
        putchar(*fontqueue.tail);
}

static void
print_word(const char *s)
{

        if ((MMAN_PP | MMAN_sp | MMAN_br | MMAN_nl) & outflags) {
        if (mm->need_nl) {
                /*
                 * If we need a newline, print it now and start afresh.
                 */
                if (MMAN_PP & outflags) {
                        if (MMAN_sp & outflags) {
                                if (MMAN_PD & outflags) {
```

```
313                                     printf("\n.PD");
314                                     outflags &= ~MMAN_PD;
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) {
214             mm->need_space = 0;
215             mm->need_nl = 0;
216     } else if (mm->need_space && '\0' != s[0])
332             /*
333              * If we need a space, only print it if
334              * (1) it is forced by 'No' or
335              * (2) what follows is not terminating punctuation or
336              * (3) what follows is longer than one character.
218              * If we need a space, only print it before
219              * (1) a nonzero length word;
220              * (2) a word that is non-punctuation; and
221              * (3) if punctuation, non-terminating puncutation.
337              */
338             if (MMAN_spc_force & outflags || '\0' == s[0] ||
339                 NULL == strchr(".,:;)]?!", s[0]) || '\0' != s[1]) {
340                     if (MMAN_Bk & outflags &&
341                         ! (MMAN_Bk_susp & outflags))
342                             putchar('\\');
223             if (NULL == strchr(".,:;)]?!", s[0]) || '\0' != s[1])
343                     putchar(' ');
344                     if (TPremain)
345                             TPremain--;
346             }
347     }

349     /*
350      * Reassign needing space if we're not following opening
351      * punctuation.
352      */
353     if (MMAN_Sm & outflags && ('\0' == s[0] ||
354         (('(' != s[0] && '[' != s[0]) || '\0' != s[1])))
355             outflags |= MMAN_spc;
356     else
357             outflags &= ~MMAN_spc;
358     outflags &= ~(MMAN_spc_force | MMAN_Bk_susp);
230     mm->need_space =
231             ('(' != s[0] && '[' != s[0]) || '\0' != s[1];

360     for ( ; *s; s++) {
361             switch (*s) {
362             case (ASCII_NBRSP):
363                     printf("\\ ");
236                     printf("\\~");
364                     break;
365             case (ASCII_HYPH):
366                     putchar('-');
367                     break;
```



```
368             case (' '):
369                     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 }

384 static void
385 print_line(const char *s, int newflags)
386 {

388     outflags &= ~MMAN_br;
389     outflags |= MMAN_nl;
390     print_word(s);
391     outflags |= newflags;
392 }

394 static void
395 print_block(const char *s, int newflags)
396 {

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 }

412 static void
413 print_offs(const char *v)
414 {
415     char              buf[24];
416     struct roffsu     su;
417     size_t            sz;

419     print_line(".RS", MMAN_Bk_susp);

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);

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];

453         snprintf(buf, sizeof(buf), "%zun", sz);
454         print_word(buf);
455         outflags |= MMAN_nl;
456 }

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;

469         numeric = 1;
470         remain = 0;

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);

485         /* XXX Rough estimation, might have multiple parts. */
486         chsz = (NULL != child && MDOC_TEXT == child->type) ?
487                         strlen(child->string) : 0;

489         /* Maybe we are inside an enclosing list? */
490         mid_it();

492         /*
493          * Save our own indentation,
494          * such that child lists can use it.
495          */
496         Bl_stack[Bl_stack_len++] = sz + 2;

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 }

513 void
514 print_count(int *count)
515 {
516         char              buf[12];

518         snprintf(buf, sizeof(buf), "%d.", ++*count);
519         print_word(buf);
520 }

522 void
523 man_man(void *arg, const struct man *man)
524 {

526         /*
527          * Dump the keep buffer.
528          * We're guaranteed by now that this exists (is non-NULL).
529          * Flush stdout afterward, just in case.
530          */
531         fputs(mparse_getkeep(man_mparse(man)), stdout);
532         fflush(stdout);
533 }

535 void
536 man_mdoc(void *arg, const struct mdoc *mdoc)
537 {
538         const struct mdoc_meta *meta;
264         const struct mdoc_meta *m;
539         const struct mdoc_node *n;
266         struct mman             mm;

541         meta = mdoc_meta(mdoc);
268         m = mdoc_meta(mdoc);
542         n = mdoc_node(mdoc);

544         printf(".TH \"%s\" \"%s\" \"%s\" \"%s\" \"%s\"\n",
545                         meta->title, meta->msec, meta->date,
546                         meta->os, meta->vol);
271         printf(".TH \"%s\" \"%s\" \"%s\" \"%s\" \"%s\"",
272                         m->title, m->msec, m->date, m->os, m->vol);

548         /* Disable hyphenation and if nroff, disable justification. */
549         printf(".nh\n.if n .ad l");
274         memset(&mm, 0, sizeof(struct mman));

551         outflags = MMAN_nl | MMAN_Sm;
552         if (0 == fontqueue.size) {
553                 fontqueue.size = 8;
554                 fontqueue.head = fontqueue.tail = mandoc_malloc(8);
555                 *fontqueue.tail = 'R';
556         }
557         print_node(meta, n);
276         mm.need_nl = 1;
277         print_node(m, n, &mm);
```

```
558         putchar('\n');
559 }

561 static void
562 print_node(DECL_ARGS)
563 {
564         const struct mdoc_node  *sub;
284         const struct mdoc_node  *prev, *sub;
565         const struct manact     *act;
566         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;
292         prev = n->prev ? n->prev : n->parent;
293         if (prev && prev->line < n->line)
294                 mm->need_nl = 1;

575         act = NULL;
576         cond = 0;
577         do_sub = 1;

579         if (MDOC_TEXT == n->type) {
580                 /*
581                  * Make sure that we don't happen to start with a
582                  * control character at the start of a line.
583                  */
584                 if (MMAN_nl & outflags && ('.' == *n->string ||
305                 if (mm->need_nl && ('.' == *n->string ||
585                                     '\'' == *n->string)) {
586                         print_word("");
587                         printf("\\&");
588                         outflags &= ~MMAN_spc;
307                         print_word(mm, "\\&");
308                         mm->need_space = 0;
589                 }
590                 print_word(n->string);
310                 print_word(mm, n->string);
591         } else {
592                 /*
593                  * Conditionally run the pre-node action handler for a
594                  * node.
595                  */
596                 act = manacts + n->tok;
597                 cond = NULL == act->cond || (*act->cond)(meta, n);
317                 cond = NULL == act->cond || (*act->cond)(m, n, mm);
598                 if (cond && act->pre)
599                         do_sub = (*act->pre)(meta, n);
319                         do_sub = (*act->pre)(m, n, mm);
600         }

602         /*
603          * Conditionally run all child nodes.
604          * Note that this iterates over children instead of using
605          * recursion.  This prevents unnecessary depth in the stack.
606          */
607         if (do_sub)
608                 for (sub = n->child; sub; sub = sub->next)
609                         print_node(meta, sub);
329                         print_node(m, sub, mm);

611         /*
612          * Lastly, conditionally run the post-node handler.
```



```
613          */
614         if (cond && act->post)
615                 (*act->post)(meta, n);
335                 (*act->post)(m, n, mm);
616 }
______unchanged_portion_omitted_

352 /*
353  * Output a font encoding before a node, e.g., \fR.
354  * This obviously has no trailing space.
355  */
632 static int
633 pre_enc(DECL_ARGS)
634 {
635         const char      *prefix;

637         prefix = manacts[n->tok].prefix;
638         if (NULL == prefix)
639                 return(1);
640         print_word(prefix);
641         outflags &= ~MMAN_spc;
364         print_word(mm, prefix);
365         mm->need_space = 0;
642         return(1);
643 }

369 /*
370  * Output a font encoding subsequent a node, e.g., \fP.
371  */
645 static void
646 post_enc(DECL_ARGS)
647 {
648         const char *suffix;

650         suffix = manacts[n->tok].suffix;
651         if (NULL == suffix)
652                 return;
653         outflags &= ~MMAN_spc;
654         print_word(suffix);
380         mm->need_space = 0;
381         print_word(mm, suffix);
655 }

384 /*
385  * Used in listings (percent = %A, e.g.).
386  * FIXME: this is incomplete.
387  * It doesn't print a nice ", and" for lists.
388  */
657 static void
658 post_font(DECL_ARGS)
659 {

661         font_pop();
662 }

664 static void
665 post_percent(DECL_ARGS)
666 {

668         if (pre_em == manacts[n->tok].pre)
669                 font_pop();
670         if (n->next) {
671                 print_word(",");
672                 if (n->prev &&  n->prev->tok == n->tok &&
673                                 n->next->tok == n->tok)
674                         print_word("and");
```

```
675         } else {
676                 print_word(".");
677                 outflags |= MMAN_nl;
393         post_enc(m, n, mm);
394         if (n->next)
395                 print_word(mm, ",");
396         else {
397                 print_word(mm, ".");
398                 mm->need_nl = 1;
678         }
679 }

681 static int
682 pre__t(DECL_ARGS)
683 {

685         if (n->parent && MDOC_Rs == n->parent->tok &&
686                         n->parent->norm->Rs.quote_T) {
687                 print_word("");
688                 putchar('\"');
689                 outflags &= ~MMAN_spc;
690         } else
691                 font_push('I');
692         return(1);
693 }

695 static void
696 post__t(DECL_ARGS)
697 {

699         if (n->parent && MDOC_Rs == n->parent->tok &&
700                         n->parent->norm->Rs.quote_T) {
701                 outflags &= ~MMAN_spc;
702                 print_word("");
703                 putchar('\"');
704         } else
705                 font_pop();
706         post_percent(meta, n);
707 }

709 /*
710  * Print before a section header.
711  */
712 static int
713 pre_sect(DECL_ARGS)
714 {

716         if (MDOC_HEAD == n->type) {
717                 outflags |= MMAN_sp;
718                 print_block(manacts[n->tok].prefix, 0);
719                 print_word("");
720                 putchar('\"');
721                 outflags &= ~MMAN_spc;
722         }
409         if (MDOC_HEAD != n->type)
723         return(1);
411         mm->need_nl = 1;
412         print_word(mm, manacts[n->tok].prefix);
413         print_word(mm, "\"");
414         mm->need_space = 0;
415         return(1);
724 }

726 /*
727  * Print subsequent a section header.
728  */
```



```
729 static void
730 post_sect(DECL_ARGS)
731 {

733         if (MDOC_HEAD != n->type)
734                 return;
735         outflags &= ~MMAN_spc;
736         print_word("");
737         putchar('\"');
738         outflags |= MMAN_nl;
739         if (MDOC_Sh == n->tok && SEC_AUTHORS == n->sec)
740                 outflags &= ~(MMAN_An_split | MMAN_An_nosplit);
427         mm->need_space = 0;
428         print_word(mm, "\"");
429         mm->need_nl = 1;
741 }

743 /* See mdoc_term.c, synopsis_pre() for comments. */
744 static void
745 pre_syn(const struct mdoc_node *n)
746 {

748         if (NULL == n->prev || ! (MDOC_SYNPRETTY & n->flags))
749                 return;

751         if (n->prev->tok == n->tok &&
752                         MDOC_Ft != n->tok &&
753                         MDOC_Fo != n->tok &&
754                         MDOC_Fn != n->tok) {
755                 outflags |= MMAN_br;
756                 return;
757         }

759         switch (n->prev->tok) {
760         case (MDOC_Fd):
761                 /* FALLTHROUGH */
762         case (MDOC_Fn):
763                 /* FALLTHROUGH */
764         case (MDOC_Fo):
765                 /* FALLTHROUGH */
766         case (MDOC_In):
767                 /* FALLTHROUGH */
768         case (MDOC_Vt):
769                 outflags |= MMAN_sp;
770                 break;
771         case (MDOC_Ft):
772                 if (MDOC_Fn != n->tok && MDOC_Fo != n->tok) {
773                         outflags |= MMAN_sp;
774                         break;
775                 }
776                 /* FALLTHROUGH */
777         default:
778                 outflags |= MMAN_br;
779                 break;
780         }
781 }

783 static int
784 pre_an(DECL_ARGS)
785 {

787         switch (n->norm->An.auth) {
788         case (AUTH_split):
789                 outflags &= ~MMAN_An_nosplit;
790                 outflags |= MMAN_An_split;
791                 return(0);
```

```
792         case (AUTH_nosplit):
793                 outflags &= ~MMAN_An_split;
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;
436         mm->need_space = 0;
437         print_word(mm, "‘");
438         mm->need_space = 0;
813         return(0);
814 }

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.offs);
447             DISP_literal  == n->norm->Bd.type) {
448                 mm->need_nl = 1;
449                 print_word(mm, ".nf");
450         }
451         mm->need_nl = 1;
828         return(1);
829 }

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);
460             DISP_literal  == n->norm->Bd.type) {
461                 mm->need_nl = 1;
462                 print_word(mm, ".fi");
857         }
858         switch (n->norm->Bf.font) {
859         case (FONT_Em):
860                 font_push('I');
861                 break;
862         case (FONT_Sy):
863                 font_push('B');
864                 break;
865         default:
866                 font_push('R');
867                 break;
868         }
869         return(1);
464         mm->need_nl = 1;
870 }

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.offs) {
914                 print_offs(n->norm->Bl.offs);
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.offs) {
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;
471         mm->need_nl = 1;
472         print_word(mm, ".br");
473         mm->need_nl = 1;
972         return(0);
973 }
```



```
975 static int
976 pre_bx(DECL_ARGS)
977 {

979         n = n->child;
980         if (n) {
981                 print_word(n->string);
982                 outflags &= ~MMAN_spc;
483                 print_word(mm, n->string);
484                 mm->need_space = 0;
983                 n = n->next;
984         }
985         print_word("BSD");
487         print_word(mm, "BSD");
986         if (NULL == n)
987                 return(0);
988         outflags &= ~MMAN_spc;
989         print_word("-");
990         outflags &= ~MMAN_spc;
991         print_word(n->string);
490         mm->need_space = 0;
491         print_word(mm, "-");
492         mm->need_space = 0;
493         print_word(mm, n->string);
992         return(0);
993 }

995 static int
996 pre_dl(DECL_ARGS)
997 {

999         print_offs("6n");
501         mm->need_nl = 1;
502         print_word(mm, ".RS 6n");
503         mm->need_nl = 1;
1000        return(1);
1001 }

1003 static void
1004 post_dl(DECL_ARGS)
1005 {

1007        print_line(".RE", MMAN_nl);

1009        /* Maybe we are inside an enclosing list? */
1010        if (NULL != n->parent->next)
1011                mid_it();
511         mm->need_nl = 1;
512         print_word(mm, ".RE");
513         mm->need_nl = 1;
1012 }

1014 static int
1015 pre_em(DECL_ARGS)
1016 {

1018        font_push('I');
1019        return(1);
1020 }

1022 static void
1023 post_eo(DECL_ARGS)
1024 {

1026        if (MDOC_HEAD == n->type || MDOC_BODY == n->type)
1027                outflags &= ~MMAN_spc;
```

```
}

static int
pre_fa(DECL_ARGS)
{
        int      am_Fa;

        am_Fa = MDOC_Fa == n->tok;

        if (am_Fa)
                n = n->child;

        while (NULL != n) {
                font_push('I');
                if (am_Fa || MDOC_SYNPRETTY & n->flags)
                        outflags |= MMAN_nbrword;
                print_node(meta, n);
                font_pop();
                if (NULL != (n = n->next))
                        print_word(",");
        }
        return(0);
}

static void
post_fa(DECL_ARGS)
{

        if (NULL != n->next && MDOC_Fa == n->next->tok)
                print_word(",");
}

static int
pre_fd(DECL_ARGS)
{

        pre_syn(n);
        font_push('B');
        return(1);
}

static void
post_fd(DECL_ARGS)
{

        font_pop();
        outflags |= MMAN_br;
}

static int
pre_fl(DECL_ARGS)
{

        font_push('B');
        print_word("\\-");
        outflags &= ~MMAN_spc;
        return(1);
}

static void
post_fl(DECL_ARGS)
{

        font_pop();
        if (0 == n->nchild && NULL != n->next &&
                        n->next->line == n->line)
```



```
                outflags &= ~MMAN_spc;
}

static int
pre_fn(DECL_ARGS)
{

        pre_syn(n);

        n = n->child;
        if (NULL == n)
                return(0);

        if (MDOC_SYNPRETTY & n->flags)
                print_block(".HP 4n", MMAN_nl);

        font_push('B');
        print_node(meta, n);
        font_pop();
        outflags &= ~MMAN_spc;
        print_word("(");
        outflags &= ~MMAN_spc;

        n = n->next;
        if (NULL != n)
                pre_fa(meta, n);
        return(0);
}

static void
post_fn(DECL_ARGS)
{

        print_word(")");
        if (MDOC_SYNPRETTY & n->flags) {
                print_word(";");
                outflags |= MMAN_PP;
        }
}

static int
pre_fo(DECL_ARGS)
{

        switch (n->type) {
        case (MDOC_BLOCK):
                pre_syn(n);
                break;
        case (MDOC_HEAD):
                if (MDOC_SYNPRETTY & n->flags)
                        print_block(".HP 4n", MMAN_nl);
                font_push('B');
                break;
        case (MDOC_BODY):
                outflags &= ~MMAN_spc;
                print_word("(");
                outflags &= ~MMAN_spc;
                break;
        default:
                break;
        }
        return(1);
}

static void
post_fo(DECL_ARGS)
```

```
1160 {
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_sp;
1229                 outflags &= ~MMAN_br;
 521         if (MDOC_HEAD == n->type) {
 522                 mm->need_nl = 1;
 523                 print_word(mm, ".TP");
 524                 bln = n->parent->parent->prev;
1230                 switch (bln->norm->Bl.type) {
1231                 case (LIST_item):
1232                         return(0);
1233                 case (LIST_inset):
1234                         /* FALLTHROUGH */
1235                 case (LIST_diag):
1236                         /* FALLTHROUGH */
1237                 case (LIST_ohang):
1238                         if (bln->norm->Bl.type == LIST_diag)
1239                                 print_line(".B \"", 0);
1240                         else
1241                                 print_line(".R \"", 0);
1242                         outflags &= ~MMAN_spc;
1243                         return(1);
1244                 case (LIST_bullet):
1245                         /* FALLTHROUGH */
1246                 case (LIST_dash):
1247                         /* FALLTHROUGH */
1248                 case (LIST_hyphen):
1249                         print_width(bln->norm->Bl.width, NULL, 0);
1250                         TPremain = 0;
1251                         outflags |= MMAN_nl;
1252                         font_push('B');
1253                         if (LIST_bullet == bln->norm->Bl.type)
1254                                 print_word("o");
1255                         else
1256                                 print_word("-");
1257                         font_pop();
 527                         print_word(mm, "4n");
 528                         mm->need_nl = 1;
 529                         print_word(mm, "\\fBo\\fP");
1258                         break;
1259                 case (LIST_enum):
1260                         print_width(bln->norm->Bl.width, NULL, 0);
1261                         TPremain = 0;
1262                         outflags |= MMAN_nl;
1263                         print_count(&bln->norm->Bl.count);
1264                         break;
1265                 case (LIST_hang):
1266                         print_width(bln->norm->Bl.width, n->child, 6);
1267                         TPremain = 0;
1268                         break;
1269                 case (LIST_tag):
1270                         print_width(bln->norm->Bl.width, n->child, 0);
1271                         putchar('\n');
1272                         outflags &= ~MMAN_spc;
1273                         return(1);
1274                 default:
1275                         return(1);
1276                 }
1277                 outflags |= MMAN_nl;
1278         default:
 532                         if (bln->norm->Bl.width)
 533                                 print_word(mm, bln->norm->Bl.width);
1279                 break;
1280         }
1281         return(1);
1282 }
```

```
/*
 * This function is called after closing out an indented block.
 * If we are inside an enclosing list, restore its indentation.
 */
static void
mid_it(void)
{
        char             buf[24];

        /* Nothing to do outside a list. */
        if (0 == Bl_stack_len || 0 == Bl_stack[Bl_stack_len - 1])
                return;

        /* The indentation has already been set up. */
        if (Bl_stack_post[Bl_stack_len - 1])
                return;

        /* Restore the indentation of the enclosing list. */
        print_line(".RS", MMAN_Bk_susp);
        snprintf(buf, sizeof(buf), "%zun", Bl_stack[Bl_stack_len - 1]);
        print_word(buf);

        /* Remeber to close out this .RS block later. */
        Bl_stack_post[Bl_stack_len - 1] = 1;
}

static void
post_it(DECL_ARGS)
{
        const struct mdoc_node *bln;

        bln = n->parent->parent;

        switch (n->type) {
        case (MDOC_HEAD):
                switch (bln->norm->Bl.type) {
                case (LIST_diag):
                        outflags &= ~MMAN_spc;
                        print_word("\\ ");
                        break;
                case (LIST_ohang):
                        outflags |= MMAN_br;
                        break;
                default:
                        break;
                mm->need_nl = 1;
                }
                break;
        case (MDOC_BODY):
                switch (bln->norm->Bl.type) {
                case (LIST_bullet):
                        /* FALLTHROUGH */
                case (LIST_dash):
                        /* FALLTHROUGH */
                case (LIST_hyphen):
                        /* FALLTHROUGH */
                case (LIST_enum):
                        /* FALLTHROUGH */
                case (LIST_hang):
                        /* FALLTHROUGH */
                case (LIST_tag):
                        assert(Bl_stack_len);
                        Bl_stack[--Bl_stack_len] = 0;

                        /*
```



```
                         * Our indentation had to be restored
                         * after a child display or child list.
                         * Close out that indentation block now.
                         */
                        if (Bl_stack_post[Bl_stack_len]) {
                                print_line(".RE", MMAN_nl);
                                Bl_stack_post[Bl_stack_len] = 0;
                        }
                        break;
                case (LIST_column):
                        if (NULL != n->next) {
                                putchar('\t');
                                outflags &= ~MMAN_spc;
                        }
                        break;
                default:
                        break;
                }
                break;
        default:
                break;
        }
}

static void
post_lb(DECL_ARGS)
{

        if (SEC_LIBRARY == n->sec)
                outflags |= MMAN_br;
}

static int
pre_lk(DECL_ARGS)
{
        const struct mdoc_node *link, *descr;

        if (NULL == (link = n->child))
                return(0);

        if (NULL != (descr = link->next)) {
                font_push('I');
                while (NULL != descr) {
                        print_word(descr->string);
                        descr = descr->next;
                }
                print_word(":");
                font_pop();
        }

        font_push('B');
        print_word(link->string);
        font_pop();
        return(0);
}

static int
pre_li(DECL_ARGS)
{

        font_push('R');
        return(1);
}

static int
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');
 547         print_word(mm, "\\fB");
 548         mm->need_space = 0;
1434         if (NULL == n->child)
1435                 print_word(meta->name);
 550                 print_word(mm, m->name);
1436         return(1);
1437 }

1439 static void
1440 post_nm(DECL_ARGS)
1441 {

1443         switch (n->type) {
1444         case (MDOC_BLOCK):
1445                 outflags &= ~MMAN_Bk;
1446                 break;
1447         case (MDOC_HEAD):
1448                 /* FALLTHROUGH */
1449         case (MDOC_ELEM):
1450                 font_pop();
1451                 break;
1452         default:
1453                 break;
1454         }
 558         if (MDOC_ELEM != n->type && MDOC_HEAD != n->type)
 559                 return;
 560         mm->need_space = 0;
 561         print_word(mm, "\\fP");
1455 }

1457 static int
1458 pre_no(DECL_ARGS)
1459 {

1461         outflags |= MMAN_spc_force;
1462         return(1);
1463 }

1465 static int
1466 pre_ns(DECL_ARGS)
1467 {

1469         outflags &= ~MMAN_spc;
 568         mm->need_space = 0;
1470         return(0);
1471 }
```



```
1473 static void
1474 post_pf(DECL_ARGS)
1475 {

1477         outflags &= ~MMAN_spc;
 576         mm->need_space = 0;
1478 }

1480 static int
1481 pre_pp(DECL_ARGS)
1482 {

1484         if (MDOC_It != n->parent->tok)
1485                 outflags |= MMAN_PP;
1486         outflags |= MMAN_sp | MMAN_nl;
1487         outflags &= ~MMAN_br;
1488         return(0);
1489 }

1491 static int
1492 pre_rs(DECL_ARGS)
1493 {

1495         if (SEC_SEE_ALSO == n->sec) {
1496                 outflags |= MMAN_PP | MMAN_sp | MMAN_nl;
1497                 outflags &= ~MMAN_br;
1498         }
 583         mm->need_nl = 1;
 584         if (MDOC_It == n->parent->tok)
 585                 print_word(mm, ".sp");
 586         else
 587                 print_word(mm, ".PP");
 588         mm->need_nl = 1;
1499         return(1);
1500 }

1502 static int
1503 pre_sm(DECL_ARGS)
1504 {

1506         assert(n->child && MDOC_TEXT == n->child->type);
1507         if (0 == strcmp("on", n->child->string))
1508                 outflags |= MMAN_Sm | MMAN_spc;
1509         else
1510                 outflags &= ~MMAN_Sm;
1511         return(0);
1512 }

1514 static int
1515 pre_sp(DECL_ARGS)
1516 {

1518         if (MMAN_PP & outflags) {
1519                 outflags &= ~MMAN_PP;
1520                 print_line(".PP", 0);
1521         } else
1522                 print_line(".sp", 0);
 596         mm->need_nl = 1;
 597         print_word(mm, ".sp");
1523         return(1);
1524 }

1526 static void
1527 post_sp(DECL_ARGS)
1528 {
```

```
1530         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);
 615         print_node(m, n, mm);
1577         n = n->next;
1578         if (NULL == n)
1579                 return(0);
1580         outflags &= ~MMAN_spc;
1581         print_word("(");
1582         print_node(meta, n);
1583         print_word(")");
 619         mm->need_space = 0;
 620         print_word(mm, "(");
 621         print_node(m, n, mm);
 622         print_word(mm, ")");
1584         return(0);
1585 }

1587 static int
1588 pre_ux(DECL_ARGS)
```



```
1589 {

1591         print_word(manacts[n->tok].prefix);
 630         print_word(mm, manacts[n->tok].prefix);
1592         if (NULL == n->child)
1593                 return(0);
1594         outflags &= ~MMAN_spc;
1595         print_word("\\ ");
1596         outflags &= ~MMAN_spc;
 633         mm->need_space = 0;
 634         print_word(mm, "\\~");
 635         mm->need_space = 0;
1597         return(1);
1598 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   45416 Wed Jul 30 20:55:10 2014
new/usr/src/cmd/mandoc/mdoc_term.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: mdoc_term.c,v 1.258 2013/12/25 21:24:12 schwarze Exp $ */
/*      $Id: mdoc_term.c,v 1.238 2011/11/13 13:15:14 schwarze Exp $ */
/*
 * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2010, 2012, 2013 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2013 Franco Fichtner <franco@lastsummer.de>
 * Copyright (c) 2010 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

