```
**********************************************************
   10887 Tue Jul 15 13:48:02 2014
new/usr/src/cmd/Makefile
Finish integration.  Use mandoc_preconv, etc.
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  23 # Copyright 2010 Nexenta Systems, Inc.  All rights reserved.
  24 # Copyright (c) 2012 Joyent, Inc.  All rights reserved.
  25 # Copyright (c) 2012 by Delphix. All rights reserved.
  26 # Copyright (c) 2013 DEY Storage Systems, Inc. All rights reserved.
  27 # Copyright 2014 Garrett D'Amore <garrett@damore.org>

  29 include ../Makefile.master

  31 #
  32 #       Note that the commands 'lp', and 'perl' are first in
  31 #       Note that the commands 'agents', 'lp', 'perl', and 'man' are first in
  33 #       the list, violating alphabetical order.  This is because they are very
  34 #       long-running and should be given the most wall-clock time for a
  35 #       parallel build.
  36 #
  37 #       Commands in the FIRST_SUBDIRS list are built before starting the build
  38 #       of other commands.  Currently this includes only 'isaexec' and
  39 #       'platexec'.  This is necessary because $(ROOT)/usr/lib/isaexec or
  40 #       $(ROOT)/usr/lib/platexec must exist when some other commands are built
  41 #       because their 'make install' creates a hard link to one of them.
  42 #
  43 #       Commands are listed one per line so that TeamWare can auto-merge most
  44 #       changes.
  45 #

  47 FIRST_SUBDIRS=          \
  48         isaexec         \
  49         platexec

  51 COMMON_SUBDIRS=         \
  52         allocate        \
  53         availdevs       \
  54         lp              \
  55         perl            \
  55         man             \
  56         Adm             \
  57         abi             \
  58         adbgen          \
  59         acct            \
```



```
  60         acctadm         \
  61         arch            \
  62         asa             \
  63         ast             \
  64         audio           \
  65         auths           \
  66         autopush        \
  67         avs             \
  68         awk             \
  69         awk_xpg4        \
  70         backup          \
  71         banner          \
  72         bart            \
  73         basename        \
  74         bc              \
  75         bdiff           \
  76         beadm           \
  77         bfs             \
  78         bnu             \
  79         boot            \
  80         busstat         \
  81         cal             \
  82         calendar        \
  83         captoinfo       \
  84         cat             \
  85         cdrw            \
  86         cfgadm          \
  87         checkeq         \
  88         checknr         \
  89         chgrp           \
  90         chmod           \
  91         chown           \
  92         chroot          \
  93         clear           \
  94         clinfo          \
  95         cmd-crypto      \
  96         cmd-inet        \
  97         col             \
  98         compress        \
  99         consadm         \
 100         coreadm         \
 101         cpio            \
 102         cpc             \
 103         cron            \
 104         crypt           \
 105         csh             \
 106         csplit          \
 107         ctrun           \
 108         ctstat          \
 109         ctwatch         \
 110         datadm          \
 111         date            \
 112         dc              \
 113         dd              \
 114         deroff          \
 115         devfsadm        \
 116         syseventd       \
 117         devctl          \
 118         devinfo         \
 119         devmgmt         \
 120         devprop         \
 121         dfs.cmds        \
 122         diff            \
 123         diff3           \
 124         diffmk          \
 125         dircmp          \
```

```
126         dirname         \
127         dis             \
128         diskmgtd        \
129         dispadmin       \
130         dladm           \
131         dlstat          \
132         dmesg           \
133         dodatadm        \
134         dtrace          \
135         du              \
136         dumpadm         \
137         dumpcs          \
138         echo            \
139         ed              \
140         eeprom          \
141         egrep           \
142         eject           \
143         emul64ioctl     \
144         enhance         \
145         env             \
146         eqn             \
147         expand          \
148         expr            \
149         exstr           \
150         factor          \
151         false           \
152         fcinfo          \
153         fcoesvc         \
154         fdetach         \
155         fdformat        \
156         fdisk           \
157         filesync        \
158         fgrep           \
159         file            \
160         find            \
161         flowadm         \
162         flowstat        \
163         fm              \
164         fmt             \
165         fmthard         \
166         fmtmsg          \
167         fold            \
168         format          \
169         fs.d            \
170         fstyp           \
171         fuser           \
172         fwflash         \
173         gcore           \
174         gencat          \
175         geniconvtbl     \
176         genmsg          \
177         getconf         \
178         getdevpolicy    \
179         getent          \
180         getfacl         \
181         getmajor        \
182         getopt          \
183         gettext         \
184         gettxt          \
185         grep            \
186         grep_xpg4       \
187         groups          \
188         grpck           \
189         gss             \
190         hal             \
191         halt            \
```



```
192         head            \
193         hostid          \
194         hostname        \
195         hotplug         \
196         hotplugd        \
197         hwdata          \
198         ibd_upgrade     \
199         id              \
200         idmap           \
201         infocmp         \
202         init            \
203         initpkg         \
204         install.d       \
205         intrd           \
206         intrstat        \
207         ipcrm           \
208         ipcs            \
209         ipdadm          \
210         ipf             \
211         isainfo         \
212         isalist         \
213         itutools        \
214         iscsiadm        \
215         iscsid          \
216         iscsitsvc       \
217         isns            \
218         itadm           \
219         java            \
220         kbd             \
221         keyserv         \
222         killall         \
223         krb5            \
224         ksh             \
225         kvmstat         \
226         last            \
227         lastcomm        \
228         latencytop      \
229         ldap            \
230         ldapcachemgr    \
231         lgrpinfo        \
232         line            \
233         link            \
234         dlmgmtd         \
235         listen          \
236         loadkeys        \
237         locale          \
238         localedef       \
239         lockstat        \
240         locator         \
241         lofiadm         \
242         logadm          \
243         logger          \
244         login           \
245         logins          \
246         look            \
247         ls              \
248         luxadm          \
249         lvm             \
250         mach            \
251         machid          \
252         mail            \
253         mailx           \
254         makekey         \
255         man             \
256         mandoc          \
257         mdb             \
```

```
258         mesg            \
259         mkdir           \
260         mkfifo          \
261         mkfile          \
262         mkmsgs          \
263         mknod           \
264         mkpwdict        \
265         mktemp          \
266         modload         \
267         more            \
268         mpathadm        \
269         msgfmt          \
270         msgid           \
271         mt              \
272         mv              \
273         mvdir           \
274         ndmpadm         \
275         ndmpd           \
276         ndmpstat        \
277         netadm          \
278         netfiles        \
279         newform         \
280         newgrp          \
281         news            \
282         newtask         \
283         nice            \
284         nl              \
285         nlsadmin        \
286         nohup           \
287         nsadmin         \
288         nscd            \
289         oamuser         \
290         oawk            \
291         od              \
292         pack            \
293         pagesize        \
294         passmgmt        \
295         passwd          \
296         pathchk         \
297         pbind           \
298         pcidr           \
299         pcitool         \
300         pfexec          \
301         pfexecd         \
302         pginfo          \
303         pgstat          \
304         pgrep           \
305         picl            \
306         plimit          \
307         policykit       \
308         pools           \
309         power           \
310         powertop        \
311         ppgsz           \
312         pg              \
313         plockstat       \
314         pr              \
315         prctl           \
316         print           \
317         printf          \
318         priocntl        \
319         profiles        \
320         projadd         \
321         projects        \
322         prstat          \
323         prtconf         \
```



```
324         prtdiag         \
325         prtvtoc         \
326         ps              \
327         psradm          \
328         psrinfo         \
329         psrset          \
330         ptools          \
331         pwck            \
332         pwconv          \
333         pwd             \
334         pyzfs           \
335         raidctl         \
336         ramdiskadm      \
337         rcap            \
338         rcm_daemon      \
339         rctladm         \
340         refer           \
341         regcmp          \
342         renice          \
343         rexd            \
344         rm              \
345         rmdir           \
346         rmformat        \
347         rmmount         \
348         rmt             \
349         rmvolmgr        \
350         roles           \
351         rpcbind         \
352         rpcgen          \
353         rpcinfo         \
354         rpcsvc          \
355         runat           \
356         sa              \
357         saf             \
358         sasinfo         \
359         savecore        \
360         sbdadm          \
361         script          \
362         scsi            \
363         sdiff           \
364         sdpadm          \
365         sed             \
366         sendmail        \
367         setfacl         \
368         setmnt          \
369         setpgrp         \
370         setuname        \
371         sgs             \
372         sh              \
373         shcomp          \
374         smbios          \
375         smbsrv          \
376         smserverd       \
377         soelim          \
378         sort            \
379         spell           \
380         split           \