#include <assert.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mandoc.h"
#include "out.h"
#include "term.h"
#include "mdoc.h"
#include "main.h"

struct  termpair {
        struct termpair  *ppair;
        int               count;
};

#define DECL_ARGS struct termp *p, \
                  struct termpair *pair, \
                  const struct mdoc_meta *meta, \
                  struct mdoc_node *n
                  const struct mdoc_meta *m, \
                  const struct mdoc_node *n

struct  termact {
        int     (*pre)(DECL_ARGS);
        void    (*post)(DECL_ARGS);
};

static  size_t    a2width(const struct termp *, const char *);
static  size_t    a2height(const struct termp *, const char *);
static  size_t    a2offs(const struct termp *, const char *);
```



```
static  void      print_bvspace(struct termp *,
                        const struct mdoc_node *,
                        const struct mdoc_node *);
static  void      print_mdoc_node(DECL_ARGS);
static  void      print_mdoc_nodelist(DECL_ARGS);
static  void      print_mdoc_head(struct termp *, const void *);
static  void      print_mdoc_foot(struct termp *, const void *);
static  void      synopsis_pre(struct termp *,
                        const struct mdoc_node *);

static  void      termp____post(DECL_ARGS);
static  void      termp__t_post(DECL_ARGS);
static  void      termp_an_post(DECL_ARGS);
static  void      termp_bd_post(DECL_ARGS);
static  void      termp_bk_post(DECL_ARGS);
static  void      termp_bl_post(DECL_ARGS);
static  void      termp_fd_post(DECL_ARGS);
static  void      termp_d1_post(DECL_ARGS);
static  void      termp_fo_post(DECL_ARGS);
static  void      termp_in_post(DECL_ARGS);
static  void      termp_it_post(DECL_ARGS);
static  void      termp_lb_post(DECL_ARGS);
static  void      termp_nm_post(DECL_ARGS);
static  void      termp_pf_post(DECL_ARGS);
static  void      termp_quote_post(DECL_ARGS);
static  void      termp_sh_post(DECL_ARGS);
static  void      termp_ss_post(DECL_ARGS);

static  int       termp__a_pre(DECL_ARGS);
static  int       termp__t_pre(DECL_ARGS);
static  int       termp_an_pre(DECL_ARGS);
static  int       termp_ap_pre(DECL_ARGS);
static  int       termp_bd_pre(DECL_ARGS);
static  int       termp_bf_pre(DECL_ARGS);
static  int       termp_bk_pre(DECL_ARGS);
static  int       termp_bl_pre(DECL_ARGS);
static  int       termp_bold_pre(DECL_ARGS);
static  int       termp_bt_pre(DECL_ARGS);
static  int       termp_bx_pre(DECL_ARGS);
static  int       termp_cd_pre(DECL_ARGS);
static  int       termp_d1_pre(DECL_ARGS);
static  int       termp_ex_pre(DECL_ARGS);
static  int       termp_fa_pre(DECL_ARGS);
static  int       termp_fd_pre(DECL_ARGS);
static  int       termp_fl_pre(DECL_ARGS);
static  int       termp_fn_pre(DECL_ARGS);
static  int       termp_fo_pre(DECL_ARGS);
static  int       termp_ft_pre(DECL_ARGS);
static  int       termp_igndelim_pre(DECL_ARGS);
static  int       termp_in_pre(DECL_ARGS);
static  int       termp_it_pre(DECL_ARGS);
static  int       termp_li_pre(DECL_ARGS);
static  int       termp_lk_pre(DECL_ARGS);
static  int       termp_nd_pre(DECL_ARGS);
static  int       termp_nm_pre(DECL_ARGS);
static  int       termp_ns_pre(DECL_ARGS);
static  int       termp_quote_pre(DECL_ARGS);
static  int       termp_rs_pre(DECL_ARGS);
static  int       termp_rv_pre(DECL_ARGS);
static  int       termp_sh_pre(DECL_ARGS);
static  int       termp_sm_pre(DECL_ARGS);
static  int       termp_sp_pre(DECL_ARGS);
static  int       termp_ss_pre(DECL_ARGS);
static  int       termp_under_pre(DECL_ARGS);
static  int       termp_ud_pre(DECL_ARGS);
```

```
120 static  int       termp_vt_pre(DECL_ARGS);
121 static  int       termp_xr_pre(DECL_ARGS);
122 static  int       termp_xx_pre(DECL_ARGS);

124 static  const struct termact termacts[MDOC_MAX] = {
125         { termp_ap_pre, NULL }, /* Ap */
126         { NULL, NULL }, /* Dd */
127         { NULL, NULL }, /* Dt */
128         { NULL, NULL }, /* Os */
129         { termp_sh_pre, termp_sh_post }, /* Sh */
130         { termp_ss_pre, termp_ss_post }, /* Ss */
131         { termp_sp_pre, NULL }, /* Pp */
132         { termp_d1_pre, termp_bl_post }, /* D1 */
133         { termp_d1_pre, termp_bl_post }, /* Dl */
132         { termp_d1_pre, termp_d1_post }, /* D1 */
133         { termp_d1_pre, termp_d1_post }, /* Dl */
134         { termp_bd_pre, termp_bd_post }, /* Bd */
135         { NULL, NULL }, /* Ed */
136         { termp_bl_pre, termp_bl_post }, /* Bl */
137         { NULL, NULL }, /* El */
138         { termp_it_pre, termp_it_post }, /* It */
139         { termp_under_pre, NULL }, /* Ad */
140         { termp_an_pre, termp_an_post }, /* An */
141         { termp_under_pre, NULL }, /* Ar */
142         { termp_cd_pre, NULL }, /* Cd */
143         { termp_bold_pre, NULL }, /* Cm */
144         { NULL, NULL }, /* Dv */
145         { NULL, NULL }, /* Er */
146         { NULL, NULL }, /* Ev */
147         { termp_ex_pre, NULL }, /* Ex */
148         { termp_fa_pre, NULL }, /* Fa */
149         { termp_fd_pre, termp_fd_post }, /* Fd */
149         { termp_fd_pre, NULL }, /* Fd */
150         { termp_fl_pre, NULL }, /* Fl */
151         { termp_fn_pre, NULL }, /* Fn */
152         { termp_ft_pre, NULL }, /* Ft */
153         { termp_bold_pre, NULL }, /* Ic */
154         { termp_in_pre, termp_in_post }, /* In */
155         { termp_li_pre, NULL }, /* Li */
156         { termp_nd_pre, NULL }, /* Nd */
157         { termp_nm_pre, termp_nm_post }, /* Nm */
158         { termp_quote_pre, termp_quote_post }, /* Op */
159         { NULL, NULL }, /* Ot */
160         { termp_under_pre, NULL }, /* Pa */
161         { termp_rv_pre, NULL }, /* Rv */
162         { NULL, NULL }, /* St */
163         { termp_under_pre, NULL }, /* Va */
164         { termp_vt_pre, NULL }, /* Vt */
165         { termp_xr_pre, NULL }, /* Xr */
166         { termp__a_pre, termp____post }, /* %A */
167         { termp_under_pre, termp____post }, /* %B */
168         { NULL, termp____post }, /* %D */
169         { termp_under_pre, termp____post }, /* %I */
170         { termp_under_pre, termp____post }, /* %J */
171         { NULL, termp____post }, /* %N */
172         { NULL, termp____post }, /* %O */
173         { NULL, termp____post }, /* %P */
174         { NULL, termp____post }, /* %R */
175         { termp__t_pre, termp__t_post }, /* %T */
176         { NULL, termp____post }, /* %V */
177         { NULL, NULL }, /* Ac */
178         { termp_quote_pre, termp_quote_post }, /* Ao */
179         { termp_quote_pre, termp_quote_post }, /* Aq */
180         { NULL, NULL }, /* At */
181         { NULL, NULL }, /* Bc */
182         { termp_bf_pre, NULL }, /* Bf */
```



```
183         { termp_quote_pre, termp_quote_post }, /* Bo */
184         { termp_quote_pre, termp_quote_post }, /* Bq */
185         { termp_xx_pre, NULL }, /* Bsx */
186         { termp_bx_pre, NULL }, /* Bx */
187         { NULL, NULL }, /* Db */
188         { NULL, NULL }, /* Dc */
189         { termp_quote_pre, termp_quote_post }, /* Do */
190         { termp_quote_pre, termp_quote_post }, /* Dq */
191         { NULL, NULL }, /* Ec */ /* FIXME: no space */
192         { NULL, NULL }, /* Ef */
193         { termp_under_pre, NULL }, /* Em */
194         { termp_quote_pre, termp_quote_post }, /* Eo */
195         { termp_xx_pre, NULL }, /* Fx */
196         { termp_bold_pre, NULL }, /* Ms */
197         { NULL, NULL }, /* No */
197         { termp_igndelim_pre, NULL }, /* No */
198         { termp_ns_pre, NULL }, /* Ns */
199         { termp_xx_pre, NULL }, /* Nx */
200         { termp_xx_pre, NULL }, /* Ox */
201         { NULL, NULL }, /* Pc */
202         { NULL, termp_pf_post }, /* Pf */
202         { termp_igndelim_pre, termp_pf_post }, /* Pf */
203         { termp_quote_pre, termp_quote_post }, /* Po */
204         { termp_quote_pre, termp_quote_post }, /* Pq */
205         { NULL, NULL }, /* Qc */
206         { termp_quote_pre, termp_quote_post }, /* Ql */
207         { termp_quote_pre, termp_quote_post }, /* Qo */
208         { termp_quote_pre, termp_quote_post }, /* Qq */
209         { NULL, NULL }, /* Re */
210         { termp_rs_pre, NULL }, /* Rs */
211         { NULL, NULL }, /* Sc */
212         { termp_quote_pre, termp_quote_post }, /* So */
213         { termp_quote_pre, termp_quote_post }, /* Sq */
214         { termp_sm_pre, NULL }, /* Sm */
215         { termp_under_pre, NULL }, /* Sx */
216         { termp_bold_pre, NULL }, /* Sy */
217         { NULL, NULL }, /* Tn */
218         { termp_xx_pre, NULL }, /* Ux */
219         { NULL, NULL }, /* Xc */
220         { NULL, NULL }, /* Xo */
221         { termp_fo_pre, termp_fo_post }, /* Fo */
222         { NULL, NULL }, /* Fc */
223         { termp_quote_pre, termp_quote_post }, /* Oo */
224         { NULL, NULL }, /* Oc */
225         { termp_bk_pre, termp_bk_post }, /* Bk */
226         { NULL, NULL }, /* Ek */
227         { termp_bt_pre, NULL }, /* Bt */
228         { NULL, NULL }, /* Hf */
229         { NULL, NULL }, /* Fr */
230         { termp_ud_pre, NULL }, /* Ud */
231         { NULL, termp_lb_post }, /* Lb */
232         { termp_sp_pre, NULL }, /* Lp */
233         { termp_lk_pre, NULL }, /* Lk */
234         { termp_under_pre, NULL }, /* Mt */
235         { termp_quote_pre, termp_quote_post }, /* Brq */
236         { termp_quote_pre, termp_quote_post }, /* Bro */
237         { NULL, NULL }, /* Brc */
238         { NULL, termp____post }, /* %C */
239         { NULL, NULL }, /* Es */ /* TODO */
240         { NULL, NULL }, /* En */ /* TODO */
241         { termp_xx_pre, NULL }, /* Dx */
242         { NULL, termp____post }, /* %Q */
243         { termp_sp_pre, NULL }, /* br */
244         { termp_sp_pre, NULL }, /* sp */
245         { NULL, termp____post }, /* %U */
245         { termp_under_pre, termp____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;
254         const struct mdoc_meta  *m;
255         struct termp            *p;

257         p = (struct termp *)arg;

259         if (0 == p->defindent)
260                 p->defindent = 5;

262         p->overstep = 0;
263         p->maxrmargin = p->defrmargin;
264         p->tabwidth = term_len(p, 5);

266         if (NULL == p->symtab)
267                 p->symtab = mchars_alloc();

269         n = mdoc_node(mdoc);
270         meta = mdoc_meta(mdoc);
270         m = mdoc_meta(mdoc);

272         term_begin(p, print_mdoc_head, print_mdoc_foot, meta);
272         term_begin(p, print_mdoc_head, print_mdoc_foot, m);

274         if (n->child)
275                 print_mdoc_nodelist(p, NULL, meta, n->child);
275                 print_mdoc_nodelist(p, NULL, m, n->child);

277         term_end(p);
278 }


281 static void
282 print_mdoc_nodelist(DECL_ARGS)
283 {

285         print_mdoc_node(p, pair, meta, n);
285         print_mdoc_node(p, pair, m, n);
286         if (n->next)
287                 print_mdoc_nodelist(p, pair, meta, n->next);
287                 print_mdoc_nodelist(p, pair, m, n->next);
288 }


291 /* ARGSUSED */
292 static void
293 print_mdoc_node(DECL_ARGS)
294 {
295         int              chld;
296         const void      *font;
296         struct termpair  npair;
297         size_t           offset, rmargin;

299         chld = 1;
300         offset = p->offset;
301         rmargin = p->rmargin;
302         n->prev_font = term_fontq(p);
303         font = term_fontq(p);
```



```
304         memset(&npair, 0, sizeof(struct termpair));
305         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.
311          *
312          * Also let SYNPRETTY sections behave as if they were wrapped
313          * in a 'Bk' block.
310          */

312         if (TERMP_KEEP & p->flags) {
313                 if (n->prev ? (n->prev->lastline != n->line) :
314                     (n->parent && n->parent->line != n->line)) {
316         if (TERMP_KEEP & p->flags || MDOC_SYNPRETTY & n->flags) {
317                 if (n->prev && n->prev->line != n->line) {
315                         p->flags &= ~TERMP_KEEP;
316                         p->flags |= TERMP_PREKEEP;
320                 } else if (NULL == n->prev) {
321                         if (n->parent && n->parent->line != n->line) {
322                                 p->flags &= ~TERMP_KEEP;
323                                 p->flags |= TERMP_PREKEEP;
317                 }
318         }
326         }

320         /*
329          * Since SYNPRETTY sections aren't "turned off" with 'Ek',
330          * we have to intuit whether we should disable formatting.
331          */

333         if ( ! (MDOC_SYNPRETTY & n->flags) &&
334             ((n->prev   && MDOC_SYNPRETTY & n->prev->flags) ||
335              (n->parent && MDOC_SYNPRETTY & n->parent->flags)))
336                 p->flags &= ~(TERMP_KEEP | TERMP_PREKEEP);

338         /*
321          * After the keep flags have been set up, we may now
322          * produce output.  Note that some pre-handlers do so.
323          */

325         switch (n->type) {
326         case (MDOC_TEXT):
327                 if (' ' == *n->string && MDOC_LINE & n->flags)
328                         term_newln(p);
329                 if (MDOC_DELIMC & n->flags)
330                         p->flags |= TERMP_NOSPACE;
331                 term_word(p, n->string);
332                 if (MDOC_DELIMO & n->flags)
333                         p->flags |= TERMP_NOSPACE;
334                 break;
335         case (MDOC_EQN):
336                 term_eqn(p, n->eqn);
337                 break;
338         case (MDOC_TBL):
339                 term_tbl(p, n->span);
340                 break;
341         default:
342                 if (termacts[n->tok].pre && ENDBODY_NOT == n->end)
343                         chld = (*termacts[n->tok].pre)
344                                 (p, &npair, meta, n);
362                                 (p, &npair, m, n);
345                 break;
346         }

348         if (chld && n->child)
```

```
349                 print_mdoc_nodelist(p, &npair, meta, n->child);
367                 print_mdoc_nodelist(p, &npair, m, n->child);

351         term_fontpopq(p,
352             (ENDBODY_NOT == n->end ? n : n->pending)->prev_font);
369         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_mdoc_foot(struct termp *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, TERMFONT_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_mdoc_head(struct termp *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         strlcpy(buf, meta->vol, BUFSIZ);
```

```
482         assert(m->vol);
483         strlcpy(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 termp_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_bvspace(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.offs)
643                 offset = a2offs(p, bl->norm->Bl.offs);

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         }

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          */

711         switch (type) {
712         case (LIST_bullet):
713                 /* FALLTHROUGH */
714         case (LIST_dash):
715                 /* FALLTHROUGH */
716         case (LIST_hyphen):
717                 /* FALLTHROUGH */
730                 if (width < term_len(p, 4))
731                         width = term_len(p, 4);
732                 break;
718         case (LIST_enum):
719                 if (width < term_len(p, 2))
720                         width = term_len(p, 2);
734                 if (width < term_len(p, 5))
735                         width = term_len(p, 5);
721                 break;
722         case (LIST_hang):
723                 if (0 == width)
724                         width = term_len(p, 8);
725                 break;
726         case (LIST_column):
727                 /* FALLTHROUGH */
728         case (LIST_tag):
729                 if (0 == width)
730                         width = term_len(p, 10);
731                 break;
732         default:
733                 break;
734         }

736         /*
737          * Whitespace control.  Inset bodies need an initial space,
738          * while diagonal bodies need two.
739          */

741         p->flags |= TERMP_NOSPACE;

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         }

756         p->flags |= TERMP_NOSPACE;
```



```
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         }

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 termp_it_post() when we
771          * exit.
772          */

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 */
794         case (LIST_enum):
795                 /* FALLTHROUGH */
787         case (LIST_hyphen):
788                 if (MDOC_HEAD != n->type)
789                         break;
797                 if (MDOC_HEAD == n->type)
790                 p->flags |= TERMP_NOBREAK;
791                 p->trailspace = 1;
792                 break;
793         case (LIST_hang):
794                 if (MDOC_HEAD != n->type)
801                 if (MDOC_HEAD == n->type)
802                         p->flags |= TERMP_NOBREAK;
803                 else
795                         break;

797                 /*
798                  * This is ugly.  If '-hang' is specified and the body
799                  * is a 'Bl' or 'Bd', then we want basically to nullify
800                  * the "overstep" effect in term_flushln() and treat
801                  * this as a '-ohang' list instead.
802                  */
803                 if (n->next->child &&
804                                 (MDOC_Bl == n->next->child->tok ||
805                                  MDOC_Bd == n->next->child->tok))
815                         p->flags &= ~TERMP_NOBREAK;
816                 else
817                         p->flags |= TERMP_HANG;
806                         break;

808                 p->flags |= TERMP_NOBREAK | TERMP_HANG;
809                 p->trailspace = 1;
810                 break;
811         case (LIST_tag):
820                 if (MDOC_HEAD == n->type)
821                         p->flags |= TERMP_NOBREAK | TERMP_TWOSPACE;
```

```
812                 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 termp_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 termp_nm_pre(DECL_ARGS)
998 {

1000        if (MDOC_BLOCK == n->type) {
1001                p->flags |= TERMP_PREKEEP;
1002        if (MDOC_BLOCK == n->type)
1002                return(1);
1003        }
```



```
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) :
1010                    (NULL == n->prev->child ? term_strlen(p, m->name) :
1012                     MDOC_TEXT == n->prev->child->type ?
1013                     term_strlen(p, n->prev->child->string) :
1014                     term_len(p, 5));
1015                return(1);
1016        }

1018        if (NULL == n->child && NULL == meta->name)
1017        if (NULL == n->child && NULL == m->name)
1019                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 termp_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 termp_vt_pre(DECL_ARGS)
1379 {

1381         if (MDOC_ELEM == n->type) {
1382                 synopsis_pre(p, n);
1383                 return(termp_under_pre(p, pair, meta, n));
1378                 return(termp_under_pre(p, pair, m, 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(termp_under_pre(p, pair, meta, n));
1385         return(termp_under_pre(p, pair, m, n));
1391 }
______unchanged_portion_omitted_


1404 /* ARGSUSED */
1405 static int
1406 termp_fd_pre(DECL_ARGS)
1407 {

1409         synopsis_pre(p, n);
1410         return(termp_bold_pre(p, pair, meta, n));
1405         return(termp_bold_pre(p, pair, m, n));
1411 }


1414 /* ARGSUSED */
1415 static void
1416 termp_fd_post(DECL_ARGS)
1417 {

1419         term_newln(p);
1420 }


1423 /* ARGSUSED */
1424 static int
1425 termp_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 termp_d1_post(DECL_ARGS)
1503 {

1505         if (MDOC_BLOCK != n->type)
1506                 return;
1507         term_newln(p);
1508 }


1511 /* ARGSUSED */
1517 static int
1518 termp_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 termp_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);
1571
1572                 if (n->next) {
1573                         p->flags |= TERMP_NOSPACE;
1574                         term_word(p, ",");
1575                 }
1576         }
1577
1578         p->flags |= TERMP_NOSPACE;
1579         term_word(p, ")");
1580
1581         if (pretty) {
1582                 p->flags |= TERMP_NOSPACE;
1583                 term_word(p, ";");
1584                 term_flushln(p);
1585         }
1586
1587         return(0);
1588 }
1589
1590
1591 /* ARGSUSED */
1592 static int
1593 termp_fa_pre(DECL_ARGS)
1594 {
1595         const struct mdoc_node  *nn;
1596
1597         if (n->parent->tok != MDOC_Fo) {
1598                 term_fontpush(p, TERMFONT_UNDER);
1599                 return(1);
1600         }
1601
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);
1607
1608                 if (nn->next || (n->next && n->next->tok == MDOC_Fa)) {
1585                 if (nn->next) {
1609                         p->flags |= TERMP_NOSPACE;
1610                         term_word(p, ",");
1611                 }
1612         }
1613
1591         if (n->child && n->next && n->next->tok == MDOC_Fa) {
1592                 p->flags |= TERMP_NOSPACE;
1593                 term_word(p, ",");
1594         }
1595
1614         return(0);
1615 }
1616
1617
1618 /* ARGSUSED */
1619 static int
1620 termp_bd_pre(DECL_ARGS)
1621 {
1622         size_t                  tabwidth, rm, rmax;
1623         struct mdoc_node        *nn;
1605         const struct mdoc_node  *nn;
1624
1625         if (MDOC_BLOCK == n->type) {
1626                 print_bvspace(p, n, n);
1627                 return(1);
```



```
1628         } else if (MDOC_HEAD == n->type)
1629                 return(0);
1630
1631         if (n->norm->Bd.offs)
1632                 p->offset += a2offs(p, n->norm->Bd.offs);
1633
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
1642         if (DISP_literal != n->norm->Bd.type &&
1643                         DISP_unfilled != n->norm->Bd.type)
1644                 return(1);
1645
1646         tabwidth = p->tabwidth;
1647         if (DISP_literal == n->norm->Bd.type)
1648                 p->tabwidth = term_len(p, 8);
1649
1650         rm = p->rmargin;
1651         rmax = p->maxrmargin;
1652         p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
1653
1654         for (nn = n->child; nn; nn = nn->next) {
1655                 print_mdoc_node(p, pair, meta, nn);
1637                 print_mdoc_node(p, pair, m, nn);
1656                 /*
1657                  * If the printed node flushes its own line, then we
1658                  * needn't do it here as well.  This is hacky, but the
1659                  * notion of selective eoln whitespace is pretty dumb
1660                  * anyway, so don't sweat it.
1661                  */
1662                 switch (nn->tok) {
1663                 case (MDOC_Sm):
1664                         /* FALLTHROUGH */
1665                 case (MDOC_br):
1666                         /* FALLTHROUGH */
1667                 case (MDOC_sp):
1668                         /* FALLTHROUGH */
1669                 case (MDOC_Bl):
1670                         /* FALLTHROUGH */
1671                 case (MDOC_D1):
1672                         /* FALLTHROUGH */
1673                 case (MDOC_Dl):
1674                         /* FALLTHROUGH */
1675                 case (MDOC_Lp):
1676                         /* FALLTHROUGH */
1677                 case (MDOC_Pp):
1678                         continue;
1679                 default:
1680                         break;
1681                 }
1682                 if (nn->next && nn->next->line == nn->line)
1683                         continue;
1684                 term_flushln(p);
1685                 p->flags |= TERMP_NOSPACE;
1686         }
1687
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 termp_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;
1774         }

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 */
1769 static int
1770 termp_igndelim_pre(DECL_ARGS)
1771 {

1773         p->flags |= TERMP_IGNDELIM;
1774         return(1);
1775 }


1778 /* ARGSUSED */
1788 static void
1789 termp_pf_post(DECL_ARGS)
1790 {

1792         p->flags |= TERMP_NOSPACE;
1793 }
```

*unchanged_portion_omitted_*


```
1904 /* ARGSUSED */
1905 static int
1906 termp_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");
1926                 term_word(p, "``");
1936                 break;
1937         case (MDOC_Eo):
1938                 break;
1939         case (MDOC_Po):
1940                 /* FALLTHROUGH */
1941         case (MDOC_Pq):
1942                 term_word(p, "(");
1943                 break;
1944         case (MDOC__T):
1945                 /* FALLTHROUGH */
1946         case (MDOC_Qo):
1947                 /* FALLTHROUGH */
1948         case (MDOC_Qq):
1949                 term_word(p, "\"");
1950                 break;
1951         case (MDOC_Ql):
1952                 /* FALLTHROUGH */
1953         case (MDOC_So):
1954                 /* FALLTHROUGH */
1955         case (MDOC_Sq):
1956                 term_word(p, "\\(oq");
1947                 term_word(p, "`");
1957                 break;
1958         default:
1959                 abort();
1960                 /* NOTREACHED */
1961         }

1963         p->flags |= TERMP_NOSPACE;
1964         return(1);
1965 }
```

```
1968 /* ARGSUSED */
1969 static void
1970 termp_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):
2013                 term_word(p, "\\(cq");
2023                 term_word(p, "'");
2024                 break;
2025         default:
2026                 abort();
2027                 /* NOTREACHED */
2028         }
2028 }


2031 /* ARGSUSED */
```



```
2032 static int
2033 termp_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, TERMFONT_BOLD);
2068         term_word(p, n->child->string);
2069         return(0);
2070 }


2073 /* ARGSUSED */
2074 static void
2075 termp_fo_post(DECL_ARGS)
2076 {

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 termp_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 termp_lk_pre(DECL_ARGS)
2184 {
2185        const struct mdoc_node *link, *descr;
2160        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 termp__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                termp_quote_post(p, pair, meta, n);
2228                termp_quote_post(p, pair, m, n);

2254        termp____post(p, pair, meta, n);
2230        termp____post(p, pair, m, n);
2255 }

2257 /* ARGSUSED */
2258 static int
2259 termp__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(termp_quote_pre(p, pair, meta, n));
2244                return(termp_quote_pre(p, pair, m, n));

2270        term_fontpush(p, TERMFONT_UNDER);
2271        return(1);
2272 }
______unchanged_portion_omitted_
```

```
**********************************************************
   55544 Wed Jul 30 20:55:10 2014
new/usr/src/cmd/mandoc/mdoc_validate.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
   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

  22 #ifndef OSNAME
  23 #include <sys/utsname.h>
  24 #endif

  26 #include <sys/types.h>

  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>

  36 #include "mdoc.h"
  37 #include "mandoc.h"
  38 #include "libmdoc.h"
  39 #include "libmandoc.h"

  41 /* FIXME: .Bl -diag can't have non-text children in HEAD. */

  43 #define PRE_ARGS  struct mdoc *mdoc, struct mdoc_node *n
  44 #define POST_ARGS struct mdoc *mdoc

  46 #define NUMSIZ    32
  47 #define DATESIZE  32

  49 enum    check_ineq {
  50         CHECK_LT,
  51         CHECK_GT,
  52         CHECK_EQ
  53 };
_____unchanged_portion_omitted_

  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);
  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);

  79 static  int      ebool(POST_ARGS);
  80 static  int      berr_ge1(POST_ARGS);
  81 static  int      bwarn_ge1(POST_ARGS);
  82 static  int      ewarn_eq0(POST_ARGS);
  83 static  int      ewarn_eq1(POST_ARGS);
  84 static  int      ewarn_ge1(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_ge1(POST_ARGS);
  89 static  int      hwarn_le1(POST_ARGS);

  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);
 102 static  int      post_defaults(POST_ARGS);
 103 static  int      post_literal(POST_ARGS);
 103 static  int      post_eoln(POST_ARGS);
 104 static  int      post_hyph(POST_ARGS);
 105 static  int      post_ignpar(POST_ARGS);
 106 static  int      post_it(POST_ARGS);
 107 static  int      post_lb(POST_ARGS);
 108 static  int      post_literal(POST_ARGS);
 109 static  int      post_nm(POST_ARGS);
 110 static  int      post_ns(POST_ARGS);
 111 static  int      post_os(POST_ARGS);
 112 static  int      post_par(POST_ARGS);
 110 static  int      post_ignpar(POST_ARGS);
 113 static  int      post_prol(POST_ARGS);
 114 static  int      post_root(POST_ARGS);
 115 static  int      post_rs(POST_ARGS);
 116 static  int      post_sh(POST_ARGS);
 117 static  int      post_sh_body(POST_ARGS);
 118 static  int      post_sh_head(POST_ARGS);
 119 static  int      post_st(POST_ARGS);
 120 static  int      post_std(POST_ARGS);
 121 static  int      post_vt(POST_ARGS);
 122 static  int      pre_an(PRE_ARGS);
 123 static  int      pre_bd(PRE_ARGS);
 124 static  int      pre_bl(PRE_ARGS);
 125 static  int      pre_dd(PRE_ARGS);
 126 static  int      pre_display(PRE_ARGS);
 127 static  int      pre_dt(PRE_ARGS);
 128 static  int      pre_it(PRE_ARGS);
 129 static  int      pre_literal(PRE_ARGS);
 130 static  int      pre_os(PRE_ARGS);
 131 static  int      pre_par(PRE_ARGS);
```

```
132 static  int      pre_sh(PRE_ARGS);
133 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_ge1, NULL };
139 static  v_post   posts_bf[] = { hwarn_le1, post_bf, NULL };
140 static  v_post   posts_bk[] = { hwarn_eq0, bwarn_ge1, NULL };
141 static  v_post   posts_bl[] = { bwarn_ge1, 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_d1[] = { bwarn_ge1, post_hyph, NULL };
147 static  v_post   posts_dd[] = { post_dd, post_prol, NULL };
148 static  v_post   posts_dl[] = { post_literal, bwarn_ge1, NULL };
149 static  v_post   posts_dt[] = { post_dt, post_prol, NULL };
150 static  v_post   posts_fo[] = { hwarn_eq1, bwarn_ge1, NULL };
151 static  v_post   posts_hyph[] = { post_hyph, NULL };
152 static  v_post   posts_hyphtext[] = { ewarn_ge1, 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_ge1, post_hyph, NULL };
150 static  v_post   posts_nd[] = { berr_ge1, NULL };
156 static  v_post   posts_nm[] = { post_nm, NULL };
157 static  v_post   posts_notext[] = { ewarn_eq0, NULL };
158 static  v_post   posts_ns[] = { post_ns, NULL };
159 static  v_post   posts_os[] = { post_os, post_prol, NULL };
160 static  v_post   posts_pp[] = { post_par, ewarn_eq0, NULL };
161 static  v_post   posts_rs[] = { post_rs, NULL };
162 static  v_post   posts_sh[] = { post_ignpar,hwarn_ge1,post_sh,post_hyph,NULL };
163 static  v_post   posts_sp[] = { post_par, ewarn_le1, NULL };
164 static  v_post   posts_ss[] = { post_ignpar, hwarn_ge1, post_hyph, NULL };
156 static  v_post   posts_sh[] = { post_ignpar, hwarn_ge1, post_sh, NULL };
157 static  v_post   posts_sp[] = { ewarn_le1, NULL };
158 static  v_post   posts_ss[] = { post_ignpar, hwarn_ge1, NULL };
165 static  v_post   posts_st[] = { post_st, NULL };
166 static  v_post   posts_std[] = { post_std, NULL };
167 static  v_post   posts_text[] = { ewarn_ge1, NULL };
168 static  v_post   posts_text1[] = { ewarn_eq1, NULL };
169 static  v_post   posts_vt[] = { post_vt, NULL };
164 static  v_post   posts_wline[] = { bwarn_ge1, NULL };
170 static  v_pre    pres_an[] = { pre_an, NULL };
171 static  v_pre    pres_bd[] = { pre_display, pre_bd, pre_literal, pre_par, NULL }
172 static  v_pre    pres_bl[] = { pre_bl, pre_par, NULL };
173 static  v_pre    pres_d1[] = { pre_display, NULL };
174 static  v_pre    pres_dl[] = { pre_literal, pre_display, NULL };
175 static  v_pre    pres_dd[] = { pre_dd, NULL };
176 static  v_pre    pres_dt[] = { pre_dt, NULL };
172 static  v_pre    pres_er[] = { NULL, NULL };
173 static  v_pre    pres_fd[] = { NULL, NULL };
177 static  v_pre    pres_it[] = { pre_it, pre_par, NULL };
178 static  v_pre    pres_os[] = { pre_os, NULL };
179 static  v_pre    pres_pp[] = { pre_par, NULL };
180 static  v_pre    pres_sh[] = { pre_sh, NULL };
181 static  v_pre    pres_ss[] = { pre_ss, NULL };
182 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_d1, posts_d1 },                  /* D1 */
188         { pres_pp, posts_notext },              /* Pp */
189         { pres_d1, posts_wline },               /* D1 */
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 */
202         { pres_er, NULL },                      /* Er */
206         { NULL, NULL },                         /* Ev */
207         { pres_std, posts_std },                /* Ex */
208         { NULL, NULL },                         /* Fa */
209         { NULL, posts_text },                   /* Fd */
206         { pres_fd, posts_text },                /* Fd */
210         { NULL, NULL },                         /* Fl */
211         { NULL, NULL },                         /* Fn */
212         { NULL, NULL },                         /* Ft */
213         { NULL, NULL },                         /* Ic */
214         { NULL, posts_text1 },                  /* In */
215         { NULL, posts_defaults },               /* Li */
216         { NULL, posts_nd },                     /* Nd */
217         { NULL, posts_nm },                     /* Nm */
218         { NULL, NULL },                         /* Op */
219         { NULL, NULL },                         /* Ot */
220         { NULL, posts_defaults },               /* Pa */
221         { pres_std, posts_std },                /* Rv */
222         { NULL, posts_st },                     /* St */
223         { NULL, NULL },                         /* Va */
224         { NULL, posts_vt },                     /* Vt */
225         { NULL, posts_text },                   /* Xr */
226         { NULL, posts_text },                   /* %A */
227         { NULL, posts_hyphtext },               /* %B */ /* FIXME: can be used o
224         { NULL, posts_text },                   /* %B */ /* FIXME: can be used o
228         { NULL, posts_text },                   /* %D */
229         { NULL, posts_text },                   /* %I */
230         { NULL, posts_text },                   /* %J */
231         { NULL, posts_hyphtext },               /* %N */
232         { NULL, posts_hyphtext },               /* %O */
228         { NULL, posts_text },                   /* %N */
229         { NULL, posts_text },                   /* %O */
233         { NULL, posts_text },                   /* %P */
234         { NULL, posts_hyphtext },               /* %R */
235         { NULL, posts_hyphtext },               /* %T */ /* FIXME: can be used o
231         { NULL, posts_text },                   /* %R */
232         { NULL, posts_text },                   /* %T */ /* FIXME: can be used o
236         { NULL, posts_text },                   /* %V */
237         { NULL, NULL },                         /* Ac */
238         { NULL, NULL },                         /* Ao */
239         { NULL, NULL },                         /* Aq */
240         { NULL, posts_at },                     /* At */
241         { NULL, NULL },                         /* Bc */
242         { NULL, posts_bf },                     /* Bf */
243         { NULL, NULL },                         /* Bo */
244         { NULL, NULL },                         /* Bq */
245         { NULL, NULL },                         /* Bsx */
246         { NULL, posts_bx },                     /* Bx */
247         { NULL, posts_bool },                   /* Db */
```

```
248         { NULL, NULL },                         /* Dc */
249         { 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 */
272         { NULL, NULL },                         /* Sx */
276         { NULL, NULL },                         /* Sy */
277         { NULL, NULL },                         /* Tn */
278         { NULL, NULL },                         /* Ux */
279         { NULL, NULL },                         /* Xc */
280         { NULL, NULL },                         /* Xo */
281         { NULL, posts_fo },                     /* Fo */
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)
516 check_args(struct mdoc *m, struct mdoc_node *n)
520 {
521         int              i;

523         if (NULL == n->args)
524                 return;

526         assert(n->args->argc);
527         for (i = 0; i < (int)n->args->argc; i++)
528                 check_argv(mdoc, n, &n->args->argv[i]);
525                 check_argv(m, n, &n->args->argv[i]);
529 }

531 static void
532 check_argv(struct mdoc *mdoc, struct mdoc_node *n, struct mdoc_argv *v)
529 check_argv(struct mdoc *m, struct mdoc_node *n, struct mdoc_argv *v)
533 {
534         int              i;

536         for (i = 0; i < (int)v->sz; i++)
537                 check_text(mdoc, v->line, v->pos, v->value[i]);
534                 check_text(m, v->line, v->pos, v->value[i]);

539         /* FIXME: move to post_std(). */

541         if (MDOC_Std == v->arg)
542                 if ( ! (v->sz || mdoc->meta.name))
543                         mdoc_nmsg(mdoc, n, MANDOCERR_NONAME);
539                 if ( ! (v->sz || m->meta.name))
540                         mdoc_nmsg(m, n, MANDOCERR_NONAME);
544 }

546 static void
547 check_text(struct mdoc *mdoc, int ln, int pos, char *p)
544 check_text(struct mdoc *m, int ln, int pos, char *p)
548 {
549         char            *cp;

551         if (MDOC_LITERAL & mdoc->flags)
548         if (MDOC_LITERAL & m->flags)
552                 return;

554         for (cp = p; NULL != (p = strchr(p, '\t')); p++)
555                 mdoc_pmsg(mdoc, ln, pos + (int)(p - cp), MANDOCERR_BADTAB);
552                 mdoc_pmsg(m, ln, pos + (int)(p - cp), MANDOCERR_BADTAB);
556 }
```

*unchanged_portion_omitted_*

```
594 static int
595 pre_bl(PRE_ARGS)
596 {
597         int              i, comp, dup;
598         const char      *offs, *width;
599         enum mdoc_list   lt;
600         struct mdoc_node *np;

602         if (MDOC_BLOCK != n->type) {
603                 if (ENDBODY_NOT != n->end) {
604                         assert(n->pending);
605                         np = n->pending->parent;
606                 } else
607                         np = n->parent;
```



```
609                 assert(np);
610                 assert(MDOC_BLOCK == np->type);
611                 assert(MDOC_Bl == np->tok);
612                 return(1);
613         }

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          */

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.offs);
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.offs = 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.offs ||
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)
741             if (n->norm->Bl.width)
742                     break;
746                     mdoc_nmsg(mdoc, n, MANDOCERR_NOWIDTHARG);
747             break;
748     case (LIST_column):
749             /* FALLTHROUGH */
750     case (LIST_diag):
751             /* FALLTHROUGH */
752     case (LIST_ohang):
753             /* FALLTHROUGH */
754     case (LIST_inset):
755             /* FALLTHROUGH */
756     case (LIST_item):
757             if (n->norm->Bl.width)
758                     mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
759             break;
760     case (LIST_bullet):
761             /* FALLTHROUGH */
762     case (LIST_dash):
763             /* FALLTHROUGH */
764     case (LIST_hyphen):
765             if (NULL == n->norm->Bl.width)
766                     n->norm->Bl.width = "2n";
767             break;
768     case (LIST_enum):
769             if (NULL == n->norm->Bl.width)
770                     n->norm->Bl.width = "3n";
771             break;
772     default:
773             break;
774     }

776     return(1);
777 }
______unchanged_portion_omitted_


886 static int
887 pre_sh(PRE_ARGS)
888 {

890     if (MDOC_BLOCK != n->type)
891             return(1);

878     roff_regunset(mdoc->roff, REG_nS);
892     return(check_parent(mdoc, n, MDOC_MAX, MDOC_ROOT));
893 }
______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)
1114         /* If no child specified, make sure we have the meta name. */
```

```
1116         if (NULL == mdoc->last->child && NULL == mdoc->meta.name) {
1117                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NONAME);
1128                 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;
1337         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 */
1547         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):
1549         if (MDOC_HEAD == mdoc->last->type)
1603                 return(post_bl_head(mdoc));
1604         case (MDOC_BODY):
1605                 break;
1606         default:
1551         if (MDOC_BLOCK == mdoc->last->type)
1552                 return(post_bl_block(mdoc));
1553         if (MDOC_BODY != mdoc->last->type)
1607                 return(1);
1608         }

1610         nchild = nbody->child;
1611         while (NULL != nchild) {
1612                 if (MDOC_It == nchild->tok || MDOC_Sm == nchild->tok) {
1613                         nchild = nchild->next;
1556         for (n = mdoc->last->child; n; n = n->next) {
1557                 switch (n->tok) {
1558                 case (MDOC_Lp):
1559                         /* FALLTHROUGH */
1560                 case (MDOC_Pp):
1561                         mdoc_nmsg(mdoc, n, MANDOCERR_CHILD);
```

```
1562                        /* FALLTHROUGH */
1563                case (MDOC_It):
1564                        /* FALLTHROUGH */
1565                case (MDOC_Sm):
1614                        continue;
1567                default:
1568                        break;
1615                }

1617                mdoc_nmsg(mdoc, nchild, MANDOCERR_CHILD);

1619                /*
1620                 * Move the node out of the Bl block.
1621                 * First, collect all required node pointers.
1622                 */

1624                nblock  = nbody->parent;
1625                nprev   = nblock->prev;
1626                nparent = nblock->parent;
1627                nnext   = nchild->next;

1629                /*
1630                 * Unlink this child.
1631                 */

1633                assert(NULL == nchild->prev);
1634                if (0 == --nbody->nchild) {
1635                        nbody->child = NULL;
1636                        nbody->last  = NULL;
1637                        assert(NULL == nnext);
1638                } else {
1639                        nbody->child = nnext;
1640                        nnext->prev = NULL;
1571                mdoc_nmsg(mdoc, n, MANDOCERR_SYNTCHILD);
1572                return(0);
1641                }

1643                /*
1644                 * Relink this child.
1645                 */

1647                nchild->parent = nparent;
1648                nchild->prev   = nprev;
1649                nchild->next   = nblock;

1651                nblock->prev = nchild;
1652                nparent->nchild++;
1653                if (NULL == nprev)
1654                        nparent->child = nchild;
1655                else
1656                        nprev->next = nchild;

1658                nchild = nnext;
1659        }

1661        return(1);
1662 }

1664 static int
1665 ebool(struct mdoc *mdoc)
1666 {

1668        if (NULL == mdoc->last->child) {
1669                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_MACROEMPTY);
1670                mdoc_node_delete(mdoc, mdoc->last);
1671                return(1);
```



```
1672        }
1673        check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 1);

1675        assert(MDOC_TEXT == mdoc->last->child->type);

1677        if (0 == strcmp(mdoc->last->child->string, "on")) {
1678                if (MDOC_Sm == mdoc->last->tok)
1679                        mdoc->flags &= ~MDOC_SMOFF;
1591        if (0 == strcmp(mdoc->last->child->string, "on"))
1680                return(1);
1681        }
1682        if (0 == strcmp(mdoc->last->child->string, "off")) {
1683                if (MDOC_Sm == mdoc->last->tok)
1684                        mdoc->flags |= MDOC_SMOFF;
1593        if (0 == strcmp(mdoc->last->child->string, "off"))
1685                return(1);
1686        }

1688        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADBOOL);
1689        return(1);
1690 }
______unchanged_portion_omitted_

1866 /*
1867  * For some arguments of some macros,
1868  * convert all breakable hyphens into ASCII_HYPH.
1869  */
1870 static int
1871 post_hyph(POST_ARGS)
1872 {
1873        struct mdoc_node        *n, *nch;
1874        char                    *cp;

1876        n = mdoc->last;
1877        switch (n->type) {
1878        case (MDOC_HEAD):
1879                if (MDOC_Sh == n->tok || MDOC_Ss == n->tok)
1880                        break;
1881                return(1);
1882        case (MDOC_BODY):
1883                if (MDOC_D1 == n->tok || MDOC_Nd == n->tok)
1884                        break;
1885                return(1);
1886        case (MDOC_ELEM):
1887                break;
1888        default:
1889                return(1);
1890        }

1892        for (nch = n->child; nch; nch = nch->next) {
1893                if (MDOC_TEXT != nch->type)
1894                        continue;
1895                cp = nch->string;
1896                if (3 > strnlen(cp, 3))
1897                        continue;
1898                while ('\0' != *(++cp))
1899                        if ('-' == *cp &&
1900                            isalpha((unsigned char)cp[-1]) &&
1901                            isalpha((unsigned char)cp[1]))
1902                                *cp = ASCII_HYPH;
1903        }
1904        return(1);
1905 }

1907 static int
1908 post_ns(POST_ARGS)
```

```
1909 {

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                 mandoc_msg(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
```



```
2171        return(1);
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 = mandoc_a2msec(nn->string);
2273         if (cp) {
2274                 mdoc->meta.vol = mandoc_strdup(cp);
2275                 mdoc->meta.msec = mandoc_strdup(nn->string);
2276         } else {
2277                 mdoc_nmsg(mdoc, n, MANDOCERR_BADMSEC);
2278                 mdoc->meta.vol = mandoc_strdup(nn->string);
2279                 mdoc->meta.msec = mandoc_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 = mandoc_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 = mandoc_strdup(nn->string);
2302                 } else
2303                         mdoc->meta.arch = mandoc_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\"", if 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 = mandoc_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 = mandoc_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 = mandoc_strdup(buf);
2411         return(1);
2412 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5106 Wed Jul 30 20:55:10 2014
new/usr/src/cmd/mandoc/msec.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

LINE("1",               "User Commands")
LINE("1B",              "BSD Compatibility Package Commands")
LINE("1B",              "illumos/BSD Compatibility Package Commands")
LINE("1b",              "illumos/BSD Compatibility Package Commands")
LINE("1C",              "Communication Commands")
LINE("1c",              "Communication Commands")
LINE("1F",              "FMLI Commands")
LINE("1f",              "FMLI Commands")
LINE("1G",              "Graphics and CAD Commands")
LINE("1g",              "Graphics and CAD Commands")
LINE("1HAS",            "User Commands")
LINE("1has",            "User Commands")
LINE("1M",              "Maintenance Commands")
LINE("1m",              "Maintenance Commands")
LINE("1S",              "illumos Specific Commands")
LINE("1s",              "illumos Specific Commands")
LINE("2",               "System Calls")
LINE("3",               "Introduction to Library Functions")
LINE("3BSDMALLOC",      "BSD Memory Allocation Library")
LINE("3AIO",            "Asynchronous I/O Library Functions")
LINE("3aio",            "Asynchronous I/O Library Functions")
LINE("3BSM",            "Security and Auditing Library Functions")
LINE("3bsm",            "Security and Auditing Library Functions")
LINE("3C",              "Standard C Library Functions")
LINE("3C++",            "C++ Library Functions")
LINE("3c",              "Standard C Library Functions")
LINE("3C_DB",           "Threads Debugging Library Functions")
LINE("3C_db",           "Threads Debugging Library Functions")
LINE("3CFGADM",         "Configuration Administration Library Functions")
LINE("3COMMPUTIL",      "Communication Protocol Parser Utilities Library Functio
LINE("3cfgadm",         "Configuration Administration Library Functions")
LINE("3COMPPUTIL",      "Communication Protocol Parser Utilities Library Functio
LINE("3compputil",      "Communication Protocol Parser Utilities Library Functio
LINE("3CONTRACT",       "Contract Management Library Functions")
LINE("3contract",       "Contract Management Library Functions")
LINE("3CPC",            "CPU Performance Counters Library Functions")
LINE("3cpc",            "CPU Performance Counters Library Functions")
LINE("3CURSES",         "Curses Library Functions")
LINE("3curses",         "Curses Library Functions")
LINE("3DAT",            "Direct Access Transport Library Functions")
LINE("3dat",            "Direct Access Transport Library Functions")
LINE("3DEVID",          "Device ID Library Functions")
LINE("3devid",          "Device ID Library Functions")
LINE("3DEVINFO",        "Device Information Library Functions")
```



```
LINE("3devinfo",        "Device Information Library Functions")
LINE("3DL",             "Dynamic Linking Library Functions")
LINE("3dl",             "Dynamic Linking Library Functions")
LINE("3DLPI",           "Data Link Provider Interface Library Functions")
LINE("3dlpi",           "Data Link Provider Interface Library Functions")
LINE("3DMI",            "DMI Library Functions")
LINE("3dmi",            "DMI Library Functions")
LINE("3DNS_SD",         "DNS Service Discovery Library Functions")
LINE("3dns_sd",         "DNS Service Discovery Library Functions")
LINE("3DOOR",           "Door Library Functions")
LINE("3door",           "Door Library Functions")
LINE("3ELF",            "ELF Library Functions")
LINE("3elf",            "ELF Library Functions")
LINE("3EXACCT",         "Extended Accounting File Access Library Functions")
LINE("3exacct",         "Extended Accounting File Access Library Functions")
LINE("3EXT",            "Extended Library Functions")
LINE("3ext",            "Extended Library Functions")
LINE("3FCOE",           "FCoE Port Management Library Functions")
LINE("3fcoe",           "FCoE Port Management Library Functions")
LINE("3FSTYP",          "File System Type Identification Library Functions")
LINE("3fstyp",          "File System Type Identification Library Functions")
LINE("3GEN",            "String Pattern-Matching Library Functions")
LINE("3gen",            "String Pattern-Matching Library Functions")
LINE("3GSS",            "Generic Security Services API Library Functions")
LINE("3gss",            "Generic Security Services API Library Functions")
LINE("3HEAD",           "Headers")
LINE("3head",           "Headers")
LINE("3ISCSIT",         "iSCSI Management Library Functions")
LINE("3iscsit",         "iSCSI Management Library Functions")
LINE("3KRB",            "Kerberos Library Functions")
LINE("3KRB5",           "MIT Kerberos 5 Library Functions")
LINE("3krb",            "Kerberos Library Functions")
LINE("3KSTAT",          "Kernel Statistics Library Functions")
LINE("3kstat",          "Kernel Statistics Library Functions")
LINE("3KVM",            "Kernel VM Library Functions")
LINE("3kvm",            "Kernel VM Library Functions")
LINE("3LDAP",           "LDAP Library Functions")
LINE("3ldap",           "LDAP Library Functions")
LINE("3LGRP",           "Locality Group Library Functions")
LINE("3lgrp",           "Locality Group Library Functions")
LINE("3LIB",            "Interface Libraries")
LINE("3lib",            "Interface Libraries")
LINE("3LIBUCB",         "illumos/BSD Compatibility Interface Libraries")
LINE("3libucb",         "illumos/BSD Compatibility Interface Libraries")
LINE("3M",              "Mathematical Library Functions")
LINE("3m",              "Mathematical Library Functions")
LINE("3MAIL",           "User Mailbox Library Functions")
LINE("3mail",           "User Mailbox Library Functions")
LINE("3MALLOC",         "Memory Allocation Library Functions")
LINE("3malloc",         "Memory Allocation Library Functions")
LINE("3MP",             "Multiple Precision Library Functions")
LINE("3mp",             "Multiple Precision Library Functions")
LINE("3MPAPI",          "Common Multipath Management Library Functions")
LINE("3mpapi",          "Common Multipath Management Library Functions")
LINE("3NSL",            "Networking Services Library Functions")
LINE("3nsl",            "Networking Services Library Functions")
LINE("3NVPAIR",         "Name-value Pair Library Functions")
LINE("3nvpair",         "Name-value Pair Library Functions")
LINE("3PAM",            "PAM Library Functions")
LINE("3pam",            "PAM Library Functions")
LINE("3PAPI",           "PAPI Library Functions")
LINE("3papi",           "PAPI Library Functions")
LINE("3PERL",           "Perl Library Functions")
LINE("3perl",           "Perl Library Functions")
LINE("3PICL",           "PICL Library Functions")
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("3sasl",          "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("3sysevent",      "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 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/out.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

22 #include <sys/types.h>

24 #include <assert.h>
25 #include <ctype.h>
26 #include <stdio.h>
27 #include <stdlib.h>
28 #include <string.h>
29 #include <time.h>

31 #include "mandoc.h"
32 #include "out.h"

34 static  void    tblcalc_data(struct rofftbl *, struct roffcol *,
35                         const struct tbl_opts *, const struct tbl_dat *);
35                         const struct tbl *, 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 *);
39                         const struct tbl *, const struct tbl_dat *);

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;

52         if ('\0' == *src)
53                 return(0);

55         i = hasd = 0;
```



```
57         switch (*src) {
58         case ('+'):
59                 src++;
60                 break;
61         case ('-'):
62                 buf[i++] = *src++;
63                 break;
64         default:
65                 break;
66         }

68         if ('\0' == *src)
69                 return(0);

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         }

83         if (BUFSIZ == i || (*src && *(src + 1)))
84                 return(0);

86         buf[i] = '\0';