381         sqlite          \
382         srchtxt         \
383         srptadm         \
384         srptsvc         \
385         ssh             \
386         stat            \
387         stmfadm         \
388         stmfproxy       \
389         stmfsvc         \
```

```
390         stmsboot        \
391         streams         \
392         strings         \
393         su              \
394         sulogin         \
395         sunpc           \
396         svc             \
397         svr4pkg         \
398         swap            \
399         sync            \
400         sysdef          \
401         syseventadm     \
402         syslogd         \
403         tabs            \
404         tail            \
405         tar             \
406         tbl             \
407         tcopy           \
408         tcpd            \
409         terminfo        \
410         th_tools        \
411         tic             \
412         time            \
413         tip             \
414         tnf             \
415         touch           \
416         tput            \
417         tr              \
418         trapstat        \
419         troff           \
420         true            \
421         truss           \
422         tsol            \
423         tty             \
424         ttymon          \
425         tzreload        \
426         uadmin          \
427         ul              \
428         uname           \
429         units           \
430         unlink          \
431         unpack          \
432         userattr        \
433         users           \
434         utmp_update     \
435         utmpd           \
436         valtools        \
437         vgrind          \
438         vi              \
439         volcheck        \
440         volrmmount      \
441         vrrpadm         \
442         vscan           \
443         vt              \
444         w               \
445         wall            \
446         which           \
447         who             \
448         whodo           \
449         wracct          \
450         write           \
451         wusbadm         \
452         xargs           \
453         xstr            \
454         yes             \
455         ypcmd           \
```



```
456         yppasswd        \
457         zdb             \
458         zdump           \
459         zfs             \
460         zhack           \
461         zic             \
462         zinject         \
463         zlogin          \
464         zoneadm         \
465         zoneadmd        \
466         zonecfg         \
467         zonename        \
468         zpool           \
469         zlook           \
470         zonestat        \
471         zstreamdump     \
472         ztest

474 i386_SUBDIRS=           \
475         acpihpd         \
476         addbadsec       \
477         biosdev         \
478         diskscan        \
479         lms             \
480         ntfsprogs       \
481         parted          \
482         rtc             \
483         ucodeadm        \
484         xvm

486 sparc_SUBDIRS=          \
487         cvcd            \
488         dcs             \
489         device_remap    \
490         drd             \
491         fruadm          \
492         ldmad           \
493         oplhpd          \
494         prtdscp         \
495         prtfru          \
496         scadm           \
497         sckmd           \
498         sf880drd        \
499         virtinfo        \
500         vntsd

502 #
503 # Commands that are messaged.  Note that 'lp' comes first
504 # (see previous comment about 'lp'.)
501 # Commands that are messaged.  Note that 'lp' and 'man' come first
502 # (see previous comment about 'lp' and 'man').
505 #
506 MSGSUBDIRS=             \
507         lp              \
506         man             \
508         abi             \
509         acctadm         \
510         allocate        \
511         asa             \
512         audio           \
513         audit           \
514         auditconfig     \
515         auditd          \
516         auditrecord     \
517         auditset        \
518         auths           \
```

```
519         autopush        \
520         avs             \
521         awk             \
522         awk_xpg4        \
523         backup          \
524         banner          \
525         bart            \
526         basename        \
527         beadm           \
528         bnu             \
529         busstat         \
530         cal             \
531         cat             \
532         cdrw            \
533         cfgadm          \
534         checkeq         \
535         checknr         \
536         chgrp           \
537         chmod           \
538         chown           \
539         cmd-crypto      \
540         cmd-inet        \
541         col             \
542         compress        \
543         consadm         \
544         coreadm         \
545         cpio            \
546         cpc             \
547         cron            \
548         csh             \
549         csplit          \
550         ctrun           \
551         ctstat          \
552         ctwatch         \
553         datadm          \
554         date            \
555         dc              \
556         dcs             \
557         dd              \
558         deroff          \
559         devfsadm        \
560         dfs.cmds        \
561         diff            \
562         diffmk          \
563         dladm           \
564         dlstat          \
565         du              \
566         dumpcs          \
567         ed              \
568         eject           \
569         env             \
570         eqn             \
571         expand          \
572         expr            \
573         fcinfo          \
574         fgrep           \
575         file            \
576         filesync        \
577         find            \
578         flowadm         \
579         flowstat        \
580         fm              \
581         fold            \
582         fs.d            \
583         fwflash         \
584         geniconvtbl     \
```



```
585         genmsg          \
586         getconf         \
587         getent          \
588         gettext         \
589         gettxt          \
590         grep            \
591         grep_xpg4       \
592         grpck           \
593         gss             \
594         halt            \
595         head            \
596         hostname        \
597         hotplug         \
598         id              \
599         idmap           \
600         isaexec         \
601         iscsiadm        \
602         iscsid          \
603         isns            \
604         itadm           \
605         kbd             \
606         krb5            \
607         ksh             \
608         last            \
609         ldap            \
610         ldapcachemgr    \
611         lgrpinfo        \
612         locale          \
613         lofiadm         \
614         logadm          \
615         logger          \
616         logins          \
617         ls              \
618         luxadm          \
619         lvm             \
620         mailx           \
621         man             \
622         mesg            \
623         mkdir           \
624         mkpwdict        \
625         mktemp          \
626         more            \
627         mpathadm        \
628         msgfmt          \
629         mv              \
630         ndmpadm         \
631         ndmpstat        \
632         newgrp          \
633         newtask         \
634         nice            \
635         nohup           \
636         oawk            \
637         pack            \
638         passwd          \
639         passmgmt        \
640         pathchk         \
641         pfexec          \
642         pg              \
643         pgrep           \
644         picl            \
645         pools           \
646         power           \
647         pr              \
648         praudit         \
649         print           \
650         profiles        \
```

```
651         projadd         \
652         projects        \
653         prstat          \
654         prtdiag         \
655         ps              \
656         psrinfo         \
657         ptools          \
658         pwconv          \
659         pwd             \
660         pyzfs           \
661         raidctl         \
662         ramdiskadm      \
663         rcap            \
664         rcm_daemon      \
665         refer           \
666         regcmp          \
667         renice          \
668         roles           \
669         rm              \
670         rmdir           \
671         rmformat        \
672         rmmount         \
673         rmvolmgr        \
674         sasinfo         \
675         sbdadm          \
676         scadm           \
677         script          \
678         scsi            \
679         sdiff           \
680         sdpadm          \
681         sgs             \
682         sh              \
683         shcomp          \
684         smbsrv          \
685         sort            \
686         split           \
687         srptadm         \
688         ssh             \
689         stat            \
690         stmfadm         \
691         stmsboot        \
692         strings         \
693         su              \
694         svc             \
695         svr4pkg         \
696         swap            \
697         syseventadm     \
698         syseventd       \
699         tabs            \
700         tar             \
701         tbl             \
702         time            \
703         tnf             \
704         touch           \
705         tput            \
706         troff           \
707         tsol            \
708         tty             \
709         ttymon          \
710         tzreload        \
711         ul              \
712         uname           \
713         units           \
714         unlink          \
715         unpack          \
716         userattr        \
```



```
717         valtools        \
718         vgrind          \
719         vi              \
720         volcheck        \
721         volrmmount      \
722         vrrpadm         \
723         vscan           \
724         w               \
725         who             \
726         whodo           \
727         wracct          \
728         write           \
729         wusbadm         \
730         xargs           \
731         yppasswd        \
732         zdump           \
733         zfs             \
734         zic             \
735         zlogin          \
736         zoneadm         \
737         zoneadmd        \
738         zonecfg         \
739         zonename        \
740         zpool           \
741         zonestat

743 sparc_MSGSUBDIRS=       \
744         fruadm          \
745         prtdscp         \
746         prtfru          \
747         virtinfo        \
748         vntsd

750 i386_MSGSUBDIRS=        \
751         ucodeadm

753 #
754 # commands that use dcgettext for localized time, LC_TIME
755 #
756 DCSUBDIRS=              \
757         cal             \
758         cfgadm          \
759         diff            \
760         ls              \
761         pr              \
762         ps              \
763         tar             \
764         w               \
765         who             \
766         whodo           \
767         write

769 #
770 # commands that belong only to audit.
771 #
772 AUDITSUBDIRS=           \
773         amt             \
774         audit           \
775         audit_warn      \
776         auditconfig     \
777         auditd          \
778         auditrecord     \
779         auditreduce     \
780         auditset        \
781         auditstat       \
782         praudit
```

```
784 #
785 # commands not owned by the systems group
786 #
787 BWOSDIRS=


790 all :=          TARGET = all
791 install :=      TARGET = install
792 clean :=        TARGET = clean
793 clobber :=      TARGET = clobber
794 lint :=         TARGET = lint
795 _msg :=         TARGET = _msg
796 _dc :=          TARGET = _dc

798 .KEEP_STATE:

800 SUBDIRS = $(COMMON_SUBDIRS) $($(MACH)_SUBDIRS)

802 .PARALLEL:      $(BWOSDIRS) $(SUBDIRS) $(MSGSUBDIRS) $(AUDITSUBDIRS)

804 all install clean clobber lint: $(FIRST_SUBDIRS) .WAIT $(SUBDIRS) \
805         $(AUDITSUBDIRS)

807 #
808 # Manifests cannot be checked in parallel, because we are using
809 # the global repository that is in $(SRC)/cmd/svc/seed/global.db.
810 # For this reason, to avoid .PARALLEL and .NO_PARALLEL conflicts,
811 # we spawn off a sub-make to perform the non-parallel 'make check'
812 #
813 check:
814         $(MAKE) -f Makefile.check check

816 #
817 # The .WAIT directive works around an apparent bug in parallel make.
818 # Evidently make was getting the target _msg vs. _dc confused under
819 # some level of parallelization, causing some of the _dc objects
820 # not to be built.
821 #
822 _msg: $(MSGSUBDIRS) $($(MACH)_MSGSUBDIRS) .WAIT _dc

824 _dc: $(DCSUBDIRS)

826 #
827 # Dependencies
828 #
829 fs.d: fstyp
830 ksh:    shcomp isaexec
831 mdb:    terminfo
832 print:  lp

834 $(FIRST_SUBDIRS) $(BWOSDIRS) $(SUBDIRS) $(AUDITSUBDIRS): FRC
835         @if [ -f $@/Makefile  ]; then \
836                 cd $@; pwd; $(MAKE) $(TARGET); \
837         else \
838                 true; \
839         fi

841 FRC:
```

```
**********************************************************
     937 Tue Jul 15 13:48:03 2014
new/usr/src/cmd/man/Makefile
Initial import of man functionality.
**********************************************************
```

```
#
# 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.
# CDDL HEADER START
#
# 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.
# The contents of this file are subject to the terms of the
# Common Development and Distribution License, Version 1.0 only
# (the "License").  You may not use this file except in compliance
# with the License.
#

# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# Copyright 2012 Nexenta Systems, Inc. All rights reserved.
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
#ident  "%Z%%M% %I%     %E% SMI"
#
# Copyright (c) 1990 by Sun Microsystems, Inc.
#
# cmd/man/Makefile

PROG=           man
LINKS=          apropos whatis
OBJS=           makewhatis.o man.o stringlist.o
SRCS=           $(OBJS:%.o=%.c)
include ../Makefile.cmd
SUBDIRS = src

include         $(SRC)/cmd/Makefile.cmd
all :=          TARGET= all
install :=      TARGET= install
clean :=        TARGET= clean
clobber :=      TARGET= clobber
lint :=         TARGET= lint
_msg :=         TARGET= catalog

CFLAGS +=       $(CCVERBOSE)
#for message catalog files
POFILE = man.po
POFILES = src/src.po

ROOTLINKS=      $(LINKS:%=$(ROOTBIN)/%)
.KEEP_STATE:

.KEEP_STATE :
```



```
all install clean lint: $(SUBDIRS)

all:            $(PROG)
clobber: $(SUBDIRS)     local_clobber

clean:
                $(RM) $(OBJS)
local_clobber:
        $(RM)   $(CLOBBERFILES)

install:        all $(ROOTPROG) $(ROOTLINKS)
_msg:   $(SUBDIRS)
        $(RM)   $(POFILE)
        cat $(POFILES)      > $(POFILE)
        $(RM)  $(MSGDOMAIN)/$(POFILE)
        cp $(POFILE) $(MSGDOMAIN)

lint:           lint_SRCS
$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

$(PROG):        $(OBJS)
                $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
                $(POST_PROCESS)

$(ROOTLINKS):   $(ROOTPROG)
                $(RM) $@; $(LN) $(ROOTPROG) $@

include         $(SRC)/cmd/Makefile.targ
FRC:
```

```
**********************************************************
    4680 Tue Jul 15 13:48:03 2014
new/usr/src/cmd/man/THIRDPARTYLICENSE
Initial import of man functionality.
**********************************************************
```

man.c:

makewhatis.c:

stringlist.c, stringlist.h:

```
**********************************************************
   18043 Tue Jul 15 13:48:03 2014
new/usr/src/cmd/man/makewhatis.c
Finish integration.  Use mandoc_preconv, etc.
**********************************************************
```

```
/*
 * Copyright (c) 2002 John Rochester
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer,
 *    in this position and unchanged.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>

#include <ctype.h>
#include <dirent.h>
#include <err.h>
#include <signal.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "man.h"
#include "stringlist.h"


/* Information collected about each man page in a section */
struct page_info {
        char    *filename;
        char    *name;
        char    *suffix;
        ino_t   inode;
};

/* An expanding string */
struct sbuf {
        char    *content;       /* the start of the buffer */
```



```
        char    *end;           /* just past the end of the content */
        char    *last;          /* the last allocated character */
};

/* Remove the last amount characters from the sbuf */
#define sbuf_retract(sbuf, amount) ((sbuf)->end -= (amount))
/* Return the length of the sbuf content */
#define sbuf_length(sbuf) ((sbuf)->end - (sbuf)->content)

typedef char *edited_copy(char *from, char *to, int length);

/*
 * While the whatis line is being formed, it is stored in whatis_proto.
 * When finished, it is reformatted into whatis_final and then appended
 * to whatis_lines.
 */
static struct sbuf      *whatis_proto;
static struct sbuf      *whatis_final;
static stringlist       *whatis_lines;  /* collected output lines */

static char tempfile[MAXPATHLEN];       /* path of temporary file, if any */

#define MDOC_COMMANDS   "ArDvErEvFlLiNmPa"


/* Free a struct page_info and its content */
static void
free_page_info(struct page_info *info)
{

        free(info->filename);
        free(info->name);
        free(info->suffix);
        free(info);
}

/*
 * Allocate and fill in a new struct page_info given the
 * name of the man section directory and the dirent of the file.
 * If the file is not a man page, return NULL.
 */
static struct page_info *
new_page_info(char *dir, struct dirent *dirent)
{
        struct page_info *info;
        int             basename_length;
        char            *suffix;
        struct stat     st;

        if ((info = malloc(sizeof (struct page_info))) == NULL)
                err(1, "malloc");
        basename_length = strlen(dirent->d_name);
        suffix = &dirent->d_name[basename_length];
        if (asprintf(&info->filename, "%s/%s", dir, dirent->d_name) == -1)
                err(1, "asprintf");
        for (;;) {
                if (--suffix == dirent->d_name || !isalnum(*suffix)) {
                        if (*suffix == '.')
                                break;
                        free(info->filename);
                        free(info);
                        return (NULL);
                }
        }
        *suffix++ = '\0';
        info->name = strdup(dirent->d_name);
```

```
128         info->suffix = strdup(suffix);
129         if (stat(info->filename, &st) < 0) {
130                 warn("%s", info->filename);
131                 free_page_info(info);
132                 return (NULL);
133         }
134         if (!S_ISREG(st.st_mode)) {
135                 free_page_info(info);
136                 return (NULL);
137         }
138         info->inode = st.st_ino;
139         return (info);
140 }

142 /*
143  * Reset sbuf length to 0.