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         }
127
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 }
134
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;
145         struct roffcol          *col;
146         int                      spans;
147
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          */
153
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));
158
159         hp = sp->head;
160
158         for ( ; sp; sp = sp->next) {
159                 if (TBL_SPAN_DATA != sp->pos)
160                         continue;
161                 spans = 1;
162                 /*
163                  * Account for the data cells in the layout, matching it
164                  * to data cells in the data section.
165                  */
166                 for (dp = sp->first; dp; dp = dp->next) {
167                         /* Do not used spanned cells in the calculation. */
168                         if (0 < --spans)
169                                 continue;
170                         spans = dp->spans;
171                         if (1 < spans)
172                                 continue;
173                         assert(dp->layout);
174                         col = &tbl->cols[dp->layout->head->ident];
175                         tblcalc_data(tbl, col, sp->opts, dp);
178                         tblcalc_data(tbl, col, sp->tbl, dp);
176                 }
177         }
178
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         }
178 }
179
180 static void
181 tblcalc_data(struct rofftbl *tbl, struct roffcol *col,
182                 const struct tbl_opts *opts, const struct tbl_dat *dp)
204                 const struct tbl *tp, const struct tbl_dat *dp)
183 {
184         size_t           sz;
185
186         /* Branch down into data sub-types. */
187
188         switch (dp->layout->pos) {
189         case (TBL_CELL_HORIZ):
190                 /* FALLTHROUGH */
191         case (TBL_CELL_DHORIZ):
192                 sz = (*tbl->len)(1, tbl->arg);
193                 if (col->width < sz)
194                         col->width = sz;
195                 break;
196         case (TBL_CELL_LONG):
197                 /* FALLTHROUGH */
198         case (TBL_CELL_CENTRE):
199                 /* FALLTHROUGH */
200         case (TBL_CELL_LEFT):
201                 /* FALLTHROUGH */
202         case (TBL_CELL_RIGHT):
203                 tblcalc_literal(tbl, col, dp);
204                 break;
205         case (TBL_CELL_NUMBER):
206                 tblcalc_number(tbl, col, opts, dp);
228                 tblcalc_number(tbl, col, tp, dp);
207                 break;
208         case (TBL_CELL_DOWN):
209                 break;
210         default:
211                 abort();
212                 /* NOTREACHED */
213         }
214 }
_____unchanged_portion_omitted_

230 static void
231 tblcalc_number(struct rofftbl *tbl, struct roffcol *col,
232                 const struct tbl_opts *opts, const struct tbl_dat *dp)
254                 const struct tbl *tp, const struct tbl_dat *dp)
233 {
234         int              i;
235         size_t           sz, psz, ssz, d;
236         const char      *str;
237         char            *cp;
238         char             buf[2];
239
240         /*
```

```
241          * First calculate number width and decimal place (last + 1 for
242          * 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;
276         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 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/preconv.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
   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 (MAP_FILE, etc.).
  36  */

  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          ssz;

 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         }
 266
 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);
 274                 fb->buf = mmap(NULL, fb->sz, PROT_READ,
 275                                 MAP_FILE|MAP_SHARED, fd, 0);
 271                 if (fb->buf != MAP_FAILED)
 272                         return(1);
 273         }
 274 #endif

 276         /*
 277          * If this isn't a regular file (like, say, stdin), then we must
 278          * go the old way and just read things in bit by bit.
 279          */

 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
 291                 if (off == fb->sz)
 292                         resize_buf(fb, 65536);

 294                 ssz = read(fd, fb->buf + (int)off, fb->sz - off);
 295                 if (ssz == 0) {
 296                         fb->sz = off;
 297                         return(1);
 298                 }
 299                 if (ssz == -1) {
 300                         perror(f);
 301                         break;
 302                 }
 303                 off += (size_t)ssz;
 304         }

 306         free(fb->buf);
 307         fb->buf = NULL;
 308         return(0);
 309 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2099 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/predefs.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/read.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
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

22 #ifdef HAVE_MMAP
23 # include <sys/stat.h>
24 # include <sys/mman.h>
25 #endif

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>

37 #include "mandoc.h"
38 #include "libmandoc.h"
39 #include "mdoc.h"
40 #include "man.h"
41 #include "main.h"

43 #ifndef MAP_FILE
44 #define MAP_FILE        0
45 #endif

43 #define REPARSE_LIMIT   1000

45 struct  buf {
46         char             *buf; /* binary input buffer */
47         size_t            sz; /* size of binary buffer */
48 };

50 struct  mparse {
51         enum mandoclevel  file_status; /* status of current parse */
52         enum mandoclevel  wlevel; /* ignore messages below this */
53         int               line; /* line number in the file */
```



```
54         enum mparset      inttype; /* which parser to use */
55         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 };

68 static  void      resize_buf(struct buf *, size_t);
69 static  void      mparse_buf_r(struct mparse *, struct buf, int);
73 static  void      mparse_readfd_r(struct mparse *, int, const char *, int);
70 static  void      pset(const char *, int, struct mparse *);
71 static  int       read_whole_file(const char *, int, struct buf *, int *);
72 static  void      mparse_end(struct mparse *);
73 static  void      mparse_parse_buffer(struct mparse *, struct buf,
74                         const char *);

76 static  const enum mandocerr    mandoclimits[MANDOCLEVEL_MAX] = {
77         MANDOCERR_OK,
78         MANDOCERR_WARNING,
79         MANDOCERR_WARNING,
80         MANDOCERR_ERROR,
81         MANDOCERR_FATAL,
82         MANDOCERR_MAX,
83         MANDOCERR_MAX
84 };

86 static  const char * const      mandocerrs[MANDOCERR_MAX] = {
87         "ok",

89         "generic warning",

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",

103        /* related to document structure */
104        ".so is fragile, better use ln(1)",
105        "NAME section must come first",
106        "bad NAME section contents",
108        "manual name not yet set",
107        "sections out of conventional order",
108        "duplicate section name",
109        "section header suited to sections 2, 3, and 9 only",
111        "section not in conventional manual section",

111        /* related to macros and nesting */
112        "skipping obsolete macro",
113        "skipping paragraph macro",
114        "moving paragraph macro out of list",
115        "skipping no-space macro",
116        "blocks badly nested",
```

```
117         "child violates parent syntax",
118         "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);
250                         curp->pmdoc = mdoc_alloc(curp->roff, curp);
254                 assert(curp->pmdoc);
255                 curp->mdoc = curp->pmdoc;
256                 return;
257         case (MPARSE_MAN):
258                 if (NULL == curp->pman)
259                         curp->pman = man_alloc(curp->roff, curp);
260                 assert(curp->pman);
261                 curp->man = curp->pman;
262                 return;
263         default:
264                 break;
```

```
265         }

267         if (pos >= 3 && 0 == memcmp(buf, ".Dd", 3))  {
268                 if (NULL == curp->pmdoc)
269                         curp->pmdoc = mdoc_alloc(curp->roff, curp,
270                                                  curp->defos);
266                         curp->pmdoc = mdoc_alloc(curp->roff, curp);
271                 assert(curp->pmdoc);
272                 curp->mdoc = curp->pmdoc;
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++;
341                                 if (pos >= (int)ln.sz)
342                                         resize_buf(&ln, 256);
354                                 ln.buf[pos++] = '?';
355                                 continue;
356                         }

358                         /* Trailing backslash = a plain char. */

360                         if ('\\' != blk.buf[i] || i + 1 == (int)blk.sz) {
350                                 if (pos >= (int)ln.sz)
351                                         resize_buf(&ln, 256);
361                                 ln.buf[pos++] = blk.buf[i++];
362                                 continue;
363                         }

365                         /*
366                          * Found escape and at least one other character.
367                          * When it's a newline character, skip it.
368                          * When there is a carriage return in between,
369                          * skip that one as well.
370                          */

372                         if ('\r' == blk.buf[i + 1] && i + 2 < (int)blk.sz &&
373                             '\n' == blk.buf[i + 2])
374                                 ++i;
375                         if ('\n' == blk.buf[i + 1]) {
376                                 i += 2;
377                                 ++lnn;
378                                 continue;
379                         }

381                         if ('"' == blk.buf[i + 1] || '#' == blk.buf[i + 1]) {
382                                 i += 2;
383                                 /* Comment, skip to end of line */
384                                 for (; i < (int)blk.sz; ++i) {
385                                         if ('\n' == blk.buf[i]) {
386                                                 ++i;
387                                                 ++lnn;
388                                                 break;
389                                         }
390                                 }
```

```
392                             /* Backout trailing whitespaces */
393                             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                     }

402                     /* Catch escaped bogus characters. */

404                     c = (unsigned char) blk.buf[i+1];

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                     }

415                     /* Some other escape sequence, copy & cont. */

395                     if (pos + 1 >= (int)ln.sz)
396                             resize_buf(&ln, 256);

417                     ln.buf[pos++] = blk.buf[i++];
418                     ln.buf[pos++] = blk.buf[i++];
419             }

421             if (pos >= (int)ln.sz)
422                     resize_buf(&ln, 256);

424             ln.buf[pos] = '\0';

426             /*
427              * A significant amount of complexity is contained by
428              * the roff preprocessor.  It's line-oriented but can be
429              * expressed on one line, so we need at times to
430              * readjust our starting point and re-run it.  The roff
431              * preprocessor can also readjust the buffers with new
432              * data, so we pass them in wholesale.
433              */

435             of = 0;

437             /*
438              * Maintain a lookaside buffer of all parsed lines.  We
439              * only do this if mparse_keep() has been invoked (the
440              * buffer may be accessed with mparse_getkeep()).
441              */

443             if (curp->secondary) {
444                     curp->secondary->buf =
445                             mandoc_realloc
446                             (curp->secondary->buf,
447                              curp->secondary->sz + pos + 2);
448                     memcpy(curp->secondary->buf +
449                                     curp->secondary->sz,
450                                     ln.buf, pos);
451                     curp->secondary->sz += pos;
452                     curp->secondary->buf
453                             [curp->secondary->sz] = '\n';
454                     curp->secondary->sz++;
```



```
455                     curp->secondary->buf
456                             [curp->secondary->sz] = '\0';
457             }
458 rerun:
459             rr = roff_parseln
460                     (curp->roff, curp->line,
461                      &ln.buf, &ln.sz, of, &of);

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);
472                     mparse_readfd_r(curp, -1, ln.buf + of, 1);
492                     if (MANDOCLEVEL_FATAL <= curp->file_status)
493                             break;
494                     pos = 0;
495                     continue;
496             default:
497                     break;
498             }

500             /*
501              * If we encounter errors in the recursive parse, make
502              * sure we don't continue parsing.
503              */

505             if (MANDOCLEVEL_FATAL <= curp->file_status)
506                     break;

508             /*
509              * If input parsers have not been allocated, do so now.
510              * We keep these instanced between parsers, but set them
511              * locally per parse routine since we can use different
512              * parsers with each one.
513              */

515             if ( ! (curp->man || curp->mdoc))
516                     pset(ln.buf + of, pos - of, curp);

518             /*
519              * Lastly, push down into the parsers themselves.  One
```

```
520                  * of these will have already been set in the pset()
521                  * 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                 if (fb->buf != MAP_FAILED)
599                         return(1);
600         }
601 #endif

603         /*
604          * If this isn't a regular file (like, say, stdin), then we must
605          * go the old way and just read things in bit by bit.
606          */

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)
646 mparse_parse_buffer(struct mparse *curp, struct buf blk, const char *file,
647                 int re)
665 {
666         const char      *svfile;
667         static int       recursion_depth;

669         if (64 < recursion_depth) {
670                 mandoc_msg(MANDOCERR_ROFFLOOP, curp, curp->line, 0, NULL);
671                 return;
672         }
```

```
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
678 mparse_readfd_r(struct mparse *curp, int fd, const char *file, int re)
703 {
704         struct buf       blk;
705         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 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/roff.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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

 22 #include <assert.h>
 23 #include <ctype.h>
 24 #include <stdio.h>
 25 #include <stdlib.h>
 26 #include <string.h>

 28 #include "mandoc.h"
 29 #include "libroff.h"
 30 #include "libmandoc.h"

 32 /* Maximum number of nested if-else conditionals. */
 33 #define RSTACK_MAX      128

 35 /* Maximum number of string expansions per line, to break infinite loops. */
 36 #define EXPAND_LIMIT    1000

 38 enum    rofft {
 39         ROFF_ad,
 40         ROFF_am,
 41         ROFF_ami,
 42         ROFF_am1,
 43         ROFF_cc,
 44         ROFF_de,
 45         ROFF_dei,
 46         ROFF_de1,
 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 };

 78 enum    roffrule {
 79         ROFFRULE_DENY,
 80         ROFFRULE_ALLOW
 73         ROFFRULE_ALLOW,
 74         ROFFRULE_DENY
 81 };

 83 /*
 78  * A single register entity.  If "set" is zero, the value of the
 79  * register should be the default one, which is per-register.
 80  * Registers are assumed to be unsigned ints for now.
 81  */
 82 struct  reg {
 83         int             set; /* whether set or not */
 84         unsigned int    u; /* unsigned integer */
 85 };

 87 /*
 84  * An incredibly-simple string buffer.
 85  */
 86 struct  roffstr {
 87         char            *p; /* nil-terminated buffer */
 88         size_t           sz; /* saved strlen(p) */
 89 };
______unchanged_portion_omitted_

100 /*
101  * A single number register as part of a singly-linked list.
102  */
103 struct  roffreg {
104         struct roffstr   key;
105         int              val;
106         struct roffreg  *next;
107 };

109 struct  roff {
110         enum mparset     parsetype; /* requested parse type */
111         struct mparse   *parse; /* parse point */
112         struct roffnode *last; /* leaf of stack */
113         enum roffrule    rstack[RSTACK_MAX]; /* stack of !'ie' rules */
114         char             control; /* control character */
115         int              rstackpos; /* position in rstack */
116         struct roffreg  *regtab; /* number registers */
109         struct reg       regs[REG__MAX];
117         struct roffkv   *strtab; /* user-defined strings & macros */
```

```
118         struct roffkv   *xmbtab; /* multi-byte trans table (‘tr’) */
119         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 rofft       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 rofft,
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_free1(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_getop(const char *, int *, char *);
192 static  int              roff_getregn(const struct roff *,
193                                 const char *, size_t);
194 static  const char      *roff_getstrn(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 rofft       roff_parse(struct roff *, const char *, int *);
202 static  enum rofferr     roff_parsetext(char **, size_t *, int, int *);
188 static  enum rofferr     roff_parsetext(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         { "am1", 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         { "de1", 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 },
227         { "it", roff_line_ignore, NULL, NULL, 0, NULL },
247         { "ne", roff_line_ignore, NULL, NULL, 0, NULL },
248         { "nh", roff_line_ignore, NULL, NULL, 0, NULL },
249         { "nr", roff_nr, NULL, NULL, 0, NULL },
250         { "ns", roff_line_ignore, NULL, NULL, 0, NULL },
251         { "ps", roff_line_ignore, NULL, NULL, 0, NULL },
252         { "rm", roff_rm, NULL, NULL, 0, NULL },
253         { "so", roff_so, NULL, NULL, 0, NULL },
254         { "ta", roff_line_ignore, NULL, NULL, 0, NULL },
255         { "tr", roff_tr, NULL, NULL, 0, NULL },
256         { "Dd", roff_Dd, NULL, NULL, 0, NULL },
257         { "TH", roff_TH, NULL, NULL, 0, NULL },
258         { "TS", roff_TS, NULL, NULL, 0, NULL },
259         { "TE", roff_TE, NULL, NULL, 0, NULL },
260         { "T&", roff_T_, NULL, NULL, 0, NULL },
261         { "EQ", roff_EQ, NULL, NULL, 0, NULL },
262         { "EN", roff_EN, NULL, NULL, 0, NULL },
263         { ".", roff_cblock, NULL, NULL, 0, NULL },
264         { "\\}", roff_ccond, NULL, NULL, 0, NULL },
265         { NULL, roff_userdef, NULL, NULL, 0, NULL },
266 };

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", "D1",
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_free1(struct roff *r)
408 {
409         struct tbl_node *tbl;
354         struct tbl_node *t;
410         struct eqn_node *e;
411         int              i;

413         while (NULL != (tbl = r->first_tbl)) {
414                 r->first_tbl = tbl->next;
415                 tbl_free(tbl);
358         while (NULL != (t = r->first_tbl)) {
359                 r->first_tbl = t->next;
360                 tbl_free(t);
416         }
```



```
418         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_free1(r);

454         r->control = 0;
395         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)
412 roff_alloc(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 /*
```

```
 * In the current line, expand user-defined strings ("\*")
 * and references to number registers ("\n").
 * Also check the syntax of other escape sequences.
 * Pre-filter each and every line for reserved words (one beginning with
 * '\*', e.g., '\*(ab').  These must be handled before the actual line
 * is processed.
 * This also checks the syntax of regular escapes.
 */
static enum rofferr
roff_res(struct roff *r, char **bufp, size_t *szp, int ln, int pos)
{
        char             ubuf[12]; /* buffer to print the number */
        enum mandoc_esc  esc;
        const char      *stesc; /* start of an escape sequence ('\\') */
        const char      *stnam; /* start of the name, after "[(*" */
        const char      *cp;    /* end of the name, e.g. before ']' */
        const char      *res;   /* the string to be substituted */
        char            *nbuf;  /* new buffer to copy bufp to */
        size_t           nsz;   /* size of the new buffer */
        size_t           maxl;  /* expected length of the escape name */
        size_t           naml;  /* actual length of the escape name */
        int              expand_count;  /* to avoid infinite loops */
        int              i, maxl, expand_count;
        size_t           nsz;
        char            *n;

        expand_count = 0;

again:
        cp = *bufp + pos;
        while (NULL != (cp = strchr(cp, '\\'))) {
                stesc = cp++;

                /*
                 * The second character must be an asterisk or an n.
                 * The second character must be an asterisk.
                 * If it isn't, skip it anyway:  It is escaped,
                 * so it can't start another escape sequence.
                 */

                if ('\0' == *cp)
                        return(ROFF_CONT);

                switch (*cp) {
                case ('*'):
                        res = NULL;
                        break;
                case ('n'):
                        res = ubuf;
                        break;
                default:
                        if (ESCAPE_ERROR != mandoc_escape(&cp, NULL, NULL))
                if ('*' != *cp) {
                        res = cp;
                        esc = mandoc_escape(&cp, NULL, NULL);
                        if (ESCAPE_ERROR != esc)
                                continue;
                        cp = res;
                        mandoc_msg
                                (MANDOCERR_BADESCAPE, r->parse,
                                 ln, (int)(stesc - *bufp), NULL);
                        return(ROFF_CONT);
                }

                cp++;
```



```
                /*
                 * The third character decides the length
                 * of the name of the string or register.
                 * of the name of the string.
                 * Save a pointer to the name.
                 */

                switch (*cp) {
                case ('\0'):
                        return(ROFF_CONT);
                case ('('):
                        cp++;
                        maxl = 2;
                        break;
                case ('['):
                        cp++;
                        maxl = 0;
                        break;
                default:
                        maxl = 1;
                        break;
                }
                stnam = cp;

                /* Advance to the end of the name. */

                for (naml = 0; 0 == maxl || naml < maxl; naml++, cp++) {
                for (i = 0; 0 == maxl || i < maxl; i++, cp++) {
                        if ('\0' == *cp) {
                                mandoc_msg
                                        (MANDOCERR_BADESCAPE,
                                         r->parse, ln,
                                         (int)(stesc - *bufp), NULL);
                                return(ROFF_CONT);
                        }
                        if (0 == maxl && ']' == *cp)
                                break;
                }

                /*
                 * Retrieve the replacement string; if it is
                 * undefined, resume searching for escapes.
                 */

                if (NULL == res)
                        res = roff_getstrn(r, stnam, naml);
                else
                        snprintf(ubuf, sizeof(ubuf), "%d",
                            roff_getregn(r, stnam, naml));
                res = roff_getstrn(r, stnam, (size_t)i);

                if (NULL == res) {
                        mandoc_msg
                                (MANDOCERR_BADESCAPE, r->parse,
                                 ln, (int)(stesc - *bufp), NULL);
                        res = "";
                }

                /* Replace the escape sequence by the string. */

                pos = stesc - *bufp;

                nsz = *szp + strlen(res) + 1;
                nbuf = mandoc_malloc(nsz);
                n = mandoc_malloc(nsz);
```

```
604                 strlcpy(nbuf, *bufp, (size_t)(stesc - *bufp + 1));
605                 strlcat(nbuf, res, nsz);
606                 strlcat(nbuf, cp + (maxl ? 0 : 1), nsz);
535                 strlcpy(n, *bufp, (size_t)(stesc - *bufp + 1));
536                 strlcat(n, res, nsz);
537                 strlcat(n, cp + (maxl ? 0 : 1), nsz);

608                 free(*bufp);

610                 *bufp = nbuf;
541                 *bufp = n;
611                 *szp = nsz;

613                 if (EXPAND_LIMIT >= ++expand_count)
614                         goto again;

616                 /* Just leave the string unexpanded. */
617                 mandoc_msg(MANDOCERR_ROFFLOOP, r->parse, ln, pos, NULL);
618                 return(ROFF_IGN);
619         }
620         return(ROFF_CONT);
621 }

623 /*
624  * Process text streams:
625  * Convert all breakable hyphens into ASCII_HYPH.
626  * Decrement and spring input line trap.
555  * Process text streams: convert all breakable hyphens into ASCII_HYPH.
627  */
628 static enum rofferr
629 roff_parsetext(char **bufp, size_t *szp, int pos, int *offs)
558 roff_parsetext(char *p)
630 {
631         size_t           sz;
632         const char      *start;
633         char            *p;
634         int              isz;
635         enum mandoc_esc  esc;

637         start = p = *bufp + pos;
564         start = p;

639         while ('\0' != *p) {
640                 sz = strcspn(p, "-\\");
641                 p += sz;

643                 if ('\0' == *p)
644                         break;

646                 if ('\\' == *p) {
647                         /* Skip over escapes. */
648                         p++;
649                         esc = mandoc_escape((const char **)&p, NULL, NULL);
576                         esc = mandoc_escape
577                                 ((const char **)&p, NULL, NULL);
650                         if (ESCAPE_ERROR == esc)
651                                 break;
652                         continue;
653                 } else if (p == start) {
654                         p++;
655                         continue;
656                 }

658                 if (isalpha((unsigned char)p[-1]) &&
659                     isalpha((unsigned char)p[1]))
660                         *p = ASCII_HYPH;
```



```
661                 p++;
662         }

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 }

683 enum rofferr
684 roff_parseln(struct roff *r, int ln, char **bufp,
685                 size_t *szp, int pos, int *offs)
686 {
687         enum rofft       t;
688         enum rofferr     e;
689         int              ppos, ctl;

691         /*
692          * Run the reserved-word filter only if we have some reserved
693          * words to fill in.
694          */

696         e = roff_res(r, bufp, szp, ln, pos);
697         if (ROFF_IGN == e)
698                 return(e);
699         assert(ROFF_CONT == e);

701         ppos = pos;
702         ctl = roff_getcontrol(r, *bufp, &pos);
614         ctl = mandoc_getcontrol(*bufp, &pos);

704         /*
705          * First, if a scope is open and we're not a macro, pass the
706          * text through the macro's filter.  If a scope isn't open and
707          * we're not a macro, just let it through.
708          * Finally, if there's an equation scope open, divert it into it
709          * no matter our state.
710          */

712         if (r->last && ! ctl) {
713                 t = r->last->tok;
714                 assert(roffs[t].text);
715                 e = (*roffs[t].text)
716                         (r, t, bufp, szp, ln, pos, pos, offs);
717                 assert(ROFF_IGN == e || ROFF_CONT == e);
718                 if (ROFF_CONT != e)
719                         return(e);
720         }
721         if (r->eqn)
722                 return(eqn_read(&r->eqn, ln, *bufp, ppos, offs));
723         if ( ! ctl) {
633                         return(eqn_read(&r->eqn, ln, *bufp, pos, offs));
724                 if (r->tbl)
```

```
725                         return(tbl_read(r->tbl, ln, *bufp, pos));
726                 return(roff_parsetext(bufp, szp, pos, offs));
727         }
636                 return(roff_parsetext(*bufp + pos));
637         } else if ( ! ctl) {
638                 if (r->eqn)
639                         return(eqn_read(&r->eqn, ln, *bufp, pos, offs));
640                 if (r->tbl)
641                         return(tbl_read(r->tbl, ln, *bufp, pos));
642                 return(roff_parsetext(*bufp + pos));
643         } else if (r->eqn)
644                 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)
781                 if (--r->last->endspan < 0)
865                         break;
866                 roffnode_pop(r);
867         }
868 }
```

unchanged_portion_omitted_

```
1060 /* ARGSUSED */
1061 static enum rofferr
1062 roff_cond_sub(ROFF_ARGS)
1063 {
1064         enum rofft       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.
 989          * If the macro is unknown, first check if it contains a closing
 990          * delimiter '\}'.  If it does, close out our scope and return
 991          * the currently-scoped rule (ignore or continue).  Else, drop
 992          * into the currently-scoped rule.
1075          */

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))
 997                 for ( ; NULL != (ep = strchr(ep, '\\')); ep++) {
 998                         ep++;
 999                         if ('}' != *ep)
1090                                 continue;

1092                 /*
1003                          * Make the \} go away.
1004                          * This is a little haphazard, as it's not quite
1005                          * clear how nroff does this.
1093                  * If we're at the end of line, then just chop
1094                  * off the \} and resize the buffer.
1095                  * If we aren't, then convert it to spaces.
1008                          * If we aren't, then conver it to spaces.
1096                  */

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);
1022         }

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
```

```
roff_cond(ROFF_ARGS)
{
        int              sv;
        enum roffrule    rule;

        roffnode_push(r, tok, NULL, ln, ppos);

        /*
         * An '.el' has no conditional body: it will consume the value
         * of the current rstack entry set in prior 'ie' calls or
         * defaults to DENY.
         *
         * If we're not an 'el', however, then evaluate the conditional.
         */

        r->last->rule = ROFF_el == tok ?
        rule = ROFF_el == tok ?
                (r->rstackpos < 0 ?
                 ROFFRULE_DENY : r->rstack[r->rstackpos--]) :
                roff_evalcond(*bufp, &pos);

        sv = pos;
        while (' ' == (*bufp)[pos])
                pos++;

        /*
         * Roff is weird.  If we have just white-space after the
         * conditional, it's considered the BODY and we exit without
         * really doing anything.  Warn about this.  It's probably
         * wrong.
         */

        if ('\0' == (*bufp)[pos] && sv != pos) {
                mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
                return(ROFF_IGN);
        }

        roffnode_push(r, tok, NULL, ln, ppos);

        r->last->rule = rule;

        /*
         * An if-else will put the NEGATION of the current evaluated
         * conditional into the stack of rules.
         */

        if (ROFF_ie == tok) {
                if (r->rstackpos == RSTACK_MAX - 1) {
                        mandoc_msg(MANDOCERR_MEM,
                                        r->parse, ln, ppos, NULL);
                        return(ROFF_ERR);
                }
                r->rstack[++r->rstackpos] =
                        ROFFRULE_DENY == r->last->rule ?
                        ROFFRULE_ALLOW : ROFFRULE_DENY;
        }