144  */
145 static void
146 sbuf_clear(struct sbuf *sbuf)
147 {

149         sbuf->end = sbuf->content;
150 }

152 /*
153  * Allocate a new sbuf.
154  */
155 static struct sbuf *
156 new_sbuf(void)
157 {
158         struct sbuf     *sbuf;

160         if ((sbuf = malloc(sizeof (struct sbuf))) == NULL)
161                 err(1, "malloc");
162         if ((sbuf->content = (char *)malloc(LINE_ALLOC)) == NULL)
163                 err(1, "malloc");
164         sbuf->last = sbuf->content + LINE_ALLOC - 1;
165         sbuf_clear(sbuf);

167         return (sbuf);
168 }

170 /*
171  * Ensure that there is enough room in the sbuf
172  * for nchars more characters.
173  */
174 static void
175 sbuf_need(struct sbuf *sbuf, int nchars)
176 {
177         char *new_content;
178         size_t size, cntsize;

180         /* Double the size of the allocation until the buffer is big enough */
181         while (sbuf->end + nchars > sbuf->last) {
182                 size = sbuf->last + 1 - sbuf->content;
183                 size *= 2;
184                 cntsize = sbuf->end - sbuf->content;

186                 new_content = (char *)malloc(size);
187                 (void) memcpy(new_content, sbuf->content, cntsize);
188                 free(sbuf->content);
189                 sbuf->content = new_content;
190                 sbuf->end = new_content + cntsize;
191                 sbuf->last = new_content + size - 1;
192         }
193 }
```



```
195 /*
196  * Append a string of a given length to the sbuf.
197  */
198 static void
199 sbuf_append(struct sbuf *sbuf, const char *text, int length)
200 {
201         if (length > 0) {
202                 sbuf_need(sbuf, length);
203                 (void) memcpy(sbuf->end, text, length);
204                 sbuf->end += length;
205         }
206 }

208 /*
209  * Append a null-terminated string to the sbuf.
210  */
211 static void
212 sbuf_append_str(struct sbuf *sbuf, char *text)
213 {

215         sbuf_append(sbuf, text, strlen(text));
216 }

218 /*
219  * Append an edited null-terminated string to the sbuf.
220  */
221 static void
222 sbuf_append_edited(struct sbuf *sbuf, char *text, edited_copy copy)
223 {
224         int     length;

226         if ((length = strlen(text)) > 0) {
227                 sbuf_need(sbuf, length);
228                 sbuf->end = copy(text, sbuf->end, length);
229         }
230 }

232 /*
233  * Strip any of a set of chars from the end of the sbuf.
234  */
235 static void
236 sbuf_strip(struct sbuf *sbuf, const char *set)
237 {

239         while (sbuf->end > sbuf->content && strchr(set, sbuf->end[-1]) != NULL)
240                 sbuf->end--;
241 }

243 /*
244  * Return the null-terminated string built by the sbuf.
245  */
246 static char *
247 sbuf_content(struct sbuf *sbuf)
248 {

250         *sbuf->end = '\0';
251         return (sbuf->content);
252 }

254 /*
255  * Return true if no man page exists in the directory with
256  * any of the names in the stringlist.
257  */
258 static int
259 no_page_exists(char *dir, stringlist *names, char *suffix)
```

```
{
        char    path[MAXPATHLEN];
        size_t  i;

        for (i = 0; i < names->sl_cur; i++) {
                (void) snprintf(path, MAXPATHLEN, "%s/%s.%s.gz",
                    dir, names->sl_str[i], suffix);
                if (access(path, F_OK) < 0) {
                        path[strlen(path) - 3] = '\0';
                        if (access(path, F_OK) < 0)
                                continue;
                }
                return (0);
        }
        return (1);
}

/* ARGSUSED sig */
static void
trap_signal(int sig)
{

        if (tempfile[0] != '\0')
                (void) unlink(tempfile);

        exit(1);
}

/*
 * Attempt to open an output file.
 * Return NULL if unsuccessful.
 */
static FILE *
open_output(char *name)
{
        FILE    *output;

        whatis_lines = sl_init();
        (void) snprintf(tempfile, MAXPATHLEN, "%s.tmp", name);
        name = tempfile;
        if ((output = fopen(name, "w")) == NULL) {
                warn("%s", name);
                return (NULL);
        }
        return (output);
}

static int
linesort(const void *a, const void *b)
{

        return (strcmp((*(const char * const *)a), (*(const char * const *)b)));
}

/*
 * Write the unique sorted lines to the output file.
 */
static void
finish_output(FILE *output, char *name)
{
        size_t  i;
        char    *prev = NULL;

        qsort(whatis_lines->sl_str, whatis_lines->sl_cur, sizeof (char *),
            linesort);
        for (i = 0; i < whatis_lines->sl_cur; i++) {
```



```
                char *line = whatis_lines->sl_str[i];
                if (i > 0 && strcmp(line, prev) == 0)
                        continue;
                prev = line;
                (void) fputs(line, output);
                (void) putc('\n', output);
        }
        (void) fclose(output);
        sl_free(whatis_lines, 1);
        (void) rename(tempfile, name);
        (void) unlink(tempfile);
}

static FILE *
open_whatis(char *mandir)
{
        char    filename[MAXPATHLEN];

        (void) snprintf(filename, MAXPATHLEN, "%s/%s", mandir, WHATIS);
        return (open_output(filename));
}

static void
finish_whatis(FILE *output, char *mandir)
{
        char    filename[MAXPATHLEN];

        (void) snprintf(filename, MAXPATHLEN, "%s/%s", mandir, WHATIS);
        finish_output(output, filename);
}

/*
 * Remove trailing spaces from a string, returning a pointer to just
 * beyond the new last character.
 */
static char *
trim_rhs(char *str)
{
        char    *rhs;

        rhs = &str[strlen(str)];
        while (--rhs > str && isspace(*rhs))
                ;
        *++rhs = '\0';
        return (rhs);
}

/*
 * Return a pointer to the next non-space character in the string.
 */
static char *
skip_spaces(char *s)
{

        while (*s != '\0' && isspace(*s))
                s++;

        return (s);
}

/*
 * Return whether the line is of one of the forms:
 *      .Sh NAME
 *      .Sh "NAME"
 *      etc.
 * assuming that section_start is ".Sh".
```

```
 */
static int
name_section_line(char *line, const char *section_start)
{
        char            *rhs;

        if (strncmp(line, section_start, 3) != 0)
                return (0);
        line = skip_spaces(line + 3);
        rhs = trim_rhs(line);
        if (*line == '"') {
                line++;
                if (*--rhs == '"')
                        *rhs = '\0';
        }
        if (strcmp(line, "NAME") == 0)
                return (1);

        return (0);
}

/*
 * Copy characters while removing the most common nroff/troff markup:
 *      \(em, \(mi, \s[+-N], \&
 *      \fF, \f(fo, \f[font]
 *      \*s, \*(st, \*[stringvar]
 */
static char *
de_nroff_copy(char *from, char *to, int fromlen)
{
        char    *from_end = &from[fromlen];

        while (from < from_end) {
                switch (*from) {
                case '\\':
                        switch (*++from) {
                        case '(':
                                if (strncmp(&from[1], "em", 2) == 0 ||
                                    strncmp(&from[1], "mi", 2) == 0) {
                                        from += 3;
                                        continue;
                                }
                                break;
                        case 's':
                                if (*++from == '-')
                                        from++;
                                while (isdigit(*from))
                                        from++;
                                continue;
                        case 'f':
                        case '*':
                                if (*++from == '(') {
                                        from += 3;
                                } else if (*from == '[') {
                                        while (*++from != ']' &&
                                            from < from_end)
                                                ;
                                        from++;
                                } else {
                                        from++;
                                }
                                continue;
                        case '&':
                                from++;
                                continue;
                        }
```



```
                        break;
                }
                *to++ = *from++;
        }
        return (to);
}

/*
 * Append a string with the nroff formatting removed.
 */
static void
add_nroff(char *text)
{

        sbuf_append_edited(whatis_proto, text, de_nroff_copy);
}

/*
 * Appends "name(suffix), " to whatis_final
 */
static void
add_whatis_name(char *name, char *suffix)
{

        if (*name != '\0') {
                sbuf_append_str(whatis_final, name);
                sbuf_append(whatis_final, "(", 1);
                sbuf_append_str(whatis_final, suffix);
                sbuf_append(whatis_final, "), ", 3);
        }
}

/*
 * Processes an old-style man(7) line. This ignores commands with only
 * a single number argument.
 */
static void
process_man_line(char *line)
{
        char    *p;

        if (*line == '.') {
                while (isalpha(*++line))
                        ;
                p = line = skip_spaces(line);
                while (*p != '\0') {
                        if (!isdigit(*p))
                                break;
                        p++;
                }
                if (*p == '\0')
                        return;
        } else
                line = skip_spaces(line);
        if (*line != '\0') {
                add_nroff(line);
                sbuf_append(whatis_proto, " ", 1);
        }
}

/*
 * Processes a new-style mdoc(7) line.
 */
static void
process_mdoc_line(char *line)
{
```

```
524         int     xref;
525         int     arg = 0;
526         char    *line_end = &line[strlen(line)];
527         int     orig_length = sbuf_length(whatis_proto);
528         char    *next;

530         if (*line == '\0')
531                 return;
532         if (line[0] != '.' || !isupper(line[1]) || !islower(line[2])) {
533                 add_nroff(skip_spaces(line));
534                 sbuf_append(whatis_proto, " ", 1);
535                 return;
536         }
537         xref = strncmp(line, ".Xr", 3) == 0;
538         line += 3;
539         while ((line = skip_spaces(line)) < line_end) {
540                 if (*line == '"') {
541                         next = ++line;
542                         for (;;) {
543                                 next = strchr(next, '"');
544                                 if (next == NULL)
545                                         break;
546                                 (void) memmove(next, next + 1, strlen(next));
547                                 line_end--;
548                                 if (*next != '"')
549                                         break;
550                                 next++;
551                         }
552                 } else {
553                         next = strpbrk(line, " \t");
554                 }
555                 if (next != NULL)
556                         *next++ = '\0';
557                 else
558                         next = line_end;
559                 if (isupper(*line) && islower(line[1]) && line[2] == '\0') {
560                         if (strcmp(line, "Ns") == 0) {
561                                 arg = 0;
562                                 line = next;
563                                 continue;
564                         }
565                         if (strstr(line, MDOC_COMMANDS) != NULL) {
566                                 line = next;
567                                 continue;
568                         }
569                 }
570                 if (arg > 0 && strchr(",.:;?!)]", *line) == 0) {
571                         if (xref) {
572                                 sbuf_append(whatis_proto, "(", 1);
573                                 add_nroff(line);
574                                 sbuf_append(whatis_proto, ")", 1);
575                                 xref = 0;
576                         } else {
577                                 sbuf_append(whatis_proto, " ", 1);
578                         }
579                 }
580                 add_nroff(line);
581                 arg++;
582                 line = next;
583         }
584         if (sbuf_length(whatis_proto) > orig_length)
585                 sbuf_append(whatis_proto, " ", 1);
586 }

588 /*
589  * Collect a list of comma-separated names from the text.
```



```
590  */
591 static void
592 collect_names(stringlist *names, char *text)
593 {
594         char    *arg;

596         for (;;) {
597                 arg = text;
598                 text = strchr(text, ',');
599                 if (text != NULL)
600                         *text++ = '\0';
601                 (void) sl_add(names, arg);
602                 if (text == NULL)
603                         return;
604                 if (*text == ' ')
605                         text++;
606         }
607 }

609 enum { STATE_UNKNOWN, STATE_MANSTYLE, STATE_MDOCNAME, STATE_MDOCDESC };

611 /*
612  * Process a man page source into a single whatis line and add it
613  * to whatis_lines.
614  */
615 static void
616 process_page(struct page_info *page, char *section_dir)
617 {
618         FILE            *fp;
619         stringlist      *names;
620         char            *descr;
621         int             state = STATE_UNKNOWN;
622         size_t          i;
623         char            *line = NULL;
624         size_t          linecap = 0;

626         sbuf_clear(whatis_proto);
627         if ((fp = fopen(page->filename, "r")) == NULL) {
628                 warn("%s", page->filename);
629                 return;
630         }
631         while (getline(&line, &linecap, fp) > 0) {
632                 /* Skip comments */
633                 if (strncmp(line, ".\\\"", 3) == 0)
634                         continue;
635                 switch (state) {
636                 /* Haven't reached the NAME section yet */
637                 case STATE_UNKNOWN:
638                         if (name_section_line(line, ".SH"))
639                                 state = STATE_MANSTYLE;
640                         else if (name_section_line(line, ".Sh"))
641                                 state = STATE_MDOCNAME;
642                         continue;
643                 /* Inside an old-style .SH NAME section */
644                 case STATE_MANSTYLE:
645                         if (strncmp(line, ".SH", 3) == 0 ||
646                             strncmp(line, ".SS", 3) == 0)
647                                 break;
648                         (void) trim_rhs(line);
649                         if (strcmp(line, ".") == 0)
650                                 continue;
651                         if (strncmp(line, ".IX", 3) == 0) {
652                                 line += 3;
653                                 line = skip_spaces(line);
654                         }
655                         process_man_line(line);
```

```
656                     continue;
657             /* Inside a new-style .Sh NAME section (the .Nm part) */
658             case STATE_MDOCNAME:
659                     (void) trim_rhs(line);
660                     if (strncmp(line, ".Nm", 3) == 0) {
661                             process_mdoc_line(line);
662                             continue;
663                     } else {
664                             if (strcmp(line, ".") == 0)
665                                     continue;
666                             sbuf_append(whatis_proto, "- ", 2);
667                             state = STATE_MDOCDESC;
668                     }
669                     /* FALLTHROUGH */
670             /* Inside a new-style .Sh NAME section (after the .Nm-s) */
671             case STATE_MDOCDESC:
672                     if (strncmp(line, ".Sh", 3) == 0)
673                             break;
674                     (void) trim_rhs(line);
675                     if (strcmp(line, ".") == 0)
676                             continue;
677                     process_mdoc_line(line);
678                     continue;
679             }
680             break;
681     }
682     (void) fclose(fp);
683     sbuf_strip(whatis_proto, " \t.-");
684     line = sbuf_content(whatis_proto);
685     /*
686      * Line now contains the appropriate data, but without the
687      * proper indentation or the section appended to each name.
688      */
689     descr = strstr(line, " - ");
690     if (descr == NULL) {
691             descr = strchr(line, ' ');
692             if (descr == NULL)
693                     return;
694             *descr++ = '\0';
695     } else {
696             *descr = '\0';
697             descr += 3;
698     }
699     names = sl_init();
700     collect_names(names, line);
701     sbuf_clear(whatis_final);
702     if (!sl_find(names, page->name) &&
703         no_page_exists(section_dir, names, page->suffix)) {
704             /*
705              * Add the page name since that's the only
706              * thing that man(1) will find.
707              */
708             add_whatis_name(page->name, page->suffix);
709     }
710     for (i = 0; i < names->sl_cur; i++)
711             add_whatis_name(names->sl_str[i], page->suffix);
712     sl_free(names, 0);
713     /* Remove last ", " */
714     sbuf_retract(whatis_final, 2);
715     while (sbuf_length(whatis_final) < INDENT)
716             sbuf_append(whatis_final, " ", 1);
717     sbuf_append(whatis_final, " - ", 3);
718     sbuf_append_str(whatis_final, skip_spaces(descr));
719     (void) sl_add(whatis_lines, strdup(sbuf_content(whatis_final)));
720 }
```



```
722 /*
723  * Sort pages first by inode number, then by name.
724  */
725 static int
726 pagesort(const void *a, const void *b)
727 {
728         const struct page_info *p1 = *(struct page_info * const *) a;
729         const struct page_info *p2 = *(struct page_info * const *) b;
730
731         if (p1->inode == p2->inode)
732                 return (strcmp(p1->name, p2->name));
733
734         return (p1->inode - p2->inode);
735 }
736
737 /*
738  * Process a single man section.