        /* If the parent has false as its rule, then so do we. */

        if (r->last->parent && ROFFRULE_DENY == r->last->parent->rule)
                r->last->rule = ROFFRULE_DENY;

        /*
         * Determine scope.
         * If there is nothing on the line after the conditional,
         * not even whitespace, use next-line scope.
```



```
         * Determine scope.  If we're invoked with "\{" trailing the
         * conditional, then we're in a multiline scope.  Else our scope
         * expires on the next line.
         */

        if ('\0' == (*bufp)[pos]) {
                r->last->endspan = 2;
                goto out;
        }
        r->last->endspan = 1;

        while (' ' == (*bufp)[pos])
                pos++;

        /* An opening brace requests multiline scope. */

        if ('\\' == (*bufp)[pos] && '{' == (*bufp)[pos + 1]) {
                r->last->endspan = -1;
                pos += 2;
                goto out;
        }

        /*
         * Anything else following the conditional causes
         * single-line scope.  Warn if the scope contains
         * nothing but trailing whitespace.
         * If there are no arguments on the line, the next-line scope is
         * assumed.
         */

        if ('\0' == (*bufp)[pos])
                mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
                return(ROFF_IGN);

        r->last->endspan = 1;
        /* Otherwise re-run the roff parser after recalculating. */

out:
        *offs = pos;
        return(ROFF_RERUN);
}
```

*unchanged_portion_omitted_*

```
void
roff_setreg(struct roff *r, const char *name, int val, char sign)
{
        struct roffreg  *reg;

        /* Search for an existing register with the same name. */
        reg = r->regtab;

        while (reg && strcmp(name, reg->key.p))
                reg = reg->next;

        if (NULL == reg) {
                /* Create a new register. */
                reg = mandoc_malloc(sizeof(struct roffreg));
                reg->key.p = mandoc_strdup(name);
                reg->key.sz = strlen(name);
                reg->val = 0;
                reg->next = r->regtab;
                r->regtab = reg;
        }

        if ('+' == sign)
                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;
1355         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);
1362         t = tbl_alloc(ppos, ln, r->parse);

1609         if (r->last_tbl)
1610                 r->last_tbl->next = tbl;
1365                 r->last_tbl->next = t;
1611         else
1612                 r->first_tbl = r->last_tbl = tbl;
1367                 r->first_tbl = r->last_tbl = t;

1614         r->tbl = r->last_tbl = tbl;
1369         r->tbl = r->last_tbl = t;
1615         return(ROFF_IGN);
1616 }

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->xmbtab, 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_strndup(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.
1473          * and null-terminate them.
1736          */
1737         cp = *bufp + pos;
1738         for (i = 0; i < 9; i++)
1739                 arg[i] = '\0' == *cp ? "" :
1740                     mandoc_getarg(r->parse, &cp, ln, &pos);

1742         /*
1743          * Expand macro arguments.
1744          */
1745         *szp = 0;
1746         n1 = 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(n1) - 3 + strlen(arg[i]) + 1;
1756                 n2 = mandoc_malloc(*szp);

1758                 strlcpy(n2, n1, (size_t)(cp - n1 + 1));
1759                 strlcat(n2, arg[i], *szp);
1760                 strlcat(n2, cp + 3, *szp);

1762                 cp = n2 + (cp - n1);
1763                 free(n1);
1764                 n1 = n2;
1765         }

1767         /*
1768          * Replace the macro invocation
1769          * by the expanded macro.
1770          */
1771         free(*bufp);
1772         *bufp = n1;
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 ('\\' != *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 ('\\' != *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 == strncmp(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 slighly 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 Wed Jul 30 20:55:11 2014
new/usr/src/cmd/mandoc/st.in
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
/*      $Id: st.in,v 1.22 2013/12/25 14:09:32 schwarze Exp $ */
/*      $Id: st.in,v 1.19 2012/02/26 21:47:09 schwarze Exp $ */
/*
 * Copyright (c) 2009, 2010 Kristaps Dzonsons <kristaps@bsd.lv>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

/*
 * This file defines the .St macro arguments.  If you add a new
 * standard, make sure that the left-and side corresponds to the .St
 * argument (like .St -p1003.1) and the right-hand side corresponds to
 * the formatted output string.
 *
 * Be sure to escape strings.
 * The non-breaking blanks prevent ending an output line right before
 * a number.  Groff prevent line breaks at the same places.
 *
 * REMEMBER TO ADD NEW STANDARDS TO MDOC.7!
 */

LINE("-p1003.1-88",     "IEEE Std 1003.1-1988 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1-90",     "IEEE Std 1003.1-1990 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1-96",     "ISO/IEC 9945-1:1996 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1-2001",   "IEEE Std 1003.1-2001 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1-2004",   "IEEE Std 1003.1-2004 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1-2008",   "IEEE Std 1003.1-2008 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1",        "IEEE Std 1003.1 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1b",       "IEEE Std 1003.1b (\\(lqPOSIX.1b\\(rq)")
LINE("-p1003.1b-93",    "IEEE Std 1003.1b-1993 (\\(lqPOSIX.1b\\(rq)")
LINE("-p1003.1c-95",    "IEEE Std 1003.1c-1995 (\\(lqPOSIX.1c\\(rq)")
LINE("-p1003.1d-99",    "IEEE Std 1003.1d-1999 (\\(lqPOSIX.1d\\(rq)")
LINE("-p1003.1g-2000",  "IEEE Std 1003.1g-2000 (\\(lqPOSIX.1g\\(rq)")
LINE("-p1003.1i-95",    "IEEE Std 1003.1i-1995 (\\(lqPOSIX.1i\\(rq)")
LINE("-p1003.1j-2000",  "IEEE Std 1003.1j-2000 (\\(lqPOSIX.1j\\(rq)")
LINE("-p1003.1q-2000",  "IEEE Std 1003.1q-2000 (\\(lqPOSIX.1q\\(rq)")
LINE("-p1003.2",        "IEEE Std 1003.2 (\\(lqPOSIX.2\\(rq)")
LINE("-p1003.1b",       "IEEE Std 1003.1b (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1b-93",    "IEEE Std 1003.1b-1993 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1c-95",    "IEEE Std 1003.1c-1995 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1g-2000",  "IEEE Std 1003.1g-2000 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.1i-95",    "IEEE Std 1003.1i-1995 (\\(lqPOSIX.1\\(rq)")
LINE("-p1003.2-92",     "IEEE Std 1003.2-1992 (\\(lqPOSIX.2\\(rq)")
LINE("-p1003.2a-92",    "IEEE Std 1003.2a-1992 (\\(lqPOSIX.2\\(rq)")
LINE("-p1387.2",        "IEEE Std 1387.2 (\\(lqPOSIX.7.2\\(rq)")
LINE("-p1387.2-95",     "IEEE Std 1387.2-1995 (\\(lqPOSIX.7.2\\(rq)")
LINE("-p1003.2",        "IEEE Std 1003.2 (\\(lqPOSIX.2\\(rq)")
LINE("-p1387.2",        "IEEE Std 1387.2 (\\(lqPOSIX.7.2\\(rq)")
LINE("-isoC",           "ISO/IEC 9899:1990 (\\(lqISO\\~C90\\(rq)")
```



```
LINE("-isoC-90",        "ISO/IEC 9899:1990 (\\(lqISO\\~C90\\(rq)")
LINE("-isoC-amd1",      "ISO/IEC 9899/AMD1:1995 (\\(lqISO\\~C90, Amendment 1\\(r
LINE("-isoC-tcor1",     "ISO/IEC 9899/TCOR1:1994 (\\(lqISO\\~C90, Technical Corr
LINE("-isoC-tcor2",     "ISO/IEC 9899/TCOR2:1995 (\\(lqISO\\~C90, Technical Corr
LINE("-isoC-99",        "ISO/IEC 9899:1999 (\\(lqISO\\~C99\\(rq)")
LINE("-isoC-2011",      "ISO/IEC 9899:2011 (\\(lqISO\\~C11\\(rq)")
LINE("-iso9945-1-90",   "ISO/IEC 9945-1:1990 (\\(lqPOSIX.1\\(rq)")
LINE("-iso9945-1-96",   "ISO/IEC 9945-1:1996 (\\(lqPOSIX.1\\(rq)")
LINE("-iso9945-2-93",   "ISO/IEC 9945-2:1993 (\\(lqPOSIX.2\\(rq)")
LINE("-ansiC",          "ANSI X3.159-1989 (\\(lqANSI\\~C89\\(rq)")
LINE("-ansiC-89",       "ANSI X3.159-1989 (\\(lqANSI\\~C89\\(rq)")
LINE("-ansiC-99",       "ANSI/ISO/IEC 9899-1999 (\\(lqANSI\\~C99\\(rq)")
LINE("-ieee754",        "IEEE Std 754-1985")
LINE("-iso8802-3",      "ISO 8802-3: 1989")
LINE("-iso8601",        "ISO 8601")
LINE("-ieee1275-94",    "IEEE Std 1275-1994 (\\(lqOpen Firmware\\(rq)")
LINE("-xpg3",           "X/Open Portability Guide Issue\\~3 (\\(lqXPG3\\(rq)")
LINE("-xpg4",           "X/Open Portability Guide Issue\\~4 (\\(lqXPG4\\(rq)")
LINE("-xpg4.2",         "X/Open Portability Guide Issue\\~4, Version\\~2 (\\(lqX
LINE("-xpg4.3",         "X/Open Portability Guide Issue\\~4, Version\\~3 (\\(lqX
LINE("-xbd5",           "X/Open Base Definitions Issue\\~5 (\\(lqXBD5\\(rq)")
LINE("-xcu5",           "X/Open Commands and Utilities Issue\\~5 (\\(lqXCU5\\(rq
LINE("-xsh4.2",         "X/Open System Interfaces and Headers Issue\\~4, Version
LINE("-xsh5",           "X/Open System Interfaces and Headers Issue\\~5 (\\(lqXS
LINE("-xns5",           "X/Open Networking Services Issue\\~5 (\\(lqXNS5\\(rq)")
LINE("-xns5.2",         "X/Open Networking Services Issue\\~5.2 (\\(lqXNS5.2\\(r
LINE("-xns5.2d2.0",     "X/Open Networking Services Issue\\~5.2 Draft\\~2.0 (\\(
LINE("-xcurses4.2",     "X/Open Curses Issue\\~4, Version\\~2 (\\(lqXCURSES4.2\\
LINE("-susv2",          "Version\\~2 of the Single UNIX Specification (\\(lqSUSv
LINE("-susv3",          "Version\\~3 of the Single UNIX Specification (\\(lqSUSv
LINE("-susv2",          "Version\\~2 of the Single UNIX Specification")
LINE("-susv3",          "Version\\~3 of the Single UNIX Specification")
LINE("-svid4",          "System\\~V Interface Definition, Fourth Edition (\\(lqS
```

```
**********************************************************
    4085 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/tbl.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: tbl.c,v 1.27 2013/05/31 22:08:09 schwarze Exp $ */
/*      $Id: tbl.c,v 1.26 2011/07/25 15:37:00 kristaps Exp $ */
/*
 * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "mandoc.h"
#include "libmandoc.h"
#include "libroff.h"

enum rofferr
tbl_read(struct tbl_node *tbl, int ln, const char *p, int offs)
{
        int              len;
        const char      *cp;

        cp = &p[offs];
        len = (int)strlen(cp);

        /*
         * If we're in the options section and we don't have a
         * terminating semicolon, assume we've moved directly into the
         * layout section.  No need to report a warning: this is,
         * apparently, standard behaviour.
         */

        if (TBL_PART_OPTS == tbl->part && len)
                if (';' != cp[len - 1])
                        tbl->part = TBL_PART_LAYOUT;

        /* Now process each logical section of the table.  */

        switch (tbl->part) {
        case (TBL_PART_OPTS):
                return(tbl_option(tbl, ln, p) ? ROFF_IGN : ROFF_ERR);
        case (TBL_PART_LAYOUT):
                return(tbl_layout(tbl, ln, p) ? ROFF_IGN : ROFF_ERR);
```



```
        case (TBL_PART_CDATA):
                return(tbl_cdata(tbl, ln, p) ? ROFF_TBL : ROFF_IGN);
        default:
                break;
        }

        /*
         * This only returns zero if the line is empty, so we ignore it
         * and continue on.
         */
        return(tbl_data(tbl, ln, p) ? ROFF_TBL : ROFF_IGN);
}

struct tbl_node *
tbl_alloc(int pos, int line, struct mparse *parse)
{
        struct tbl_node *tbl;
        struct tbl_node *p;

        tbl = mandoc_calloc(1, sizeof(struct tbl_node));
        tbl->line = line;
        tbl->pos = pos;
        tbl->parse = parse;
        tbl->part = TBL_PART_OPTS;
        tbl->opts.tab = '\t';
        tbl->opts.linesize = 12;
        tbl->opts.decimal = '.';
        return(tbl);
        p = mandoc_calloc(1, sizeof(struct tbl_node));
        p->line = line;
        p->pos = pos;
        p->parse = parse;
        p->part = TBL_PART_OPTS;
        p->opts.tab = '\t';
        p->opts.linesize = 12;
        p->opts.decimal = '.';
        return(p);
}

void
tbl_free(struct tbl_node *tbl)
tbl_free(struct tbl_node *p)
{
        struct tbl_row  *rp;
        struct tbl_cell *cp;
        struct tbl_span *sp;
        struct tbl_dat  *dp;
        struct tbl_head *hp;

        while (NULL != (rp = tbl->first_row)) {
                tbl->first_row = rp->next;
        while (NULL != (rp = p->first_row)) {
                p->first_row = rp->next;
                while (rp->first) {
                        cp = rp->first;
                        rp->first = cp->next;
                        free(cp);
                }
                free(rp);
        }

        while (NULL != (sp = tbl->first_span)) {
                tbl->first_span = sp->next;
        while (NULL != (sp = p->first_span)) {
                p->first_span = sp->next;
                while (sp->first) {
```

```
                        dp = sp->first;
                        sp->first = dp->next;
                        if (dp->string)
                                free(dp->string);
                        free(dp);
                }
                free(sp);
        }

        while (NULL != (hp = tbl->first_head)) {
                tbl->first_head = hp->next;
        while (NULL != (hp = p->first_head)) {
                p->first_head = hp->next;
                free(hp);
        }

        free(tbl);
        free(p);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    6344 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/tbl_data.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: tbl_data.c,v 1.27 2013/06/01 04:56:50 schwarze Exp $ */
/*      $Id: tbl_data.c,v 1.24 2011/03/20 16:02:05 kristaps Exp $ */
/*
 * Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "mandoc.h"
#include "libmandoc.h"
#include "libroff.h"

static  int              data(struct tbl_node *, struct tbl_span *,
                                int, const char *, int *);
static  struct tbl_span *newspan(struct tbl_node *, int,
                                struct tbl_row *);

static int
data(struct tbl_node *tbl, struct tbl_span *dp,
                int ln, const char *p, int *pos)
{
        struct tbl_dat  *dat;
        struct tbl_cell *cp;
        int              sv, spans;

        cp = NULL;
        if (dp->last && dp->last->layout)
                cp = dp->last->layout->next;
        else if (NULL == dp->last)
                cp = dp->layout->first;

        /*
         * Skip over spanners, since
         * Skip over spanners and vertical lines to data formats, since
         * we want to match data with data layout cells in the header.
         */

        while (cp && TBL_CELL_SPAN == cp->pos)
        while (cp && (TBL_CELL_VERT == cp->pos ||
                                TBL_CELL_DVERT == cp->pos ||
                                TBL_CELL_SPAN == cp->pos))
                cp = cp->next;

        /*
         * Stop processing when we reach the end of the available layout
         * cells.  This means that we have extra input.
         */

        if (NULL == cp) {
                mandoc_msg(MANDOCERR_TBLEXTRADAT,
                                tbl->parse, ln, *pos, NULL);
                /* Skip to the end... */
                while (p[*pos])
                        (*pos)++;
                return(1);
        }

        dat = mandoc_calloc(1, sizeof(struct tbl_dat));
        dat->layout = cp;
        dat->pos = TBL_DATA_NONE;

        assert(TBL_CELL_SPAN != cp->pos);

        for (spans = 0, cp = cp->next; cp; cp = cp->next)
                if (TBL_CELL_SPAN == cp->pos)
                        spans++;
                else
                        break;

        dat->spans = spans;

        if (dp->last) {
                dp->last->next = dat;
                dp->last = dat;
        } else
                dp->last = dp->first = dat;

        sv = *pos;
        while (p[*pos] && p[*pos] != tbl->opts.tab)
                (*pos)++;

        /*
         * Check for a continued-data scope opening.  This consists of a
         * trailing 'T{' at the end of the line.  Subsequent lines,
         * until a standalone 'T}', are included in our cell.
         */

        if (*pos - sv == 2 && 'T' == p[sv] && '{' == p[sv + 1]) {
                tbl->part = TBL_PART_CDATA;
                return(1);
                return(0);
        }

        assert(*pos - sv >= 0);

        dat->string = mandoc_malloc((size_t)(*pos - sv + 1));
        memcpy(dat->string, &p[sv], (size_t)(*pos - sv));
        dat->string[*pos - sv] = '\0';

        if (p[*pos])
                (*pos)++;

        if ( ! strcmp(dat->string, "_"))
                dat->pos = TBL_DATA_HORIZ;
        else if ( ! strcmp(dat->string, "="))
```

```
120                 dat->pos = TBL_DATA_DHORIZ;
121         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;
190         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 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/tbl_html.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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_tblopen(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 }
```

*_____unchanged_portion_omitted_*

```
 89 void
 90 print_tbl(struct html *h, const struct tbl_span *sp)
 91 {
 92         const struct tbl_head *hp;
 93         const struct tbl_dat *dp;
 94         struct htmlpair  tag;
 95         struct tag      *tt;

 97         /* Inhibit printing of spaces: we do padding ourselves. */

 99         if (NULL == h->tblt)
100                 html_tblopen(h, sp);

102         assert(h->tblt);

104         h->flags |= HTML_NONOSPACE;
```



```
105         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):
122                         if (NULL == dp)
123                                 break;
124                         if (TBL_CELL_DOWN != dp->layout->pos)
125                                 if (dp->string)
126                                         print_text(h, dp->string);
127                         dp = dp->next;
134                                 break;
128                 }
136                 }
129                 break;
130         }

132         print_tagq(h, tt);

134         h->flags &= ~HTML_NONOSPACE;

136         if (TBL_SPAN_LAST & sp->flags) {
137                 assert(h->tbl.cols);
138                 free(h->tbl.cols);
139                 h->tbl.cols = NULL;
140                 print_tblclose(h);
141         }

143 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    8025 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/tbl_layout.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
 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

22 #include <assert.h>
23 #include <ctype.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <time.h>

28 #include "mandoc.h"
29 #include "libmandoc.h"
30 #include "libroff.h"

32 struct  tbl_phrase {
33         char             name;
34         enum tbl_cellt   key;
35 };

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  */

43 #define KEYS_MAX         11

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 }
54         { '=',           TBL_CELL_DHORIZ },
55         { '|',           TBL_CELL_VERT }
56 };
```



```
58 static  int              mods(struct tbl_node *, struct tbl_cell *,
59                                 int, const char *, int *);
60 static  int              cell(struct tbl_node *, struct tbl_row *,
61                                 int, const char *, int *);
62 static  void             row(struct tbl_node *, int, const char *, int *);
63 static  struct tbl_cell *cell_alloc(struct tbl_node *, struct tbl_row *,
64                                 enum tbl_cellt, int vert);
63 static  struct tbl_cell *cell_alloc(struct tbl_node *,
64                                 struct tbl_row *, enum tbl_cellt);
65 static  void             head_adjust(const struct tbl_cell *,
66                                 struct tbl_head *);

66 static int
67 mods(struct tbl_node *tbl, struct tbl_cell *cp,
68                 int ln, const char *p, int *pos)
69 {
70         char             buf[5];
71         int              i;

73         /* Not all types accept modifiers. */

75         switch (cp->pos) {
76         case (TBL_CELL_DOWN):
77                 /* FALLTHROUGH */
78         case (TBL_CELL_HORIZ):
79                 /* FALLTHROUGH */
80         case (TBL_CELL_DHORIZ):
83                 /* FALLTHROUGH */
84         case (TBL_CELL_VERT):
85                 /* FALLTHROUGH */
86         case (TBL_CELL_DVERT):
81                 return(1);
82         default:
83                 break;
84         }

86 mod:
87         /*
88          * XXX: since, at least for now, modifiers are non-conflicting
89          * (are separable by value, regardless of position), we let
90          * modifiers come in any order.  The existing tbl doesn't let
91          * this happen.
92          */
93         switch (p[*pos]) {
94         case ('\0'):
95                 /* FALLTHROUGH */
96         case (' '):
97                 /* FALLTHROUGH */
98         case ('\t'):
99                 /* FALLTHROUGH */
100        case (','):
101                /* FALLTHROUGH */
102        case ('.'):
103                return(1);
104        default:
105                break;
106        }

108        /* Throw away parenthesised expression. */

110        if ('(' == p[*pos]) {
111                (*pos)++;
112                while (p[*pos] && ')' != p[*pos])
113                        (*pos)++;
114                if (')' == p[*pos]) {
```

```
115                         (*pos)++;
116                         goto mod;
117                 }
118                 mandoc_msg(MANDOCERR_TBLLAYOUT,
119                                 tbl->parse, ln, *pos, NULL);
120                 return(0);
121         }

123         /* Parse numerical spacing from modifier string. */

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';

133                 /* No greater than 4 digits. */

135                 if (4 == i) {
136                         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
137                                         ln, *pos, NULL);
138                         return(0);
139                 }

141                 *pos += i;
142                 cp->spacing = (size_t)atoi(buf);

144                 goto mod;
145                 /* NOTREACHED */
146         }

148         /* TODO: GNU has many more extensions. */

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_TBLLAYOUT, 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         }

202         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
203                         ln, *pos - 1, NULL);
204         return(0);
205 }

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;
217         int              i;
212         enum tbl_cellt   c;

214         /* Handle vertical lines. */
220         /* Parse the column position ('r', 'R', '|', ...). */

216         for (vert = 0; '|' == p[*pos]; ++*pos)
217                 vert++;
218         while (' ' == p[*pos])
219                 (*pos)++;

221         /* Parse the column position ('c', 'l', 'r', ...). */

223         for (i = 0; i < KEYS_MAX; i++)
224                 if (tolower((unsigned char)p[*pos]) == keys[i].name)
225                         break;

227         if (KEYS_MAX == i) {
228                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
229                                 ln, *pos, NULL);
230                 return(0);
231         }

233         c = keys[i].key;

235         /*
236          * If a span cell is found first, raise a warning and abort the
237          * parse.  If a span cell is found and the last layout element
238          * isn't a "normal" layout, bail.
239          *
240          * FIXME: recover from this somehow?
241          */

243         if (TBL_CELL_SPAN == c) {
244                 if (NULL == rp->first) {
```

```
245                         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
246                                         ln, *pos, NULL);
247                         return(0);
248                 } else if (rp->last)
249                         switch (rp->last->pos) {
249                         case (TBL_CELL_VERT):
250                         case (TBL_CELL_DVERT):
250                         case (TBL_CELL_HORIZ):
251                         case (TBL_CELL_DHORIZ):
252                                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
253                                                 ln, *pos, NULL);
254                                 return(0);
255                         default:
256                                 break;
257                         }
258         }

260         /*
261          * If a vertical spanner is found, we may not be in the first
262          * row.
263          */

265         if (TBL_CELL_DOWN == c && rp == tbl->first_row) {
266                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse, ln, *pos, NULL);
267                 return(0);
268         }

270         (*pos)++;

273         /* Extra check for the double-vertical. */

275         if (TBL_CELL_VERT == c && '|' == p[*pos]) {
276                 (*pos)++;
277                 c = TBL_CELL_DVERT;
278         }
279
272         /* Disallow adjacent spacers. */

274         if (vert > 2) {
282         if (rp->last && (TBL_CELL_VERT == c || TBL_CELL_DVERT == c) &&
283                         (TBL_CELL_VERT == rp->last->pos ||
284                          TBL_CELL_DVERT == rp->last->pos)) {
275                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse, ln, *pos - 1, NULL);
276                 return(0);
277         }

279         /* Allocate cell then parse its modifiers. */

281         return(mods(tbl, cell_alloc(tbl, rp, c, vert), ln, p, pos));
291         return(mods(tbl, cell_alloc(tbl, rp, c), ln, p, pos));
282 }


285 static void
286 row(struct tbl_node *tbl, int ln, const char *p, int *pos)
287 {
288         struct tbl_row  *rp;

290 row:    /*
291          * EBNF describing this section:
292          *
293          * row          ::= row_list [:space:]* [.]?[\n]
294          * row_list     ::= [:space:]* row_elem row_tail
295          * row_tail     ::= [:space:]*[,] row_list |
296          *                  epsilon
297          * row_elem     ::= [\t\ ]*[:alpha:]+
```



```
298          */

300         rp = mandoc_calloc(1, sizeof(struct tbl_row));
301         if (tbl->last_row)
311         if (tbl->last_row) {
302                 tbl->last_row->next = rp;
303         else
304                 tbl->first_row = rp;
305         tbl->last_row = rp;
314         } else
315                 tbl->last_row = tbl->first_row = rp;

307 cell:
308         while (isspace((unsigned char)p[*pos]))
309                 (*pos)++;

311         /* Safely exit layout context. */

313         if ('.' == p[*pos]) {
314                 tbl->part = TBL_PART_DATA;
315                 if (NULL == tbl->first_row)
316                         mandoc_msg(MANDOCERR_TBLNOLAYOUT, tbl->parse,
317                                         ln, *pos, NULL);
318                 (*pos)++;
319                 return;
320         }

322         /* End (and possibly restart) a row. */

324         if (',' == p[*pos]) {
325                 (*pos)++;
326                 goto row;
327         } else if ('\0' == p[*pos])
328                 return;

330         if ( ! cell(tbl, rp, ln, p, pos))
331                 return;

333         goto cell;
334         /* NOTREACHED */
335 }
_____unchanged_portion_omitted_

349 static struct tbl_cell *
350 cell_alloc(struct tbl_node *tbl, struct tbl_row *rp, enum tbl_cellt pos,
351                 int vert)
360 cell_alloc(struct tbl_node *tbl, struct tbl_row *rp, enum tbl_cellt pos)
352 {
353         struct tbl_cell *p, *pp;
354         struct tbl_head *h, *hp;

356         p = mandoc_calloc(1, sizeof(struct tbl_cell));

358         if (NULL != (pp = rp->last)) {
359                 pp->next = p;
360                 h = pp->head->next;
361         } else {
362                 rp->first = p;
363                 h = tbl->first_head;
364         }
368                 rp->last->next = p;
365         rp->last = p;
370         } else
371                 rp->last = rp->first = p;

367         p->pos = pos;
```

```
368         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 || l
382          *      2  | c | l
383          *      3  l l
384          *      3  || c | l |.
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 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/tbl_term.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: tbl_term.c,v 1.25 2013/05/31 21:37:17 schwarze Exp $ */
/*      $Id: tbl_term.c,v 1.21 2011/09/20 23:05:49 schwarze Exp $ */
/*
 * Copyright (c) 2009, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 * Copyright (c) 2011, 2012 Ingo Schwarze <schwarze@openbsd.org>
 * Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mandoc.h"
#include "out.h"
#include "term.h"

static  size_t  term_tbl_len(size_t, void *);
static  size_t  term_tbl_strlen(const char *, void *);
static  void    tbl_char(struct termp *, char, size_t);
static  void    tbl_data(struct termp *, const struct tbl_opts *,
static  void    tbl_data(struct termp *, const struct tbl *,
                        const struct tbl_dat *,
                        const struct roffcol *);
static  size_t  tbl_rulewidth(struct termp *, const struct tbl_head *);
static  void    tbl_hframe(struct termp *, const struct tbl_span *, int);
static  void    tbl_literal(struct termp *, const struct tbl_dat *,
                        const struct roffcol *);
static  void    tbl_number(struct termp *, const struct tbl_opts *,
static  void    tbl_number(struct termp *, const struct tbl *,
                        const struct tbl_dat *,
                        const struct roffcol *);
static  void    tbl_hrule(struct termp *, const struct tbl_span *);
static  void    tbl_vrule(struct termp *, const struct tbl_head *);


static size_t
term_tbl_strlen(const char *p, void *arg)
{

        return(term_strlen((const struct termp *)arg, p));
}
```

*_____unchanged_portion_omitted_*


```
void
term_tbl(struct termp *tp, const struct tbl_span *sp)
{
        const struct tbl_head   *hp;
        const struct tbl_dat    *dp;
        struct roffcol          *col;
        int                      spans;
        size_t                   rmargin, maxrmargin;

        rmargin = tp->rmargin;
        maxrmargin = tp->maxrmargin;

        tp->rmargin = tp->maxrmargin = TERM_MAXMARGIN;

        /* Inhibit printing of spaces: we do padding ourselves. */

        tp->flags |= TERMP_NONOSPACE;
        tp->flags |= TERMP_NOSPACE;

        /*
         * The first time we're invoked for a given table block,
         * calculate the table widths and decimal positions.
         */

        if (TBL_SPAN_FIRST & sp->flags) {
                term_flushln(tp);

                tp->tbl.len = term_tbl_len;
                tp->tbl.slen = term_tbl_strlen;
                tp->tbl.arg = tp;

                tblcalc(&tp->tbl, sp);
        }

        /* Horizontal frame at the start of boxed tables. */

        if (TBL_SPAN_FIRST & sp->flags) {
                if (TBL_OPT_DBOX & sp->opts->opts)
                if (TBL_OPT_DBOX & sp->tbl->opts)
                        tbl_hframe(tp, sp, 1);
                if (TBL_OPT_DBOX & sp->opts->opts ||
                    TBL_OPT_BOX  & sp->opts->opts)
                if (TBL_OPT_DBOX & sp->tbl->opts ||
                    TBL_OPT_BOX  & sp->tbl->opts)
                        tbl_hframe(tp, sp, 0);
        }

        /* Vertical frame at the start of each row. */

        if (TBL_OPT_BOX & sp->opts->opts || TBL_OPT_DBOX & sp->opts->opts)
        if (TBL_OPT_BOX & sp->tbl->opts || TBL_OPT_DBOX & sp->tbl->opts)
                term_word(tp, TBL_SPAN_HORIZ == sp->pos ||
                        TBL_SPAN_DHORIZ == sp->pos ? "+" : "|");

        /*
         * Now print the actual data itself depending on the span type.
         * Spanner spans get a horizontal rule; data spanners have their
         * data printed by matching data to header.
         */

        switch (sp->pos) {
        case (TBL_SPAN_HORIZ):
                /* FALLTHROUGH */
        case (TBL_SPAN_DHORIZ):
                tbl_hrule(tp, sp);
                break;
```

```
124         case (TBL_SPAN_DATA):
125                 /* Iterate over template headers. */
126                 dp = sp->first;
127                 spans = 0;
128                 for (hp = sp->head; hp; hp = hp->next) {
130                         /*
131                          * If the current data header is invoked during
132                          * a spanner ("spans" > 0), don't emit anything
133                          * at all.
134                          */
134                         switch (hp->pos) {
135                         case (TBL_HEAD_VERT):
136                                 /* FALLTHROUGH */
137                         case (TBL_HEAD_DVERT):
138                                 if (spans <= 0)
139                                         tbl_vrule(tp, hp);
140                                 continue;
141                         case (TBL_HEAD_DATA):
142                                 break;
143                         }
136                         if (--spans >= 0)
137                                 continue;
139                         /* Separate columns. */
148                         /*
149                          * All cells get a leading blank, except the
150                          * first one and those after double rulers.
151                          */
141                         if (NULL != hp->prev)
142                                 tbl_vrule(tp, hp);
153                         if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
154                                 tbl_char(tp, ASCII_NBRSP, 1);
144                         col = &tp->tbl.cols[hp->ident];
145                         tbl_data(tp, sp->opts, dp, col);
157                         tbl_data(tp, sp->tbl, dp, col);
159                         /* No trailing blanks. */
161                         if (NULL == hp->next)
162                                 break;
147                         /*
165                          * Add another blank between cells,
166                          * or two when there is no vertical ruler.
167                          */
169                         tbl_char(tp, ASCII_NBRSP,
170                             TBL_HEAD_VERT  == hp->next->pos ||
171                             TBL_HEAD_DVERT == hp->next->pos ? 1 : 2);
173                         /*
148                          * Go to the next data cell and assign the
149                          * number of subsequent spans, if applicable.
150                          */
152                         if (dp) {
153                                 spans = dp->spans;
154                                 dp = dp->next;
155                         }
156                 }
157                 break;
158         }
```



```
160         /* Vertical frame at the end of each row. */
162         if (TBL_OPT_BOX & sp->opts->opts || TBL_OPT_DBOX & sp->opts->opts)
188         if (TBL_OPT_BOX & sp->tbl->opts || TBL_OPT_DBOX & sp->tbl->opts)
163                 term_word(tp, TBL_SPAN_HORIZ == sp->pos ||
164                     TBL_SPAN_DHORIZ == sp->pos ? "+" : " |");
165         term_flushln(tp);
167         /*
168          * If we're the last row, clean up after ourselves: clear the
169          * existing table configuration and set it to NULL.
170          */
172         if (TBL_SPAN_LAST & sp->flags) {
173                 if (TBL_OPT_DBOX & sp->opts->opts ||
174                     TBL_OPT_BOX  & sp->opts->opts) {
199                 if (TBL_OPT_DBOX & sp->tbl->opts ||
200                     TBL_OPT_BOX  & sp->tbl->opts)
175                         tbl_hframe(tp, sp, 0);
176                         tp->skipvsp = 1;
177                 }
178                 if (TBL_OPT_DBOX & sp->opts->opts) {
202                 if (TBL_OPT_DBOX & sp->tbl->opts)
179                         tbl_hframe(tp, sp, 1);
180                         tp->skipvsp = 2;
181                 }
182                 assert(tp->tbl.cols);
183                 free(tp->tbl.cols);
184                 tp->tbl.cols = NULL;
185         }
187         tp->flags &= ~TERMP_NONOSPACE;
188         tp->rmargin = rmargin;
189         tp->maxrmargin = maxrmargin;
191 }
193 /*
194  * Horizontal rules extend across the entire table.
195  * Calculate the width by iterating over columns.
196  */
197 static size_t
198 tbl_rulewidth(struct termp *tp, const struct tbl_head *hp)
199 {
200         size_t           width;
202         width = tp->tbl.cols[hp->ident].width;
225         if (TBL_HEAD_DATA == hp->pos) {
204         /* Account for leading blanks. */
205         if (hp->prev)
206                 width += 2 - hp->vert;
208         /* Account for trailing blank. */
227                 if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
209         width++;
229                 /* Account for trailing blanks. */
230                 width++;
231                 if (hp->next &&
232                     TBL_HEAD_VERT  != hp->next->pos &&
233                     TBL_HEAD_DVERT != hp->next->pos)
234                         width++;
235         }
211         return(width);
```

```
 212 }

 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 termp *tp, const struct tbl_span *sp)
 220 {
 221         const struct tbl_head *hp;
 222         char             c;

 224         c = '-';
 225         if (TBL_SPAN_DHORIZ == sp->pos)
 226                 c = '=';

 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         }
 253         for (hp = sp->head; hp; hp = hp->next)
 254                 tbl_char(tp,
 255                     TBL_HEAD_DATA == hp->pos ? c : '+',
 256                     tbl_rulewidth(tp, hp));
 233 }

 235 /*
 236  * Rules above and below the table are always single
 237  * and have an additional plus at the beginning and end.
 238  * For double frames, this function is called twice,
 239  * and the outer one does not have crossings.
 240  */
 241 static void
 242 tbl_hframe(struct termp *tp, const struct tbl_span *sp, int outer)
 243 {
 244         const struct tbl_head *hp;

 246         term_word(tp, "+");
 247         for (hp = sp->head; hp; hp = hp->next) {
 248                 if (hp->prev && hp->vert)
 249                         tbl_char(tp, (outer ? '-' : '+'), hp->vert);
 250                 tbl_char(tp, '-', tbl_rulewidth(tp, hp));
 251         }
 271         for (hp = sp->head; hp; hp = hp->next)
 272                 tbl_char(tp,
 273                     outer || TBL_HEAD_DATA == hp->pos ? '-' : '+',
 274                     tbl_rulewidth(tp, hp));
 252         term_word(tp, "+");
 253         term_flushln(tp);
 254 }

 256 static void
 257 tbl_data(struct termp *tp, const struct tbl_opts *opts,
 280 tbl_data(struct termp *tp, const struct tbl *tbl,
 258                 const struct tbl_dat *dp,
 259                 const struct roffcol *col)
 260 {

 262         if (NULL == dp) {
 263                 tbl_char(tp, ASCII_NBRSP, col->width);
 264                 return;
 265         }
 266         assert(dp->layout);

 268         switch (dp->pos) {
```



```
 269         case (TBL_DATA_NONE):
 270                 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);
 326                 tbl_number(tp, tbl, dp, col);
 304                 break;
 305         case (TBL_CELL_DOWN):
 306                 tbl_char(tp, ASCII_NBRSP, col->width);
 307                 break;
 308         default:
 309                 abort();
 310                 /* NOTREACHED */
 311         }
 312 }

 314 static void
 315 tbl_vrule(struct termp *tp, const struct tbl_head *hp)
 316 {

 318         tbl_char(tp, ASCII_NBRSP, 1);
 319         if (0 < hp->vert)
 320                 tbl_char(tp, '|', hp->vert);
 321         if (2 > hp->vert)
 322                 tbl_char(tp, ASCII_NBRSP, 2 - hp->vert);
 341         switch (hp->pos) {
 342         case (TBL_HEAD_VERT):
 343                 term_word(tp, "|");
 344                 break;
 345         case (TBL_HEAD_DVERT):
 346                 term_word(tp, "||");
 347                 break;
 348         default:
 349                 break;
 350         }
 323 }
```

_____*unchanged_portion_omitted_*

```
static void
tbl_literal(struct termp *tp, const struct tbl_dat *dp,
                const struct roffcol *col)
{
        struct tbl_head         *hp;
        size_t                   width, len, padl, padr;
        int                      spans;
        size_t           len, padl, padr;

        assert(dp->string);
        len = term_strlen(tp, dp->string);

        hp = dp->layout->head->next;
        width = col->width;
        for (spans = dp->spans; spans--; hp = hp->next)
                width += tp->tbl.cols[hp->ident].width + 3;

        padr = width > len ? width - len : 0;
        padr = col->width > len ? col->width - len : 0;
        padl = 0;

        switch (dp->layout->pos) {
        case (TBL_CELL_LONG):
                padl = term_len(tp, 1);
                padr = padr > padl ? padr - padl : 0;
                break;
        case (TBL_CELL_CENTRE):
                if (2 > padr)
                        break;
                padl = padr / 2;
                padr -= padl;
                break;
        case (TBL_CELL_RIGHT):
                padl = padr;
                padr = 0;
                break;
        default:
                break;
        }

        tbl_char(tp, ASCII_NBRSP, padl);
        term_word(tp, dp->string);
        tbl_char(tp, ASCII_NBRSP, padr);
}

static void
tbl_number(struct termp *tp, const struct tbl_opts *opts,
tbl_number(struct termp *tp, const struct tbl *tbl,
                const struct tbl_dat *dp,
                const struct roffcol *col)
{
        char            *cp;
        char             buf[2];
        size_t           sz, psz, ssz, d, padl;
        int              i;

        /*
         * See calc_data_number().  Left-pad by taking the offset of our
         * and the maximum decimal; right-pad by the remaining amount.
         */

        assert(dp->string);

        sz = term_strlen(tp, dp->string);
```



```
        buf[0] = opts->decimal;
        buf[0] = tbl->decimal;
        buf[1] = '\0';

        psz = term_strlen(tp, buf);

        if (NULL != (cp = strrchr(dp->string, opts->decimal))) {
        if (NULL != (cp = strrchr(dp->string, tbl->decimal))) {
                buf[1] = '\0';
                for (ssz = 0, i = 0; cp != &dp->string[i]; i++) {
                        buf[0] = dp->string[i];
                        ssz += term_strlen(tp, buf);
                }
                d = ssz + psz;
        } else
                d = sz + psz;

        padl = col->decimal - d;

        tbl_char(tp, ASCII_NBRSP, padl);
        term_word(tp, dp->string);
        if (col->width > sz + padl)
                tbl_char(tp, ASCII_NBRSP, col->width - sz - padl);
}
______unchanged_portion_omitted_
```

```
**********
   16799 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/term.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********
```

```
  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

 22 #include <sys/types.h>

 24 #include <assert.h>
 25 #include <ctype.h>
 26 #include <stdint.h>
 27 #include <stdio.h>
 28 #include <stdlib.h>
 29 #include <string.h>

 31 #include "mandoc.h"
 32 #include "out.h"
 33 #include "term.h"
 34 #include "main.h"

 36 static  size_t           cond_width(const struct termp *, int, int *);
 37 static  void             adjbuf(struct termp *p, size_t);
 36 static  void             adjbuf(struct termp *p, int);
 38 static  void             bufferc(struct termp *, char);
 39 static  void             encode(struct termp *, const char *, size_t);
 40 static  void             encode1(struct termp *, int);

 42 void
 43 term_free(struct termp *p)
 44 {

 46         if (p->buf)
 47                 free(p->buf);
 48         if (p->symtab)
 49                 mchars_free(p->symtab);

 51         free(p);
 52 }
```

*unchanged_portion_omitted_*

```
 74 /*
 75  * Flush a line of text.  A "line" is loosely defined as being something
```



```
 76  * that should be followed by a newline, regardless of whether it's
 77  * 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  *
 86  *  - TERMP_TWOSPACE: make sure there is room for at least two space
 87  *    characters of padding.  Otherwise, rather break the line.
 88  *
 89  *  - TERMP_DANGLE: don't newline when TERMP_NOBREAK is specified and
 90  *    the line is overrun, and don't pad-right if it's underrun.
 91  *
 92  *  - TERMP_HANG: like TERMP_DANGLE, but doesn't newline when
 93  *    overrunning, instead save the position and continue at that point
 94  *    when the next invocation.
 95  *
 96  *  In-line line breaking:
 97  *
 98  *  If TERMP_NOBREAK is specified and the line overruns the right
 99  *  margin, it will break and pad-right to the right margin after
100  *  writing.  If maxrmargin is violated, it will break and continue
101  *  writing from the right-margin, which will lead to the above scenario
102  *  upon exit.  Otherwise, the line will break at the right margin.
103  */
104 void
105 term_flushln(struct termp *p)
106 {
107         size_t           i;     /* current input position in p->buf */
108         int              ntab;  /* number of tabs to prepend */
107         int              i;     /* current input position in p->buf */
109         size_t           vis;   /* current visual position on output */
110         size_t           vbl;   /* number of blanks to prepend to output */
111         size_t           vend;  /* end of word visual position on output */
112         size_t           bp;    /* visual right border position */
113         size_t           dv;    /* temporary for visual pos calculations */
114         size_t           j;     /* temporary loop index for p->buf */
115         size_t           jhy;   /* last hyph before overflow w/r/t j */
113         int              j;     /* temporary loop index for p->buf */
114         int              jhy;   /* last hyph before overflow w/r/t j */
116         size_t           maxvis; /* output position of visible boundary */
117         size_t           mmax; /* used in calculating bp */

119         /*
120          * First, establish the maximum columns of "visible" content.
121          * This is usually the difference between the right-margin and
122          * an indentation, but can be, for tagged lists or columns, a
123          * small set of values.
124          *
125          * The following unsigned-signed subtractions look strange,
126          * but they are actually correct.  If the int p->overstep
127          * is negative, it gets sign extended.  Subtracting that
128          * very large size_t effectively adds a small number to dv.
129          */
130         assert  (p->rmargin >= p->offset);
131         dv     = p->rmargin - p->offset;
132         maxvis = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;
133         dv     = p->maxrmargin - p->offset;
134         mmax   = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;
```

```
136         bp = TERMP_NOBREAK & p->flags ? mmax : maxvis;
137
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;
143
144         vis = vend = 0;
145         i = 0;
146
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                 }
160
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                  */
167
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
173                         /* Back over the the last printed character. */
174                         if (8 == p->buf[j]) {
175                                 assert(j);
176                                 vend -= (*p->width)(p, p->buf[j - 1]);
177                                 continue;
178                         }
179
180                         /* Regular word. */
181                         /* Break at the hyphen point if we overrun. */
182                         if (vend > vis && vend < bp &&
183                                         ASCII_HYPH == p->buf[j])
184                                 jhy = j;
185
186                         vend += (*p->width)(p, p->buf[j]);
187                 }
188
189                 /*
190                  * Find out whether we would exceed the right margin.
191                  * If so, break to the next line.
192                  */
193                 if (vend > bp && 0 == jhy && vis > 0) {
194                         vend -= vis;
195                         (*p->endline)(p);
196                         p->viscol = 0;
197                         if (TERMP_NOBREAK & p->flags) {
198                                 vbl = p->rmargin;
199                                 vend += p->rmargin - p->offset;
200                         } else
```



```
202                         /* use pending tabs on the new line */
194                         /* Remove the p->overstep width. */
204                         if (0 < ntab)
205                                 vbl += ntab * p->tabwidth;
206
207                         /*
208                          * Remove the p->overstep width.
209                          * Again, if p->overstep is negative,
210                          * sign extension does the right thing.
211                          */
212
213                         bp += (size_t)p->overstep;
214                         p->overstep = 0;
215                 }
216
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, ' ');
210                                 dv = (size_t)(i - j) * (*p->width)(p, ' ');
228                                 vbl += dv;
229                                 vend += dv;
230                                 break;
231                         }
232                         if (ASCII_NBRSP == p->buf[i]) {
233                                 vbl += (*p->width)(p, ' ');
234                                 continue;
235                         }
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                         }
247
248                         if (ASCII_HYPH == p->buf[i]) {
249                                 (*p->letter)(p, '-');
250                                 p->viscol += (*p->width)(p, '-');
251                                 continue;
252                         }
253
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         }
262
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, ' '));
263                 /* We need one blank after the tag. */
264                 p->overstep = (int)(vis - maxvis + (*p->width)(p, ' '));

283                 /*
267                  * Behave exactly the same way as groff:
284                  * If we have overstepped the margin, temporarily move
285                  * it to the right and flag the rest of the line to be
286                  * shorter.
287                  * If there is a request to keep the columns together,
288                  * allow negative overstep when the column is not full.
271                  * If we landed right at the margin, be happy.
272                  * If we are one step before the margin, temporarily
273                  * move it one step LEFT and flag the rest of the line
274                  * to be longer.
289                  */
290                 if (p->trailspace && p->overstep < 0)
276                 if (p->overstep < -1)
291                         p->overstep = 0;
292                 return;

294         } else if (TERMP_DANGLE & p->flags)
295                 return;

297         /* If the column was overrun, break the line. */
298         if (maxvis < vis + p->trailspace * (*p->width)(p, ' ')) {
284         if (maxvis <= vis +
285             ((TERMP_TWOSPACE & p->flags) ? (*p->width)(p, ' ') : 0)) {
299                 (*p->endline)(p);
300                 p->viscol = 0;
301         }
302 }
```

_____*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 termp *p)
328 {

330         term_newln(p);
331         p->viscol = 0;
332         if (0 < p->skipvsp)
333                 p->skipvsp--;
334         else
```



```
335                 (*p->endline)(p);
336 }
```

_____*unchanged_portion_omitted_*

```
384 void
385 term_fontpopq(struct termp *p, const void *key)
386 {

388         while (p->fonti >= 0 && key < (void *)(p->fontq + p->fonti))
372         while (p->fonti >= 0 && key != &p->fontq[p->fonti])
389                 p->fonti--;
390         assert(p->fonti >= 0);
391 }
```