739  */
740 static void
741 process_section(char *section_dir)
742 {
743         struct dirent   **entries;
744         int             nentries;
745         struct page_info **pages;
746         int             npages = 0;
747         int             i;
748         ino_t           prev_inode = 0;
749
750         /* Scan the man section directory for pages */
751         nentries = scandir(section_dir, &entries, NULL, alphasort);
752
753         /* Collect information about man pages */
754         pages = (struct page_info **)calloc(nentries,
755             sizeof (struct page_info *));
756         for (i = 0; i < nentries; i++) {
757                 struct page_info *info = new_page_info(section_dir, entries[i]);
758                 if (info != NULL)
759                         pages[npages++] = info;
760                 free(entries[i]);
761         }
762         free(entries);
763         qsort(pages, npages, sizeof (struct page_info *), pagesort);
764
765         /* Process each unique page */
766         for (i = 0; i < npages; i++) {
767                 struct page_info *page = pages[i];
768                 if (page->inode != prev_inode) {
769                         prev_inode = page->inode;
770                         process_page(page, section_dir);
771                 }
772                 free_page_info(page);
773         }
774         free(pages);
775 }
776
777 /*
778  * Return whether the directory entry is a man page section.
779  */
780 static int
781 select_sections(const struct dirent *entry)
782 {
783         const char      *p = &entry->d_name[3];
784
785         if (strncmp(entry->d_name, "man", 3) != 0)
786                 return (0);
787         while (*p != '\0') {
```

```
788                 if (!isalnum(*p++))
789                         return (0);
790         }
791         return (1);
792 }

794 /*
795  * Process a single top-level man directory by finding all the
796  * sub-directories named man* and processing each one in turn.
797  */
798 void
799 mwpath(char *path)
800 {
801         FILE            *fp = NULL;
802         struct dirent   **entries;
803         int             nsections;
804         int             i;

806         (void) signal(SIGINT, trap_signal);
807         (void) signal(SIGHUP, trap_signal);
808         (void) signal(SIGQUIT, trap_signal);
809         (void) signal(SIGTERM, trap_signal);

811         whatis_proto = new_sbuf();
812         whatis_final = new_sbuf();

814         nsections = scandir(path, &entries, select_sections, alphasort);
815         if ((fp = open_whatis(path)) == NULL)
816                 return;
817         for (i = 0; i < nsections; i++) {
818                 char    section_dir[MAXPATHLEN];

820                 (void) snprintf(section_dir, MAXPATHLEN, "%s/%s",
821                     path, entries[i]->d_name);
822                 process_section(section_dir);
823                 free(entries[i]);
824         }
825         free(entries);
826         finish_whatis(fp, path);
827 }
```

```
**********************************************************
   33002 Tue Jul 15 13:48:03 2014
new/usr/src/cmd/man/man.c
Finish integration.  Use mandoc_preconv, etc.
import complete (hopefully)
make it compile
Initial import of man functionality.
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1990, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012, Josef 'Jeff' Sipek <jeffpc@31bits.net>. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989  AT&T.   */
/*              All rights reserved.                                    */

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * Find and display reference manual pages. This version includes makewhatis
 * functionality as well.
 */

#include <sys/param.h>
#include <sys/stat.h>
#include <sys/termios.h>
#include <sys/types.h>

#include <ctype.h>
#include <dirent.h>
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <fnmatch.h>
#include <limits.h>
#include <locale.h>
```



```
#include <malloc.h>
#include <memory.h>
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "man.h"


/* Mapping of old directories to new directories */
static const struct map_entry {
        char    *old_name;
        char    *new_name;
} map[] = {
        { "3b",         "3ucb"          },
        { "3e",         "3elf"          },
        { "3g",         "3gen"          },
        { "3k",         "3kstat"        },
        { "3n",         "3socket"       },
        { "3r",         "3rt"           },
        { "3s",         "3c"            },
        { "3t",         "3thr"          },
        { "3x",         "3curses"       },
        { "3xc",        "3xcurses"      },
        { "3xn",        "3xnet"         }
};

struct suffix {
        char *ds;
        char *fs;
};

/*
 * Flags that control behavior of build_manpath()
 *
 *   BMP_ISPATH                 pathv is a vector constructed from PATH.
 *                              Perform appropriate path translations for
 *                              manpath.
 *   BMP_APPEND_DEFMANDIR       Add DEFMANDIR to the end if it hasn't
 *                              already appeared earlier.
 *   BMP_FALLBACK_DEFMANDIR     Append /usr/share/man only if no other
 *                              manpath (including derived from PATH)
 *                              elements are valid.
 */
#define BMP_ISPATH              1
#define BMP_APPEND_DEFMANDIR    2
#define BMP_FALLBACK_DEFMANDIR  4

/*
 * When doing equality comparisons of directories, device and inode
 * comparisons are done.  The secnode and dupnode structures are used
 * to form a list of lists for this processing.
 */
struct secnode {
        char            *secp;
        struct secnode  *next;
};
struct dupnode {
        dev_t           dev;    /* from struct stat st_dev */
        ino_t           ino;    /* from struct stat st_ino */
        struct secnode  *secl;  /* sections already considered */
        struct dupnode  *next;
};
```

```
/*
 * Map directories that may appear in PATH to the corresponding
 * man directory.
 */
static struct pathmap {
        char    *bindir;
        char    *mandir;
        dev_t   dev;
        ino_t   ino;
} bintoman[] = {
        { "/sbin",              "/usr/share/man,1m",            0, 0 },
        { "/usr/sbin",          "/usr/share/man,1m",            0, 0 },
        { "/usr/ucb",           "/usr/share/man,1b",            0, 0 },
        { "/usr/bin",           "/usr/share/man,1,1m,1s,1t,1c", 0, 0 },
        { "/usr/xpg4/bin",      "/usr/share/man,1",             0, 0 },
        { "/usr/xpg6/bin",      "/usr/share/man,1",             0, 0 },
        { NULL,                 NULL,                           0, 0 }
};

struct man_node {
        char            *path;          /* mandir path */
        char            **secv;         /* submandir suffices */
        int             defsrch;        /* hint for man -p */
        int             frompath;       /* hint for man -d */
        struct man_node *next;
};

static int      all = 0;
static int      apropos = 0;
static int      debug = 0;
static int      found = 0;
static int      list = 0;
static int      makewhatis = 0;
static int      printmp = 0;
static int      sargs = 0;
static int      psoutput = 0;
static int      whatis = 0;

static char     *mansec;
static char     *pager;

static char     *addlocale(char *);
static struct man_node *build_manpath(char **, int);
static void     do_makewhatis(struct man_node *);
static char     *check_config(char *);
static int      cmp(const void *, const void *);
static int      dupcheck(struct man_node *, struct dupnode **);
static int      format(char *, char *, char *, char *);
static void     free_dupnode(struct dupnode *);
static void     free_manp(struct man_node *manp);
static void     freev(char **);
static void     fullpaths(struct man_node **);
static void     get_all_sect(struct man_node *);
static int      getdirs(char *, char ***, int);
static void     getpath(struct man_node *, char **);
static void     getsect(struct man_node *, char **);
static void     init_bintoman(void);
static void     lower(char *);
static void     mandir(char **, char *, char *, int);
static int      manual(struct man_node *, char *);
static char     *map_section(char *, char *);
static char     *path_to_manpath(char *);
static void     print_manpath(struct man_node *);
static void     search_whatis(char *, char *);
static int      searchdir(char *, char *, char *);
static void     sortdir(DIR *, char ***);
```



```
static char     **split(char *, char);
static void     usage_man(void);
static void     usage_whatapro(void);
static void     whatapro(struct man_node *, char *);

static char     language[MAXPATHLEN];   /* LC_MESSAGES */
static char     localedir[MAXPATHLEN];  /* locale specific path component */

static char     *newsection = NULL;

static int      manwidth = 0;

extern const char       *__progname;

int
main(int argc, char **argv)
{
        int             c, i;
        char            **pathv;
        char            *manpath = NULL;
        static struct man_node *mandirs = NULL;
        int             bmp_flags = 0;
        int             ret = 0;
        char            *opts;
        char            *mwstr;

        (void) setlocale(LC_ALL, "");
        (void) strcpy(language, setlocale(LC_MESSAGES, (char *)NULL));
        if (strcmp("C", language) != 0)
                (void) strlcpy(localedir, language, MAXPATHLEN);

#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        if (strcmp(__progname, "apropos") == 0) {
                apropos++;
                opts = "M:ds:";
        } else if (strcmp(__progname, "whatis") == 0) {
                apropos++;
                whatis++;
                opts = "M:ds:";
        } else {
                opts = "M:adfklps:tw";
        }

        opterr = 0;
        while ((c = getopt(argc, argv, opts)) != -1) {
                switch (c) {
                case 'M':       /* Respecify path for man pages */
                        manpath = optarg;
                        break;
                case 'a':
                        all++;
                        break;
                case 'd':
                        debug++;
                        break;
                case 'f':
                        whatis++;
                        /*FALLTHROUGH*/
                case 'k':
                        apropos++;
                        break;
                case 'l':
```

```
257                         list++;
258                         /*FALLTHROUGH*/
259                 case 'p':
260                         printmp++;
261                         break;
262                 case 's':
263                         mansec = optarg;
264                         sargs++;
265                         break;
266                 case 't':
267                         psoutput++;
268                         break;
269                 case 'w':
270                         makewhatis++;
271                         break;
272                 case '?':
273                 default:
274                         if (apropos)
275                                 usage_whatapro();
276                         else
277                                 usage_man();
278                 }
279         }
280         argc -= optind;
281         argv += optind;

283         if (argc == 0) {
284                 if (apropos) {
285                         (void) fprintf(stderr, gettext("%s what?\n"),
286                             __progname);
287                         exit(1);
288                 } else if (!printmp && !makewhatis) {
289                         (void) fprintf(stderr,
290                             gettext("What manual page do you want?\n"));
291                         exit(1);
292                 }
293         }

295         init_bintoman();
296         if (manpath == NULL && (manpath = getenv("MANPATH")) == NULL) {
297                 if ((manpath = getenv("PATH")) != NULL)
298                         bmp_flags = BMP_ISPATH | BMP_APPEND_DEFMANDIR;
299                 else
300                         manpath = DEFMANDIR;
301         }
302         pathv = split(manpath, ':');
303         mandirs = build_manpath(pathv, bmp_flags);
304         freev(pathv);
305         fullpaths(&mandirs);

307         if (makewhatis) {
308                 do_makewhatis(mandirs);
309                 exit(0);
310         }

312         if (printmp) {
313                 print_manpath(mandirs);
314                 exit(0);
315         }

317         /* Collect environment information */
318         if (isatty(STDOUT_FILENO) && (mwstr = getenv("MANWIDTH")) != NULL &&
319             *mwstr != '\0') {
320                 if (strcasecmp(mwstr, "tty") == 0) {
321                         struct winsize  ws;
```



```
323                         if (ioctl(0, TIOCGWINSZ, &ws) != 0)
324                                 warn("TIOCGWINSZ");
325                         else
326                                 manwidth = ws.ws_col;
327                 } else {
328                         manwidth = (int)strtol(mwstr, (char **)NULL, 10);
329                         if (manwidth < 0)
330                                 manwidth = 0;
331                 }
332         }
333         if (manwidth != 0) {
334                 DPRINTF("-- Using non-standard page width: %d\n", manwidth);
335         }

337         if ((pager = getenv("PAGER")) == NULL || *pager == '\0')
338                 pager = PAGER;
339         DPRINTF("-- Using pager: %s\n", pager);

341         for (i = 0; i < argc; i++) {
342                 char            *cmd;
343                 static struct man_node *mp;
344                 char            *pv[2];

346                 /*
347                  * If full path to command specified, customize
348                  * the manpath accordingly.
349                  */
350                 if ((cmd = strrchr(argv[i], '/')) != NULL) {
351                         *cmd = '\0';
352                         if ((pv[0] = strdup(argv[i])) == NULL)
353                                 err(1, "strdup");
354                         pv[1] = NULL;
355                         *cmd = '/';
356                         mp = build_manpath(pv,
357                             BMP_ISPATH | BMP_FALLBACK_DEFMANDIR);
358                 } else {
359                         mp = mandirs;
360                 }

362                 if (apropos)
363                         whatapro(mp, argv[i]);
364                 else
365                         ret += manual(mp, argv[i]);

367                 if (mp != NULL && mp != mandirs) {
368                         free(pv[0]);
369                         free_manp(mp);
370                 }
371         }

373         return (ret == 0 ? 0 : 1);
374 }

376 /*
377  * This routine builds the manpage structure from MANPATH or PATH,
378  * depending on flags.  See BMP_* definitions above for valid
379  * flags.
380  */
381 static struct man_node *
382 build_manpath(char **pathv, int flags)
383 {
384         struct man_node *manpage = NULL;
385         struct man_node *currp = NULL;
386         struct man_node *lastp = NULL;
387         char            **p;
388         char            **q;
```

```
389         char            *mand = NULL;
390         char            *mandir = DEFMANDIR;
391         int             s;
392         struct dupnode  *didup = NULL;
393         struct stat     sb;

395         s = sizeof (struct man_node);
396         for (p = pathv; *p != NULL; ) {
397                 if (flags & BMP_ISPATH) {
398                         if ((mand = path_to_manpath(*p)) == NULL)
399                                 goto next;
400                         free(*p);
401                         *p = mand;
402                 }
403                 q = split(*p, ',');
404                 if (stat(q[0], &sb) != 0 || (sb.st_mode & S_IFDIR) == 0) {
405                         freev(q);
406                         goto next;
407                 }

409                 if (access(q[0], R_OK | X_OK) == 0) {
410                         /*
411                          * Some element exists.  Do not append DEFMANDIR as a
412                          * fallback.
413                          */
414                         flags &= ~BMP_FALLBACK_DEFMANDIR;

416                         if ((currp = (struct man_node *)calloc(1, s)) == NULL)
417                                 err(1, "calloc");

419                         currp->frompath = (flags & BMP_ISPATH);

421                         if (manpage == NULL)
422                                 lastp = manpage = currp;

424                         getpath(currp, p);
425                         getsect(currp, p);

427                         /*
428                          * If there are no new elements in this path,
429                          * do not add it to the manpage list.
430                          */
431                         if (dupcheck(currp, &didup) != 0) {
432                                 freev(currp->secv);
433                                 free(currp);
434                         } else {
435                                 currp->next = NULL;
436                                 if (currp != manpage)
437                                         lastp->next = currp;
438                                 lastp = currp;
439                         }
440                 }
441                 freev(q);
442 next:
443                 /*
444                  * Special handling of appending DEFMANDIR. After all pathv
445                  * elements have been processed, append DEFMANDIR if needed.
446                  */
447                 if (p == &mandir)
448                         break;
449                 p++;
450                 if (*p != NULL)
451                         continue;
452                 if (flags & (BMP_APPEND_DEFMANDIR | BMP_FALLBACK_DEFMANDIR)) {
453                         p = &mandir;
454                         flags &= ~BMP_ISPATH;
```



```
455                 }
456         }

458         free_dupnode(didup);

460         return (manpage);
461 }

463 /*
464  * Store the mandir path into the manp structure.
465  */
466 static void
467 getpath(struct man_node *manp, char **pv)
468 {
469         char    *s = *pv;
470         int     i = 0;

472         while (*s != '\0' && *s != ',')
473                 i++, s++;

475         if ((manp->path = (char *)malloc(i + 1)) == NULL)
476                 err(1, "malloc");
477         (void) strlcpy(manp->path, *pv, i + 1);
478 }

480 /*
481  * Store the mandir's corresponding sections (submandir
482  * directories) into the manp structure.