_____*unchanged_portion_omitted_*

```
402 /*
403  * Handle pwords, partial words, which may be either a single word or a
404  * phrase that cannot be broken down (such as a literal string).  This
405  * handles word styling.
406  */
407 void
408 term_word(struct termp *p, const char *word)
409 {
410         const char       nbrsp[2] = { ASCII_NBRSP, 0 };
411         const char      *seq, *cp;
412         char             c;
413         int              sz, uc;
414         size_t           ssz;
415         enum mandoc_esc  esc;

417         if ( ! (TERMP_NOSPACE & p->flags)) {
418                 if ( ! (TERMP_KEEP & p->flags)) {
402                         if (TERMP_PREKEEP & p->flags)
403                                 p->flags |= TERMP_KEEP;
419                         bufferc(p, ' ');
420                         if (TERMP_SENTENCE & p->flags)
421                                 bufferc(p, ' ');
422                 } else
423                         bufferc(p, ASCII_NBRSP);
424         }
425         if (TERMP_PREKEEP & p->flags)
426                 p->flags |= TERMP_KEEP;

428         if ( ! (p->flags & TERMP_NONOSPACE))
429                 p->flags &= ~TERMP_NOSPACE;
430         else
431                 p->flags |= TERMP_NOSPACE;

433         p->flags &= ~TERMP_SENTENCE;
416         p->flags &= ~(TERMP_SENTENCE | TERMP_IGNDELIM);

435         while ('\0' != *word) {
436                 if ('\\' != *word) {
437                         if (TERMP_SKIPCHAR & p->flags) {
438                                 p->flags &= ~TERMP_SKIPCHAR;
439                                 word++;
440                                 continue;
441                         }
442                         if (TERMP_NBRWORD & p->flags) {
443                                 if (' ' == *word) {
444                                         encode(p, nbrsp, 1);
445                                         word++;
446                                         continue;
447                                 }
448                                 ssz = strcspn(word, "\\ ");
```

```
449                         } else
450                                 ssz = strcspn(word, "\\");
419                 if ((ssz = strcspn(word, "\\")) > 0)
451                         encode(p, word, ssz);
452                         word += (int)ssz;
423                 if ('\\' != *word)
453                         continue;
454                 }

456                 word++;
457                 esc = mandoc_escape(&word, &seq, &sz);
458                 if (ESCAPE_ERROR == esc)
459                         break;

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->symtab, seq, sz);
471                                 if (uc <= 0)
472                                         break;
473                                 encode1(p, uc);
474                                 continue;
475                         default:
476                                 break;
477                         }

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->symtab, 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)
480                         if ('\0' == *word)
516                                 p->flags |= TERMP_NOSPACE;
517                         break;
518                 case (ESCAPE_SKIPCHAR):
519                         p->flags |= TERMP_SKIPCHAR;
520                         break;
521                 default:
522                         break;
523                 }
524         }
525         p->flags &= ~TERMP_NBRWORD;
526 }

528 static void
529 adjbuf(struct termp *p, size_t sz)
490 adjbuf(struct termp *p, int sz)
530 {

532         if (0 == p->maxcols)
533                 p->maxcols = 1024;
534         while (sz >= p->maxcols)
535                 p->maxcols <<= 2;

537         p->buf = mandoc_realloc(p->buf, sizeof(int) * p->maxcols);
498         p->buf = mandoc_realloc
499                 (p->buf, sizeof(int) * (size_t)p->maxcols);
538 }
_____unchanged_portion_omitted_

550 /*
551  * See encode().
552  * Do this for a single (probably unicode) value.
553  * Does not check for non-decorated glyphs.
554  */
555 static void
556 encode1(struct termp *p, int c)
557 {
558         enum termfont     f;

560         if (TERMP_SKIPCHAR & p->flags) {
561                 p->flags &= ~TERMP_SKIPCHAR;
562                 return;
563         }
522         if (p->col + 4 >= p->maxcols)
523                 adjbuf(p, p->col + 4);

565         if (p->col + 6 >= p->maxcols)
566                 adjbuf(p, p->col + 6);

568         f = term_fonttop(p);

570         if (TERMFONT_UNDER == f || TERMFONT_BI == f) {
527         if (TERMFONT_NONE == f) {
528                 p->buf[p->col++] = c;
529                 return;
530         } else if (TERMFONT_UNDER == f) {
571                 p->buf[p->col++] = '_';
572                 p->buf[p->col++] = 8;
573         }
574         if (TERMFONT_BOLD == f || TERMFONT_BI == f) {
575                 if (ASCII_HYPH == c)
576                         p->buf[p->col++] = '-';
```

```
577                 else
532         } 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 termp *p, const char *word, size_t sz)
586 {
587         size_t           i;
542         enum termfont    f;
543         int              i, len;

589         if (TERMP_SKIPCHAR & p->flags) {
590                 p->flags &= ~TERMP_SKIPCHAR;
591                 return;
592         }
545         /* LINTED */
546         len = sz;

594         /*
595          * Encode and buffer a string of characters.  If the current
596          * font mode is unset, buffer directly, else encode then buffer
597          * character by character.
598          */

600         if (TERMFONT_NONE == term_fonttop(p)) {
601                 if (p->col + sz >= p->maxcols)
602                         adjbuf(p, p->col + sz);
603                 for (i = 0; i < sz; i++)
554         if (TERMFONT_NONE == (f = term_fonttop(p))) {
555                 if (p->col + len >= p->maxcols)
556                         adjbuf(p, p->col + len);
557                 for (i = 0; i < len; i++)
604                         p->buf[p->col++] = word[i];
605                 return;
606         }

608         /* Pre-buffer, assuming worst-case. */

610         if (p->col + 1 + (sz * 5) >= p->maxcols)
611                 adjbuf(p, p->col + 1 + (sz * 5));
564         if (p->col + 1 + (len * 3) >= p->maxcols)
565                 adjbuf(p, p->col + 1 + (len * 3));

613         for (i = 0; i < sz; i++) {
614                 if (ASCII_HYPH == word[i] ||
615                     isgraph((unsigned char)word[i]))
616                         encode1(p, word[i]);
567         for (i = 0; i < len; i++) {
568                 if (ASCII_HYPH != word[i] &&
569                     ! isgraph((unsigned char)word[i])) {
570                         p->buf[p->col++] = word[i];
571                         continue;
572                 }

574                 if (TERMFONT_UNDER == f)
575                         p->buf[p->col++] = '_';
576                 else if (ASCII_HYPH == word[i])
577                         p->buf[p->col++] = '-';
617                 else
618                         p->buf[p->col++] = word[i];
```



```
581                 p->buf[p->col++] = 8;
582                 p->buf[p->col++] = word[i];
619         }
620 }
_____unchanged_portion_omitted_

629 static size_t
630 cond_width(const struct termp *p, int c, int *skip)
631 {

633         if (*skip) {
634                 (*skip) = 0;
635                 return(0);
636         } else
637                 return((*p->width)(p, c));
638 }

640 size_t
641 term_strlen(const struct termp *p, const char *cp)
642 {
643         size_t           sz, rsz, i;
644         int              ssz, skip, c;
598         int              ssz, c;
645         const char      *seq, *rhs;
646         enum mandoc_esc  esc;
647         static const char rej[] = { '\\', ASCII_HYPH, ASCII_NBRSP, '\0' };

649         /*
650          * Account for escaped sequences within string length
651          * calculations.  This follows the logic in term_word() as we
652          * must calculate the width of produced strings.
653          */

655         sz = 0;
656         skip = 0;
657         while ('\0' != *cp) {
658                 rsz = strcspn(cp, rej);
659                 for (i = 0; i < rsz; i++)
660                         sz += cond_width(p, *cp++, &skip);
613                         sz += (*p->width)(p, *cp++);

662                 c = 0;
663                 switch (*cp) {
664                 case ('\\'):
665                         cp++;
666                         esc = mandoc_escape(&cp, &seq, &ssz);
667                         if (ESCAPE_ERROR == esc)
668                                 return(sz);

670                         if (TERMENC_ASCII != p->enc)
671                                 switch (esc) {
672                                 case (ESCAPE_UNICODE):
673                                         c = mchars_num2uc
674                                                 (seq + 1, ssz - 1);
675                                         if ('\0' == c)
676                                                 break;
677                                         sz += cond_width(p, c, &skip);
630                                         sz += (*p->width)(p, c);
678                                         continue;
679                                 case (ESCAPE_SPECIAL):
680                                         c = mchars_spec2cp
681                                                 (p->symtab, seq, ssz);
682                                         if (c <= 0)
683                                                 break;
684                                         sz += cond_width(p, c, &skip);
637                                         sz += (*p->width)(p, c);
```

```
685                                     continue;
686                             default:
687                                     break;
688                             }

690                     rhs = NULL;

692                     switch (esc) {
693                     case (ESCAPE_UNICODE):
694                             sz += cond_width(p, '?', &skip);
647                             sz += (*p->width)(p, '?');
695                             break;
696                     case (ESCAPE_NUMBERED):
697                             c = mchars_num2char(seq, ssz);
698                             if ('\0' != c)
699                                     sz += cond_width(p, c, &skip);
652                                     sz += (*p->width)(p, c);
700                             break;
701                     case (ESCAPE_SPECIAL):
702                             rhs = mchars_spec2str
703                                     (p->symtab, seq, ssz, &rsz);

705                             if (ssz != 1 || rhs)
706                                     break;

708                             rhs = seq;
709                             rsz = ssz;
710                             break;
711                     case (ESCAPE_SKIPCHAR):
712                             skip = 1;
713                             break;
714                     default:
715                             break;
716                     }

718                     if (NULL == rhs)
719                             break;

721                     if (skip) {
722                             skip = 0;
723                             break;
724                     }

726                     for (i = 0; i < rsz; i++)
727                             sz += (*p->width)(p, *rhs++);
728                     break;
729             case (ASCII_NBRSP):
730                     sz += cond_width(p, ' ', &skip);
675                     sz += (*p->width)(p, ' ');
731                     cp++;
732                     break;
733             case (ASCII_HYPH):
734                     sz += cond_width(p, '-', &skip);
679                     sz += (*p->width)(p, '-');
735                     cp++;
736                     break;
737             default:
738                     break;
739             }
740         }

742         return(sz);
743 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    4745 Wed Jul 30 20:55:12 2014
new/usr/src/cmd/mandoc/term.h
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
  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. */
 60         int               maxcols;      /* Max size of buf. */
 63         size_t            offset;       /* Margin offest. */
 64         size_t            tabwidth;     /* Distance of tab positions. */
 65         size_t            col;          /* Bytes in buf. */
 63         int               col;          /* Bytes in buf. */
 66         size_t            viscol;       /* Chars on current line. */
 67         size_t            trailspace;   /* See termp_flushln(). */
 68         int               overstep;     /* See termp_flushln(). */
 69         int               skipvsp;      /* Vertical space to skip. */
```



```
 70         int               flags;
 71 #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(). */
 69 #define TERMP_NOBREAK    (1 << 4)       /* See term_flushln(). */
 70 #define TERMP_IGNDELIM   (1 << 6)       /* Delims like regulars. */
 71 #define TERMP_NONOSPACE  (1 << 7)       /* No space (no autounset). */
 72 #define TERMP_DANGLE     (1 << 8)       /* See term_flushln(). */
 73 #define TERMP_HANG       (1 << 9)       /* See term_flushln(). */
 74 #define TERMP_TWOSPACE   (1 << 10)      /* See term_flushln(). */
 81 #define TERMP_NOSPLIT    (1 << 11)      /* See termp_an_pre/post(). */
 82 #define TERMP_SPLIT      (1 << 12)      /* See termp_an_pre/post(). */
 83 #define TERMP_ANPREC     (1 << 13)      /* See termp_an_pre(). */
 78 #define TERMP_KEEP       (1 << 14)      /* Keep words together. */
 79 #define TERMP_PREKEEP    (1 << 15)      /* ...starting with the next one. */
 84         int              *buf;          /* Output buffer. */
 85         enum termenc      enc;          /* Type of encoding. */
 86         struct mchars    *symtab;       /* Encoded-symbol table. */
 87         enum termfont     fontl;        /* Last font set. */
 88         enum termfont     fontq[10];    /* Symmetric fonts. */
 89         int               fonti;        /* Index of font stack. */
 90         term_margin       headf;        /* invoked to print head */
 91         term_margin       footf;        /* invoked to print foot */
 92         void             (*letter)(struct termp *, int);
 93         void             (*begin)(struct termp *);
 94         void             (*end)(struct termp *);
 95         void             (*endline)(struct termp *);
 96         void             (*advance)(struct termp *, size_t);
 97         size_t           (*width)(const struct termp *, int);
 98         double           (*hspan)(const struct termp *,
 99                                   const struct roffsu *);
100         const void        *argf;         /* arg for headf/footf */
101         struct termp_ps   *ps;
102 };
______unchanged_portion_omitted_
```

```
**********************************************************
    5394 Wed Jul 30 20:55:13 2014
new/usr/src/cmd/mandoc/term_ascii.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
/*      $Id: term_ascii.c,v 1.21 2013/06/01 14:27:20 schwarze Exp $ */
/*      $Id: term_ascii.c,v 1.20 2011/12/04 23:10:52 schwarze Exp $ */
/*
 * Copyright (c) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

#include <assert.h>
#ifdef USE_WCHAR
# include <locale.h>
#endif
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#ifdef USE_WCHAR
# include <wchar.h>
#endif

#include "mandoc.h"
#include "out.h"
#include "term.h"
#include "main.h"

/*
 * Sadly, this doesn't seem to be defined on systems even when they
 * support it.  For the time being, remove it and let those compiling
 * the software decide for themselves what to use.
 */
#if 0
#if ! defined(__STDC_ISO_10646__)
# undef USE_WCHAR
#endif
#endif

static  struct termp     *ascii_init(enum termenc, char *);
static  double            ascii_hspan(const struct termp *,
                                const struct roffsu *);
static  size_t            ascii_width(const struct termp *, int);
static  void              ascii_advance(struct termp *, size_t);
static  void              ascii_begin(struct termp *);
static  void              ascii_end(struct termp *);
static  void              ascii_endline(struct termp *);
```



```
static  void              ascii_letter(struct termp *, int);

#ifdef  USE_WCHAR
static  void              locale_advance(struct termp *, size_t);
static  void              locale_endline(struct termp *);
static  void              locale_letter(struct termp *, int);
static  size_t            locale_width(const struct termp *, int);
#endif

static struct termp *
ascii_init(enum termenc enc, char *outopts)
{
        const char      *toks[4];
        char            *v;
        struct termp    *p;

        p = mandoc_calloc(1, sizeof(struct termp));
        p->enc = enc;

        p->tabwidth = 5;
        p->defrmargin = 78;

        p->begin = ascii_begin;
        p->end = ascii_end;
        p->hspan = ascii_hspan;
        p->type = TERMTYPE_CHAR;

        p->enc = TERMENC_ASCII;
        p->advance = ascii_advance;
        p->endline = ascii_endline;
        p->letter = ascii_letter;
        p->width = ascii_width;

#ifdef  USE_WCHAR
        if (TERMENC_ASCII != enc) {
                v = TERMENC_LOCALE == enc ?
                        setlocale(LC_ALL, "") :
                        setlocale(LC_CTYPE, "en_US.UTF-8");
                        setlocale(LC_CTYPE, "UTF-8");
                if (NULL != v && MB_CUR_MAX > 1) {
                        p->enc = enc;
                        p->advance = locale_advance;
                        p->endline = locale_endline;
                        p->letter = locale_letter;
                        p->width = locale_width;
                }
        }
#endif

        toks[0] = "indent";
        toks[1] = "width";
        toks[2] = "mdoc";
        toks[3] = NULL;

        while (outopts && *outopts)
                switch (getsubopt(&outopts, UNCONST(toks), &v)) {
                case (0):
                        p->defindent = (size_t)atoi(v);
                        break;
                case (1):
                        p->defrmargin = (size_t)atoi(v);
                        break;
                case (2):
                        /*
                         * Temporary, undocumented mode
                         * to imitate mdoc(7) output style.
```

```
123                          */
124                         p->mdocstyle = 1;
125                         p->defindent = 5;
126                         break;
127                 default:
128                         break;
129                 }

131         /* Enforce a lower boundary. */
132         if (p->defrmargin < 58)
133                 p->defrmargin = 58;

135         return(p);
136 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6575 Wed Jul 30 20:55:13 2014
new/usr/src/cmd/mandoc/tree.c
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
  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

 22 #include <assert.h>
 23 #include <limits.h>
 24 #include <stdio.h>
 25 #include <stdlib.h>
 26 #include <time.h>

 28 #include "mandoc.h"
 29 #include "mdoc.h"
 30 #include "man.h"
 31 #include "main.h"

 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);


 39 /* ARGSUSED */
 40 void
 41 tree_mdoc(void *arg, const struct mdoc *mdoc)
 42 {

 44         print_mdoc(mdoc_node(mdoc), 0);
 45 }
______unchanged_portion_omitted_


 57 static void
 58 print_mdoc(const struct mdoc_node *n, int indent)
 59 {
 60         const char       *p, *t;
 61         int               i, j;
 62         size_t            argc;
 61         size_t           argc, sz;
 62         char            **params;
 63         struct mdoc_argv *argv;
```



```
 65         argv = NULL;
 66         argc = 0;
 66         argc = sz = 0;
 67         params = NULL;
 67         t = p = NULL;

 69         switch (n->type) {
 70         case (MDOC_ROOT):
 71                 t = "root";
 72                 break;
 73         case (MDOC_BLOCK):
 74                 t = "block";
 75                 break;
 76         case (MDOC_HEAD):
 77                 t = "block-head";
 78                 break;
 79         case (MDOC_BODY):
 80                 if (n->end)
 81                         t = "body-end";
 82                 else
 83                         t = "block-body";
 84                 break;
 85         case (MDOC_TAIL):
 86                 t = "block-tail";
 87                 break;
 88         case (MDOC_ELEM):
 89                 t = "elem";
 90                 break;
 91         case (MDOC_TEXT):
 92                 t = "text";
 93                 break;
 94         case (MDOC_TBL):
 95                 /* FALLTHROUGH */
 96         case (MDOC_EQN):
 97                 break;
 98         default:
 99                 abort();
100                 /* NOTREACHED */
101         }

103         switch (n->type) {
104         case (MDOC_TEXT):
105                 p = n->string;
106                 break;
107         case (MDOC_BODY):
108                 p = mdoc_macronames[n->tok];
109                 break;
110         case (MDOC_HEAD):
111                 p = mdoc_macronames[n->tok];
112                 break;
113         case (MDOC_TAIL):
114                 p = mdoc_macronames[n->tok];
115                 break;
116         case (MDOC_ELEM):
117                 p = mdoc_macronames[n->tok];
118                 if (n->args) {
119                         argv = n->args->argv;
120                         argc = n->args->argc;
121                 }
122                 break;
123         case (MDOC_BLOCK):
124                 p = mdoc_macronames[n->tok];
125                 if (n->args) {
126                         argv = n->args->argv;
127                         argc = n->args->argc;
128                 }
```

```
129                 break;
130         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');
165                 for (i = 0; i < (int)sz; i++)
166                         printf(" [%s]", params[i]);

168                 printf(" %d:%d\n", n->line, n->pos);
171         }

173         if (n->child)
174                 print_mdoc(n->child, indent + 1);
175         if (n->next)
176                 print_mdoc(n->next, indent);
177 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1605 Wed Jul 30 20:55:13 2014
new/usr/src/man/Makefile.man
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet
# at http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011, Richard Lowe
# Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#

MANDOC=         $(ONBLD_TOOLS)/bin/${MACH}/mandoc
ROOTMAN=        $(ROOT)/usr/share/man
ROOTHASMAN=     $(ROOT)/usr/has/man
FILEMODE=       0444

# The manual section being built, client Makefiles must set this to, for e.g.
# "3perl", with case matching that of the section name as installed.
#
# MANSECT=

MANCHECKS=      $(MANFILES:%=%.check)
ROOTMANFILES=   $(MANFILES:%=$(ROOTMAN)/man$(MANSECT)/%)
ROOTMANLINKS=   $(MANLINKS:%=$(ROOTMAN)/man$(MANSECT)/%)

$(ROOTMAN)/man$(MANSECT)/% $(ROOTHASMAN)/man$(MANSECT)/%: %
                $(INS.file)

#
# Note that new mandoc adds some checks for lots of extra whitespace.
# We don't want to check our legacy pages for that.  There are thousands
# and thousands of them in our man pages.  Please still check them
# manually when editing (git pbchk will do so for you.)
#
$(MANCHECKS):
                @$(EGREP) -q "^.TH" $(@:%.check=%) || \
                ( $(ECHO) "checking $(@:%.check=%)"; \
                $(MANDOC) -Tlint $(@:%.check=%) )
                @$(ECHO) "checking $(@:%.check=%)"; \
                $(MANDOC) -Tlint $(@:%.check=%)

$(MANLINKS):
                $(RM) $@; $(SYMLINK) $(LINKSRC) $@

$(ROOTMANLINKS): $(MANLINKS)
                $(RM) $@; $(CP) -RP $(@F) $(@D)

all:

check:          $(MANCHECKS)

clean:

clobber:
```



```
                $(RM) $(MANLINKS)

.PARALLEL:

FRC:
```

```
**********************************************************
    9377 Wed Jul 30 20:55:13 2014
new/usr/src/man/man1/man.1
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
.\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
.\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
.\" Copyright (c) 1980 Regents of the University of California.
.\" The Berkeley software License Agreement specifies the terms and conditions
.\" for redistribution.
.Dd Jul 18, 2014
.Dt MAN 1
.Os
.Sh NAME
.Nm man
.Nd find and display reference manual pages
.Sh SYNOPSIS
.Nm
.Op Fl
.Op Fl adFlrt
.Op Fl T Ar macro-package
.Op Fl M Ar path
.Op Fl s Ar section
.Ar name ...
.Nm
.Op Fl M Ar path
.Op Fl s Ar section
.Fl k
.Ar keyword
.Ar ...
.Nm
.Op Fl M Ar path
.Op Fl s Ar section
.Fl f
.Ar
.Nm
.Op Fl M Ar path
.Fl w
.Sh DESCRIPTION
The
.Nm
command displays information from the reference manuals. It
displays complete manual pages that you select by
.Ar name ,
or one-line summaries selected either by
.Ar keyword
.Pq Fl k ,
or by the name of an associated file
.Pq Fl f .
If no manual page is located,
.Nm
prints an error message.
.Ss "Source Format"
Reference Manual pages are marked up with either
.Xr man 5 ,
or
.Xr mdoc 5
language tags. The
.Nm
command recognizes the type of markup and
processes the file accordingly.
.
.Ss "Location of Manual Pages"
.
```



```
The online Reference Manual page directories are conventionally located in
.Pa /usr/share/man .
Each directory corresponds to a
section of the manual. Since these directories are optionally installed, they
might not reside on your host. You might have to mount
.Pa /usr/share/man
from a host on which they do reside.
The
.Nm
command reformats a page whenever it is requested.
.Pp
If the standard output is not a terminal, or if the
.Fl
flag is given,
.Nm
pipes its output through
.Xr cat 1 .
Otherwise,
.Nm
pipes its output through a pager such as
.Xr more 1
to handle paging and underlining on the screen.
.Sh OPTIONS
The following options are supported:
.Bl -tag -width indent
.It Fl a
Shows all manual pages matching
.Ar name
within the
.Ev MANPATH
search path. Manual pages are displayed in the order found.
.It Fl d
Debugs. Displays what a section-specifier evaluates to, method used for
searching, and paths searched by
.Nm .
.It Fl f Ar file ...
Attempts to locate manual pages related to any of the given
.Ar file
names. It strips the leading path name components from each
.Ar file ,
and then prints one-line summaries containing the resulting basename or names.
This option also uses the
.Pa whatis
database.
.It Fl F
This option is present for backwards compatibility and is documented
here for reference only.  It performs no function.
.It Fl k Ar keyword ...
Prints out one-line summaries from the
.Pa whatis
database (table of contents) that contain any of the given
.Ar keyword .
The
.Pa whatis
database is created using the
.Fl w
option.
.It Fl l
Lists all manual pages found matching
.Ar name
within the search path.
.It Fl M Ar path
Specifies an alternate search path for manual pages. The
.Ar path
is a colon-separated list of directories that contain manual page directory
subtrees. For example, if
```

```
.Ar path
is
.Pa /usr/share/man:/usr/local/man ,
.Nm
searches for
.Ar name
in the standard location, and then
.Pa /usr/local/man .
When used with the
.Fl k ,
.Fl f ,
or
.Fl w
options, the
.Fl M
option must appear first. Each directory in the
.Ar path
is assumed to contain subdirectories of the form
.Pa man* ,
one for each section. This option overrides the
.Ev MANPATH
environment variable.
.It Fl r
Reformats the manual page, checking for formatting errors, but does not
display it.
.It Fl s Ar section
Specifies sections of the manual for
.Nm
to search. The directories searched for
.Ar name
are limited to those specified by
.Ar section .
.Ar section
can be a numerical digit, perhaps followed by one or more letters
to match the desired section of the manual, for example,
.Li "3libucb".
Also,
.Ar section
can be a word, for example,
.Li local ,
.Li new ,
.Li old ,
.Li public .
.Ar section
can also be a letter. To specify multiple sections,
separate each section with a comma. This option overrides the
.Ev MANPATH
environment variable and the
.Pa man.cf
file. See
.Sx Search Path
below for an explanation of how
.Nm
conducts its search.
.It Fl t
Arranges for the specified manual pages to be sent to the default
printer as PostScript.
.It Fl T Ar macro-package
This option is present for backwards compatibility and is documented
here for reference only.  It performs no function.
.It Fl w
Updates the
.Nm whatis
database.
.El
.Sh OPERANDS
```



```
The following operand is supported:
.Bl -tag -width indent
.It Ar name
The name of a standard utility or a keyword.
.El
.Sh USAGE
The usage of
.Nm
is described below:
.
.Ss "Manual Page Sections"
.
Entries in the reference manuals are organized into
.Em sections .
A section
name consists of a major section name, typically a single digit, optionally
followed by a subsection name, typically one or more letters. An unadorned
major section name, for example,
.Qq 9 ,
does not act as an abbreviation for
the subsections of that name, such as
.Qq 9e ,
.Qq 9f ,
or
.Qq 9s .
That is, each subsection must be searched separately by
.Nm
.Fl s .
Each section contains descriptions apropos to a particular reference category,
with subsections refining these distinctions. See the
.Em intro
manual pages for an explanation of the classification used in this release.
.
.Ss "Search Path"
.
Before searching for a given
.Ar name ,
.Nm
constructs a list of candidate directories and sections.
It searches for
.Ar name
in the directories specified by the
.Ev MANPATH
environment variable.
.Lp
In the absence of
.Ev MANPATH ,
.Nm
constructs its search path based upon the
.Ev PATH
environment variable, primarily by substituting
.Li man
for the last component of the
.Ev PATH
element. Special provisions are added
to account for unique characteristics of directories such as
.Pa /sbin ,
.Pa /usr/ucb ,
.Pa /usr/xpg4/bin ,
and others. If the file argument contains
a
.Qq /
character, the
.Em dirname
portion of the argument is used in place of
.Ev PATH
```

```
258 elements to construct the search path.
259 .Lp
260 Within the manual page directories,
261 .Nm
262 confines its search to the
263 sections specified in the following order:
264 .Bl -bullet
265 .It
266 .Ar sections
267 specified on the command line with the
268 .Fl s
269 option
270 .It
271 .Ar sections
272 embedded in the
273 .Ev MANPATH
274 environment variable
275 .It
276 .Ar sections
277 specified in the
278 .Pa man.cf
279 file for each directory specified in the
280 .Ev MANPATH
281 environment variable
282 .El
283 If none of the above exist,
284 .Nm
285 searches each directory in the manual
286 page path, and displays the first matching manual page found.
287 .Lp
288 The
289 .Pa man.cf
290 file has the following format:
291 .Lp
292 .Dl Pf MANSECTS= Ar section , Ns Op Ar section...
293 .Lp
294 Lines beginning with
295 .Sq Li #
296 and blank lines are considered comments, and are
297 ignored. Each directory specified in
298 .Ev MANPATH
299 can contain a manual page
300 configuration file, specifying the default search order for that directory.
301 .Sh "Referring to Other Manual Pages"
302 If the first line of the manual page is a reference to another manual
303 page entry fitting the pattern:
304 .Lp
305 .Dl \&.so man*/\fIsourcefile\fR
306 .Lp
307 .Nm
308 processes the indicated file in place of the current one. The
309 reference must be expressed as a path name relative to the root of the manual
310 page directory subtree.
311 .Lp
312 When the second or any subsequent line starts with \fB\&.so\fR, \fBman\fR
313 ignores it; \fBtroff\fR(1) or \fBnroff\fR(1) processes the request in the usual
314 manner.
315 .Sh ENVIRONMENT VARIABLES
316 See
317 .Xr environ 5
318 for descriptions of the following environment variables
319 that affect the execution of
320 .Nm man :
321 .Ev LANG ,
322 .Ev LC_ALL ,
323 .Ev LC_CTYPE ,
```



```
324 .Ev LC_MESSAGES ,
325 and
326 .Ev NLSPATH .
327 .Bl -tag -width indent
328 .It Ev MANPATH
329 A colon-separated list of directories; each directory can be followed by a
330 comma-separated list of sections. If set, its value overrides
331 \fB/usr/share/man\fR as the default directory search path, and the \fBman.cf\fR
332 file as the default section search path. The \fB-M\fR and \fB-s\fR flags, in
333 turn, override these values.)
334 .It Ev PAGER
335 A program to use for interactively delivering
336 output to the screen. If not set,
337 .Sq Nm more Fl s
338 is used. See
339 .Xr more 1 .
340 .El
341 .Sh FILES
342 .Bl -tag -width indent
343 .It Pa /usr/share/man
344 Root of the standard manual page directory subtree
345 .It Pa /usr/share/man/man?/*
346 Unformatted manual entries
347 .It Pa /usr/share/man/whatis
348 Table of contents and keyword database
349 .It Pa man.cf
350 Default search order by section
351 .El
352 .Sh EXIT STATUS
353 .Ex -std man
354 .Sh EXAMPLES
355 .
356 .Ss Example 1: Creating a PostScript Version of a man page
357 .
358 The following example spools the
359 .Xr pipe 2
360 man page in PostScript to the default printer:
361 .Pp
362 .Dl % man -t -s 2 pipe
363 .Pp
364 Note that
365 .Xr mandoc 1
366 can be used to obtain the PostScript content directly.
367 .Ss Example 2: Creating a Text Version of a man page
368 The following example creates the
369 .Xr pipe 2
370 man page in ASCII text:
371 .Pp
372 .Dl % man pipe.2 | col -x -b > pipe.text
373 .Sh CODE SET INDEPENDENCE
374 Enabled.
375 .Sh INTERFACE STABILITY
376 .Sy Committed .
376 .Nm Committed .
377 .Sh SEE ALSO
378 .Xr apropos 1 ,
379 .Xr cat 1 ,
380 .Xr col 1 ,
381 .Xr mandoc 1 ,
382 .Xr more 1 ,
383 .Xr whatis 1 ,
384 .Xr environ 5 ,
385 .Xr man 5 ,
386 .Xr mdoc 5
387 .Sh NOTES
388 The
```

```
.Fl f
and
.Fl k
options use the
.Nm whatis
database, which is
created with the
.Fl w
option.
.Sh BUGS
The manual is supposed to be reproducible either on a phototypesetter or on an
ASCII terminal. However, on a terminal some information (indicated by
font changes, for instance) is lost.
```

```
**********************************************************
   14742 Wed Jul 30 20:55:13 2014
new/usr/src/man/man1/mandoc.1
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
.\"
.\" Permission to use, copy, modify, and distribute this software for any
.\" purpose with or without fee is hereby granted, provided that the above
.\" copyright notice and this permission notice appear in all copies.
.\"
.\" THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
.\" WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
.\" MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
.\" ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
.\" WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
.\" ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
.\" OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
.\"
.\"
.\" Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
.\"
.Dd Jul 30, 2014
.Dd Jul 16, 2014
.Dt MANDOC 1
.Os
.Sh NAME
.Nm mandoc
.Nd format and display UNIX manuals
.Sh SYNOPSIS
.Nm mandoc
.Op Fl V
.Op Fl m Ns Ar format
.Op Fl O Ns Ar option
.Op Fl T Ns Ar output
.Op Fl W Ns Ar level
.Op Ar
.Sh DESCRIPTION
The
.Nm
utility formats
.Ux
manual pages for display.
.Pp
By default,
.Nm
reads
.Xr mdoc 5
or
.Xr man 5
text from stdin, implying
.Fl m Ns Cm andoc ,
and produces
.Fl T Ns Cm ascii
output.
.Pp
The arguments are as follows:
.Bl -tag -width Ds
.It Fl m Ns Ar format
Input format.
See
.Sx Input Formats
for available formats.
```



```
Defaults to
.Fl m Ns Cm andoc .
.It Fl O Ns Ar option
Comma-separated output options.
.It Fl T Ns Ar output
Output format.
See
.Sx Output Formats
for available formats.
Defaults to
.Fl T Ns Cm ascii .
.It Fl V
Print version and exit.
.It Fl W Ns Ar level
Specify the minimum message
.Ar level
to be reported on the standard error output and to affect the exit status.
The
.Ar level
can be
.Cm warning ,
.Cm error ,
or
.Cm fatal .
The default is
.Fl W Ns Cm fatal ;
.Fl W Ns Cm all
is an alias for
.Fl W Ns Cm warning .
See
.Sx EXIT STATUS
and
.Sx DIAGNOSTICS
for details.
.Pp
The special option
.Fl W Ns Cm stop
tells
.Nm
to exit after parsing a file that causes warnings or errors of at least
the requested level.
No formatted output will be produced from that file.
If both a
.Ar level
and
.Cm stop
are requested, they can be joined with a comma, for example
.Fl W Ns Cm error , Ns Cm stop .
.It Ar file
Read input from zero or more files.
If unspecified, reads from stdin.
If multiple files are specified,
.Nm
will halt with the first failed parse.
.El
.Ss Input Formats
The
.Nm
utility accepts
.Xr mdoc 5
and
.Xr man 5
input with
.Fl m Ns Cm doc
and
.Fl m Ns Cm an ,
```

```
125 respectively.
126 The
127 .Xr mdoc 5
128 format is
129 .Em strongly
130 recommended;
131 .Xr man 5
132 should only be used for legacy manuals.
133 .Pp
134 A third option,
135 .Fl m Ns Cm andoc ,
136 which is also the default, determines encoding on-the-fly: if the first
137 non-comment macro is
138 .Sq \&Dd
139 or
140 .Sq \&Dt ,
141 the
142 .Xr mdoc 5
143 parser is used; otherwise, the
144 .Xr man 5
145 parser is used.
146 .Pp
147 If multiple
148 files are specified with
149 .Fl m Ns Cm andoc ,
150 each has its file-type determined this way.
151 If multiple files are
152 specified and
153 .Fl m Ns Cm doc
154 or
155 .Fl m Ns Cm an
156 is specified, then this format is used exclusively.
157 .Ss Output Formats
158 The
159 .Nm
160 utility accepts the following
161 .Fl T
162 arguments, which correspond to output modes:
163 .Bl -tag -width "-Tlocale"
164 .It Fl T Ns Cm ascii
165 Produce 7-bit ASCII output.
166 This is the default.
167 See
168 .Sx ASCII Output .
169 .It Fl T Ns Cm html
170 Produce strict CSS1/HTML-4.01 output.
171 See
172 .Sx HTML Output .
173 .It Fl T Ns Cm lint
174 Parse only: produce no output.
175 Implies
176 .Fl W Ns Cm warning .
177 .It Fl T Ns Cm locale
178 Encode output using the current locale.
179 See
180 .Sx Locale Output .
181 .It Fl T Ns Cm man
182 Produce
183 .Xr man 5
184 format output.
185 See
186 .Sx Man Output .
187 .It Fl T Ns Cm pdf
188 Produce PDF output.
189 See
190 .Sx PDF Output .
```



```
191 .It Fl T Ns Cm ps
192 Produce PostScript output.
193 See
194 .Sx PostScript Output .
195 .It Fl T Ns Cm tree
196 Produce an indented parse tree.
197 .It Fl T Ns Cm utf8
198 Encode output in the UTF\-8 multi-byte format.
199 See
200 .Sx UTF\-8 Output .
201 .It Fl T Ns Cm xhtml
202 Produce strict CSS1/XHTML-1.0 output.
203 See
204 .Sx XHTML Output .
205 .El
206 .Pp
207 If multiple input files are specified, these will be processed by the
208 corresponding filter in-order.
209 .Ss ASCII Output
210 Output produced by
211 .Fl T Ns Cm ascii ,
212 which is the default, is rendered in standard 7-bit ASCII documented in
213 .Xr ascii 5 .
214 .Pp
215 Font styles are applied by using back-spaced encoding such that an
216 underlined character
217 .Sq c
218 is rendered as
219 .Sq _ Ns \e[bs] Ns c ,
220 where
221 .Sq \e[bs]
222 is the back-space character number 8.
223 Emboldened characters are rendered as
224 .Sq c Ns \e[bs] Ns c .
225 .Pp
226 The special characters documented in
227 .Xr mandoc_char 5
228 are rendered best-effort in an ASCII equivalent.
229 If no equivalent is found,
230 .Sq \&?
231 is used instead.
232 .Pp
233 Output width is limited to 78 visible columns unless literal input lines
234 exceed this limit.
235 .Pp
236 The following
237 .Fl O
238 arguments are accepted:
239 .Bl -tag -width Ds
240 .It Cm indent Ns = Ns Ar indent
241 The left margin for normal text is set to
242 .Ar indent
243 blank characters instead of the default of five for
244 .Xr mdoc 5
245 and seven for
246 .Xr man 5 .
247 Increasing this is not recommended; it may result in degraded formatting,
248 for example overfull lines or ugly line breaks.
249 .It Cm width Ns = Ns Ar width
250 The output width is set to
251 .Ar width ,
252 which will normalise to \(>=60.
253 .El
254 .Ss HTML Output
255 Output produced by
256 .Fl T Ns Cm html
```

```
257 conforms to HTML-4.01 strict.
258 .Pp
259 The
260 .Pa example.style.css
261 file documents style-sheet classes available for customising output.
262 If a style-sheet is not specified with
263 .Fl O Ns Ar style ,
264 .Fl T Ns Cm html
265 defaults to simple output readable in any graphical or text-based web
266 browser.
267 .Pp
268 Special characters are rendered in decimal-encoded UTF\-8.
269 .Pp
270 The following
271 .Fl O
272 arguments are accepted:
273 .Bl -tag -width Ds
274 .It Cm fragment
275 Omit the
276 .Aq !DOCTYPE
277 declaration and the
278 .Aq html ,
279 .Aq head ,
280 and
281 .Aq body
282 elements and only emit the subtree below the
283 .Aq body
284 element.
285 The
286 .Cm style
287 argument will be ignored.
288 This is useful when embedding manual content within existing documents.
289 .It Cm includes Ns = Ns Ar fmt
290 The string
291 .Ar fmt ,
292 for example,
293 .Ar ../src/%I.html ,
294 is used as a template for linked header files (usually via the
295 .Sq \&In
296 macro).
297 Instances of
298 .Sq \&%I
299 are replaced with the include filename.
300 The default is not to present a
301 hyperlink.
302 .It Cm man Ns = Ns Ar fmt
303 The string
304 .Ar fmt ,
305 for example,
306 .Ar ../html%S/%N.%S.html ,
307 is used as a template for linked manuals (usually via the
308 .Sq \&Xr
309 macro).
310 Instances of
311 .Sq \&%N
312 and
313 .Sq %S
314 are replaced with the linked manual’s name and section, respectively.
315 If no section is included, section 1 is assumed.
316 The default is not to
317 present a hyperlink.
318 .It Cm style Ns = Ns Ar style.css
319 The file
320 .Ar style.css
321 is used for an external style-sheet.
322 This must be a valid absolute or
```



```
323 relative URI.
324 .El
325 .Ss Locale Output
326 Locale-depending output encoding is triggered with
327 .Fl T Ns Cm locale .
328 This option is not available on all systems: systems without locale
329 support, or those whose internal representation is not natively UCS-4,
330 will fall back to
331 .Fl T Ns Cm ascii .
332 See
333 .Sx ASCII Output
334 for font style specification and available command-line arguments.
335 .Ss Man Output
336 Translate input format into
337 .Xr man 5
338 output format.
339 This is useful for distributing manual sources to legacy systems
340 lacking
341 .Xr mdoc 5
342 formatters.
343 .Pp
344 If
345 .Xr mdoc 5
346 is passed as input, it is translated into
347 .Xr man 5 .
348 If the input format is
349 .Xr man 5 ,
350 the input is copied to the output, expanding any
351 .Xr mandoc_roff 5
352 .Sq so
353 requests.
354 The parser is also run, and as usual, the
355 .Fl W
356 level controls which
357 .Sx DIAGNOSTICS
358 are displayed before copying the input to the output.
359 .Ss PDF Output
360 PDF-1.1 output may be generated by
361 .Fl T Ns Cm pdf .
362 See
363 .Sx PostScript Output
364 for
365 .Fl O
366 arguments and defaults.
367 .Ss PostScript Output
368 PostScript
369 .Qq Adobe-3.0
370 Level-2 pages may be generated by
371 .Fl T Ns Cm ps .
372 Output pages default to letter sized and are rendered in the Times font
373 family, 11-point.
374 Margins are calculated as 1/9 the page length and width.
375 Line-height is 1.4m.
376 .Pp
377 Special characters are rendered as in
378 .Sx ASCII Output .
379 .Pp
380 The following
381 .Fl O
382 arguments are accepted:
383 .Bl -tag -width Ds
384 .It Cm paper Ns = Ns Ar name
385 The paper size
386 .Ar name
387 may be one of
388 .Ar a3 ,
```

```
.Ar a4 ,
.Ar a5 ,
.Ar legal ,
or
.Ar letter .
You may also manually specify dimensions as
.Ar NNxNN ,
width by height in millimetres.
If an unknown value is encountered,
.Ar letter
is used.
.El
.Ss UTF\-8 Output
Use
.Fl T Ns Cm utf8
to force a UTF\-8 locale.
See
.Sx Locale Output
for details and options.
.Ss XHTML Output
Output produced by
.Fl T Ns Cm xhtml
conforms to XHTML-1.0 strict.
.Pp
See
.Sx HTML Output
for details; beyond generating XHTML tags instead of HTML tags, these
output modes are identical.
.Sh EXIT STATUS
The
.Nm
utility exits with one of the following values, controlled by the message
.Ar level
associated with the
.Fl W
option:
.Pp
.Bl -tag -width Ds -compact
.It 0
No warnings or errors occurred, or those that did were ignored because
they were lower than the requested
.Ar level .
.It 2
At least one warning occurred, but no error, and
.Fl W Ns Cm warning
was specified.
.It 3
At least one parsing error occurred, but no fatal error, and
.Fl W Ns Cm error
or
.Fl W Ns Cm warning
was specified.
.It 4
A fatal parsing error occurred.
.It 5
Invalid command line arguments were specified.
No input files have been read.
.It 6
An operating system error occurred, for example memory exhaustion or an
error accessing input files.
Such errors cause
.Nm
to exit at once, possibly in the middle of parsing or formatting a file.
.El
.Pp
Note that selecting
```



```
.Fl T Ns Cm lint
output mode implies
.Fl W Ns Cm warning .
.Sh EXAMPLES
To page manuals to the terminal:
.Pp
.Dl $ mandoc \-Wall,stop mandoc.1 2\*(Gt&1 | less
.Dl $ mandoc mandoc.1 mdoc.5 | less
.Pp
To produce HTML manuals with
.Ar style.css
as the style-sheet:
.Pp
.Dl $ mandoc \-Thtml -Ostyle=style.css mdoc.5 \*(Gt mdoc.5.html
.Pp
To check over a large set of manuals:
.Pp
.Dl $ mandoc \-Tlint 'find /usr/src -name \e*\e.[1-9]'
.Pp
To produce a series of PostScript manuals for A4 paper:
.Pp
.Dl $ mandoc \-Tps \-Opaper=a4 mdoc.5 man.5 \*(Gt manuals.ps
.Pp
Convert a modern
.Xr mdoc 5
manual to the older
.Xr man 5
format, for use on systems lacking an
.Xr mdoc 5
parser:
.Pp
.Dl $ mandoc \-Tman foo.mdoc \*(Gt foo.man
.Sh DIAGNOSTICS
Standard error messages reporting parsing errors are prefixed by
.Pp
.Sm off
.D1 Ar file : line : column : \ level :
.Sm on
.Pp
where the fields have the following meanings:
.Bl -tag -width "column"
.It Ar file
The name of the input file causing the message.
.It Ar line
The line number in that input file.
Line numbering starts at 1.
.It Ar column
The column number in that input file.
Column numbering starts at 1.
If the issue is caused by a word, the column number usually
points to the first character of the word.
.It Ar level
The message level, printed in capital letters.
.El
.Pp
Message levels have the following meanings:
.Bl -tag -width "warning"
.It Cm fatal
The parser is unable to parse a given input file at all.
No formatted output is produced from that input file.
.It Cm error
An input file contains syntax that cannot be safely interpreted,
either because it is invalid or because
.Nm
does not implement it yet.
By discarding part of the input or inserting missing tokens,
```

```
521 the parser is able to continue, and the error does not prevent
522 generation of formatted output, but typically, preparing that
523 output involves information loss, broken document structure
524 or unintended formatting.
525 .It Cm warning
526 An input file uses obsolete, discouraged or non-portable syntax.
527 All the same, the meaning of the input is unambiguous and a correct
528 rendering can be produced.
529 Documents causing warnings may render poorly when using other
530 formatting tools instead of
531 .Nm .
532 .El
533 .Pp
534 Messages of the
535 .Cm warning
536 and
537 .Cm error
538 levels are hidden unless their level, or a lower level, is requested using a
539 .Fl W
540 option or
541 .Fl T Ns Cm lint
542 output mode.
543 .Pp
544 The
545 .Nm
546 utility may also print messages related to invalid command line arguments
547 or operating system errors, for example when memory is exhausted or
548 input files cannot be read.
549 Such messages do not carry the prefix described above.
550 .Sh COMPATIBILITY
551 This section summarises
552 .Nm
553 compatibility with GNU troff.
554 Each input and output format is separately noted.
555 .Ss ASCII Compatibility
556 .Bl -bullet -compact
557 .It
558 Unrenderable unicode codepoints specified with
559 .Sq \e[uNNNN]
560 escapes are printed as
561 .Sq \&?
562 in mandoc.
563 In GNU troff, these raise an error.
564 .It
565 The
566 .Sq \&Bd \-literal
567 and
568 .Sq \&Bd \-unfilled
569 macros of
570 .Xr mdoc 5
571 in
572 .Fl T Ns Cm ascii
573 are synonyms, as are \-filled and \-ragged.
574 .It
575 In historic GNU troff, the
576 .Sq \&Pa
577 .Xr mdoc 5
578 macro does not underline when scoped under an
579 .Sq \&It
580 in the FILES section.
581 This behaves correctly in
582 .Nm .
583 .It
584 A list or display following the
585 .Sq \&Ss
586 .Xr mdoc 5
```



```
587 macro in
588 .Fl T Ns Cm ascii
589 does not assert a prior vertical break, just as it doesn’t with
590 .Sq \&Sh .
591 .It
592 The
593 .Sq \&na
594 .Xr man 5
595 macro in
596 .Fl T Ns Cm ascii
597 has no effect.
598 .It
599 Words aren’t hyphenated.
600 .El
601 .Ss HTML/XHTML Compatibility
602 .Bl -bullet -compact
603 .It
604 The
605 .Sq \efP
606 escape will revert the font to the previous
607 .Sq \ef
608 escape, not to the last rendered decoration, which is now dictated by
609 CSS instead of hard-coded.
610 It also will not span past the current scope,
611 for the same reason.
612 Note that in
613 .Sx ASCII Output
614 mode, this will work fine.
615 .It
616 The
617 .Xr mdoc 5
618 .Sq \&Bl \-hang
619 and
620 .Sq \&Bl \-tag
621 list types render similarly (no break following overreached left-hand
622 side) due to the expressive constraints of HTML.
623 .It
624 The
625 .Xr man 5
626 .Sq IP
627 and
628 .Sq TP
629 lists render similarly.
630 .El
631 .Sh INTERFACE STABILITY
632 The
633 .Nm
634 utility is
635 .Sy Committed ,
635 .Nm Committed ,
636 but the details of specific output formats other than ASCII are
637 .Nm Uncommitted .
638 .Sh SEE ALSO
639 .Xr eqn 5 ,
640 .Xr man 5 ,
641 .Xr mandoc_char 5 ,
642 .Xr mdoc 5 ,
643 .Xr mandoc_roff 5 ,
644 .Xr tbl 5
645 .Sh AUTHORS
646 The
647 .Nm
648 utility was written by
649 .An Kristaps Dzonsons ,
650 .Mt kristaps@bsd.lv .
651 .Sh CAVEATS
```

```
652 In
653 .Fl T Ns Cm html
654 and
655 .Fl T Ns Cm xhtml ,
656 the maximum size of an element attribute is determined by
657 .Dv BUFSIZ ,
658 which is usually 1024 bytes.
659 Be aware of this when setting long link
660 formats such as
661 .Fl O Ns Cm style Ns = Ns Ar really/long/link .
662 .Pp
663 Nesting elements within next-line element scopes of
664 .Fl m Ns Cm an ,
665 such as
666 .Sq br
667 within an empty
668 .Sq B ,
669 will confuse
670 .Fl T Ns Cm html
671 and
672 .Fl T Ns Cm xhtml
673 and cause them to forget the formatting of the prior next-line scope.
674 .Pp
675 The
676 .Sq \(aq
677 control character is an alias for the standard macro control character
678 and does not emit a line-break as stipulated in GNU troff.
```

```
**********************************************************
    8753 Wed Jul 30 20:55:13 2014
new/usr/src/man/man5/man.5
5051 import mdocml-1.12.3
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Approved by: TBD
**********************************************************
```

```
.\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
.\" Copyright (c) 1995, Sun Microsystems, Inc.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.Dd "Jul 30, 2014"
.Dd "Jul 19, 2014"
.Dt MAN 5
.Os
.Sh NAME
.Nm man
.Nd macros to format Reference Manual pages
.Sh SYNOPSIS
.Nm mandoc
.Fl T Ar man
.Ar
.Nm nroff
.Fl man
.Ar
.Nm troff
.Fl man
.Ar
.Sh DESCRIPTION
These macros are used to lay out the reference pages in this manual. Note: if
.Ar file
contains format input for a preprocessor, the commands shown
above must be piped through the appropriate preprocessor. This is handled
automatically by the
.Xr man 1
command. See the
.Sx Conventions
section.
.Lp
Any text argument
.Ar t
may be zero to six words. Quotes may be used to
include SPACE characters in a
.Qq word .
If
.Ar text
is empty, the special
treatment is applied to the next input line with text to be printed. In this
way
.Nm \&.I
may be used to italicize a whole line, or
.Nm \&.SB
may be used to make small bold letters.
.Lp
A prevailing indent distance is remembered between successive indented
paragraphs, and is reset to default value upon reaching a non-indented
paragraph.  Default units for indents
.Nm i
are ens.
.Lp
Type font and size are reset to default values before each paragraph, and after
processing font and size setting macros.
.Pp
These strings are predefined by
.Nm -man :
```



```
.Bl -tag -width Ds
.It Nm \e*R
.Sq \(rg ,
.Sq (Reg)
in
.Nm nroff .
.It Nm \e*S
Change to default type size.
.El
.Sh "Requests"
* n.t.l. = next text line; p.i. = prevailing indent
.Bl -column ".TH n s d f m" "Cause " "t=n.t.l.*" "Explanation " -offset Ds
.It Sy Request  Sy Cause        Sy "If No"      Sy Explanation
.It ""  Sy Break        Sy "Argument"   ""
.It Nm \&.B Ar "t"      no      Ar t Ns =n.t.l.*        Text is in bold font.
.It Nm \&.BI Ar t       no      Ar t Ns =n.t.l. Join words, alternating bold and
.It Nm \&.BR Ar t       no      Ar t Ns =n.t.l. Join words, alternating bold and
.It Nm \&.DT    no      Li \&.5i 1i...  Restore default tabs.
.It Nm \&.HP Ar i       yes     Ar i Ns =p.i.*  "Begin paragraph with hanging in
.It Nm \&.I Ar t        no      Ar t Ns =n.t.l. Text is italic.
.It Nm \&.IB Ar t       no      Ar t Ns =n.t.l. Join words, altenrating italic a
.It Nm \&.IP Ar x Ar i  yes     Ar x Ns =""     Same as
.Nm \&.TP
with tag
.Ar x .
.It Nm \&.IR Ar t       no      Ar t Ns =n.t.l. Join words, alternating italic a
.It Nm \&.IX Ar t       no      -       Index macro, not used (obsolete).
.It Nm \&.LP    yes     -       Begin left-aligned paragraph. Set prevailing ind
.It Nm \&.P     yes     -       Same as
.Nm \&.LP .
.It Nm \&.PD Ar d       no      Ar d Ns =.4v    Set vertical distance between pa
.It Nm \&.PP    yes     -       Same as
.Nm \&.LP .
.It Nm \&.RE    yes     -       End of relative indent. Restores prevailing inde
.It Nm \&.RB Ar t       no      Ar t Ns =n.t.l. Join words, alternating roman an
.It Nm \&.RI Ar t       no      Ar t Ns =n.t.l. Join words, alternating roman an
.It Nm \&.RS Ar i       yes     Ar i Ns =p.i.   Start relative indent, increase
Sets prevailing indent to .5i for nested indents.
.It Nm \&.SB Ar t       no      -       Reduce size of text by 1 point, make tex
.It Nm \&.SH Ar t       yes     -       Section Heading.
.It Nm \&.SM Ar t       no      Ar t Ns =n.t.l. Reduce size of text by 1 point.
.It Nm \&.SS Ar t       yes     Ar t Ns =n.t.l. Section Subheading.
.It Nm \&.TH Ar n s d f m       yes     -       Begin reference page Ar n , No o
.It Nm \&.TP Ar i       yes     Ar i Ns =p.i.   Begin indented paragraph, with t
.Ar i .
.It Nm \&.TX Ar t p     no      -       Resolve the title abbreviation Ar t ; No
.El
.Ss "Conventions"
When formatting a manual page,
.Nm
examines the first line to determine
whether it requires special processing. For example a first line consisting of:
.Lp
.Dl \&'\e" t
.Lp
indicates that the manual page must be run through the
.Xr tbl 1
preprocessor.
.Lp
A typical manual page for a command or function is laid out as follows:
.Bl -tag -width ".SH RETURN VALUES"
.
.It Nm \&.TH Ar title Op "1-9"
.
The name of the command or function, which serves as the title of the manual
page. This is followed by the number of the section in which it appears.
```

```
.
.It Nm SH NAME
.
The name, or list of names, by which the command is called, followed by a dash
and then a one-line summary of the action performed. All in roman font, this
section contains no
.Xr troff 1
commands or escapes, and no macro requests.
It is used to generate the database used by the
.Xr whatis 1
command.
.
.It Nm SH SYNOPSIS
.Bl -tag -width "Functions:"
.It Sy Commands:
The syntax of the command and its arguments, as typed on the command line.
When in boldface, a word must be typed exactly as printed.  When in italics, a
word can be replaced with an argument that you supply. References to bold or
italicized items are not capitalized in other sections, even when they begin a
sentence.
.Lp
Syntactic symbols appear in roman face:
.Bl -tag -width "   "
.It Op " "
An argument, when surrounded by brackets is optional.
.It |
Arguments separated by a vertical bar are exclusive. You can supply only one
item from such a list.
.It \&.\|.\|.
Arguments followed by an ellipsis can be repeated. When an ellipsis follows a
bracketed set, the expression within the brackets can be repeated.
.El
.It Sy Functions:
If required, the data declaration, or
.Li #include
directive, is shown first,
followed by the  function declaration. Otherwise, the function declaration is
shown.
.El
.
.It Nm \&.SH DESCRIPTION
.
A narrative overview of the command or function’s external behavior. This
includes how it interacts with files or data, and how it handles the standard
input, standard output and standard error. Internals and implementation details
are normally omitted. This section attempts to provide a succinct overview in
answer to the question, "what does it do?"
.Lp
Literal text from the synopsis appears in constant width, as do literal
filenames and references to items that appear elsewhere in the  reference
manuals. Arguments are italicized.
.Lp
If a command interprets either subcommands or an input grammar, its command
interface or input grammar is normally described in a
.Nm USAGE
section, which follows the
.Nm OPTIONS
section.  The
.Nm DESCRIPTION
section only
describes the behavior of the command itself, not that of subcommands.
.
.It Nm \&.SH OPTIONS
.
The list of options along with a description of how each affects the command’s
operation.
```



```
.
.It Nm \&.SH RETURN VALUES
.
A list of the values the library routine will return to the calling  program
and the conditions that cause these values to be returned.
.
.It Nm \&.SH EXIT STATUS
.
A list of the values the utility will return to the calling  program or shell,
and the conditions that cause these values to be  returned.
.
.It Nm \&.SH FILES
.
A list of files associated with the command or function.
.
.It Nm \&.SH SEE ALSO
.
A comma-separated list of related manual pages, followed by references to other
published materials.
.
.It Nm \&.SH DIAGNOSTICS
.
A list of diagnostic messages and an explanation of each.
.
.It Nm \&.SH BUGS
.
A description of limitations, known defects, and possible problems associated
with the command or function.
.El
.Sh FILES
.Pa /usr/share/man/whatis
.Sh NOTES
The
.Nm
package should not be used for new documentation.  The
.Xr mdoc 5 ,
package is preferred, as it uses semantic markup rather than physical markup.
.Sh CODE SET INDEPENDENCE
When processed with
.Xr mandoc 1 ,
this package is Code Set Independent. However, when processed with
legacy tools such as
.Xr nroff 1
and
.Xr troff 1 ,
the use of multi-byte characters may not be supported.
.Sh INTERFACE STABILITY
.Sy Obsolete Committed .
.Nm Obsolete Committed .
The
.Xr mdoc 5
package should be used instead.
.Sh SEE ALSO
.Xr eqn 1 ,
.Xr man 1 ,
.Xr mandoc 1 ,
.Xr nroff 1 ,
.Xr troff 1 ,
.Xr tbl 1 ,
.Xr whatis 1 ,
.Xr mdoc 5 ,
.Rs
.%A Dale Dougherty and Tim O’Reilly
.%B Unix Text Processing
.Re
```