483  */
484 static void
485 getsect(struct man_node *manp, char **pv)
486 {
487         char    *sections;
488         char    **sectp;

490         /* Just store all sections when doing makewhatis or apropos/whatis */
491         if (makewhatis || apropos) {
492                 manp->defsrch = 1;
493                 DPRINTF("-- Adding %s\n", manp->path);
494                 manp->secv = NULL;
495                 get_all_sect(manp);
496         } else if (sargs) {
497                 manp->secv = split(mansec, ',');
498                 for (sectp = manp->secv; *sectp; sectp++)
499                         lower(*sectp);
500         } else if ((sections = strchr(*pv, ',')) != NULL) {
501                 DPRINTF("-- Adding %s: MANSECTS=%s\n", manp->path, sections);
502                 manp->secv = split(++sections, ',');
503                 for (sectp = manp->secv; *sectp; sectp++)
504                         lower(*sectp);
505                 if (*manp->secv == NULL)
506                         get_all_sect(manp);
507         } else if ((sections = check_config(*pv)) != NULL) {
508                 manp->defsrch = 1;
509                 DPRINTF("-- Adding %s: from %s, MANSECTS=%s\n", manp->path,
510                     CONFIG, sections);
511                 manp->secv = split(sections, ',');
512                 for (sectp = manp->secv; *sectp; sectp++)
513                         lower(*sectp);
514                 if (*manp->secv == NULL)
515                         get_all_sect(manp);
516         } else {
517                 manp->defsrch = 1;
518                 DPRINTF("-- Adding %s: default sort order\n", manp->path);
519                 manp->secv = NULL;
520                 get_all_sect(manp);
```

```
521         }
522 }

524 /*
525  * Get suffices of all sub-mandir directories in a mandir.
526  */
527 static void
528 get_all_sect(struct man_node *manp)
529 {
530         DIR     *dp;
531         char    **dirv;
532         char    **dv;
533         char    **p;
534         char    *prev = NULL;
535         char    *tmp = NULL;
536         int     maxentries = MAXTOKENS;
537         int     entries = 0;

539         if ((dp = opendir(manp->path)) == 0)
540                 return;

542         sortdir(dp, &dirv);

544         (void) closedir(dp);

546         if (manp->secv == NULL) {
547                 if ((manp->secv = malloc(maxentries * sizeof (char *))) == NULL)
548                         err(1, "malloc");
549         }

551         for (dv = dirv, p = manp->secv; *dv; dv++) {
552                 if (strcmp(*dv, CONFIG) == 0) {
553                         free(*dv);
554                         continue;
555                 }

557                 free(tmp);
558                 if ((tmp = strdup(*dv + 3)) == NULL)
559                         err(1, "strdup");

561                 if (prev != NULL && strcmp(prev, tmp) == 0) {
562                         free(*dv);
563                         continue;
564                 }

566                 free(prev);
567                 if ((prev = strdup(*dv + 3)) == NULL)
568                         err(1, "strdup");

570                 if ((*p = strdup(*dv + 3)) == NULL)
571                         err(1, "strdup");

573                 p++; entries++;

575                 if (entries == maxentries) {
576                         maxentries += MAXTOKENS;
577                         if ((manp->secv = realloc(manp->secv,
578                             sizeof (char *) * maxentries)) == NULL)
579                                 err(1, "realloc");
580                         p = manp->secv + entries;
581                 }
582                 free(*dv);
583         }
584         free(tmp);
585         free(prev);
586         *p = NULL;
```



```
587         free(dirv);
588 }

590 /*
591  * Build whatis databases.
592  */
593 static void
594 do_makewhatis(struct man_node *manp)
595 {
596         struct man_node *p;
597         char            *ldir;

599         for (p = manp; p != NULL; p = p->next) {
600                 ldir = addlocale(p->path);
601                 if (*localedir != '\0' && getdirs(ldir, NULL, 0) > 0)
602                         mwpath(ldir);
603                 free(ldir);
604                 mwpath(p->path);
605         }
606 }

608 /*
609  * Count mandirs under the given manpath
610  */
611 static int
612 getdirs(char *path, char ***dirv, int flag)
613 {
614         DIR             *dp;
615         struct dirent   *d;
616         int             n = 0;
617         int             maxentries = MAXDIRS;
618         char            **dv = NULL;

620         if ((dp = opendir(path)) == NULL)
621                 return (0);

623         if (flag) {
624                 if ((*dirv = malloc(sizeof (char *) *
625                     maxentries)) == NULL)
626                         err(1, "malloc");
627                 dv = *dirv;
628         }
629         while ((d = readdir(dp))) {
630                 if (strncmp(d->d_name, "man", 3) != 0)
631                         continue;
632                 n++;

634                 if (flag) {
635                         if ((*dv = strdup(d->d_name + 3)) == NULL)
636                                 err(1, "strdup");
637                         dv++;
638                         if ((dv - *dirv) == maxentries) {
639                                 int     entries = maxentries;

641                                 maxentries += MAXTOKENS;
642                                 if ((*dirv = realloc(*dirv,
643                                     sizeof (char *) * maxentries)) == NULL)
644                                         err(1, "realloc");
645                                 dv = *dirv + entries;
646                         }
647                 }
648         }

650         (void) closedir(dp);
651         return (n);
652 }
```

```
655 /*
656  * Find matching whatis or apropos entries.
657  */
658 static void
659 whatapro(struct man_node *manp, char *word)
660 {
661         char            whatpath[MAXPATHLEN];
662         struct man_node *b;
663         char            *ldir;

665         for (b = manp; b != NULL; b = b->next) {
666                 if (*localedir != '\0') {
667                         ldir = addlocale(b->path);
668                         if (getdirs(ldir, NULL, 0) != 0) {
669                                 (void) snprintf(whatpath, sizeof (whatpath),
670                                     "%s/%s", ldir, WHATIS);
671                                 search_whatis(whatpath, word);
672                         }
673                         free(ldir);
674                 }
675                 (void) snprintf(whatpath, sizeof (whatpath), "%s/%s", b->path,
676                     WHATIS);
677                 search_whatis(whatpath, word);
678         }
679 }

681 static void
682 search_whatis(char *whatpath, char *word)
683 {
684         FILE            *fp;
685         char            *line = NULL;
686         size_t          linecap = 0;
687         char            *pkwd;
688         regex_t         preg;
689         char            **ss = NULL;
690         char            s[MAXNAMELEN];
691         int             i;

693         if ((fp = fopen(whatpath, "r")) == NULL)
694                 return;

696         DPRINTF("-- Found %s: %s\n", WHATIS, whatpath);

698         /* Build keyword regex */
699         if (asprintf(&pkwd, "%s%s%s", (whatis) ? "\\<" : "",
700             word, (whatis) ? "\\>" : "") == -1)
701                 err(1, "asprintf");

703         if (regcomp(&preg, pkwd, REG_BASIC | REG_ICASE | REG_NOSUB) != 0)
704                 err(1, "regcomp");

706         if (sargs)
707                 ss = split(mansec, ',');

709         while (getline(&line, &linecap, fp) > 0) {
710                 if (regexec(&preg, line, 0, NULL, 0) == 0) {
711                         if (sargs) {
712                                 /* Section-restricted search */
713                                 for (i = 0; ss[i] != NULL; i++) {
714                                         (void) snprintf(s, sizeof (s), "(%s)",
715                                             ss[i]);
716                                         if (strstr(line, s) != NULL) {
717                                                 (void) printf("%s", line);
718                                                 break;
```



```
719                                         }
720                                 }
721                         } else {
722                                 (void) printf("%s", line);
723                         }
724                 }
725         }

727         if (ss != NULL)
728                 freev(ss);
729         free(pkwd);
730         (void) fclose(fp);
731 }


734 /*
735  * Split a string by specified separator.
736  */
737 static char **
738 split(char *s1, char sep)
739 {
740         char    **tokv, **vp;
741         char    *mp = s1, *tp;
742         int     maxentries = MAXTOKENS;
743         int     entries = 0;

745         if ((tokv = vp = malloc(maxentries * sizeof (char *))) == NULL)
746                 err(1, "malloc");

748         for (; mp && *mp; mp = tp) {
749                 tp = strchr(mp, sep);
750                 if (mp == tp) {
751                         tp++;
752                         continue;
753                 }
754                 if (tp) {
755                         size_t  len;

757                         len = tp - mp;
758                         if ((*vp = (char *)malloc(sizeof (char) *
759                             len + 1)) == NULL)
760                                 err(1, "malloc");
761                         (void) strncpy(*vp, mp, len);
762                         *(*vp + len) = '\0';
763                         tp++;
764                         vp++;
765                 } else {
766                         if ((*vp = strdup(mp)) == NULL)
767                                 err(1, "strdup");
768                         vp++;
769                 }
770                 entries++;
771                 if (entries == maxentries) {
772                         maxentries += MAXTOKENS;
773                         if ((tokv = realloc(tokv,
774                             maxentries * sizeof (char *))) == NULL)
775                                 err(1, "realloc");
776                         vp = tokv + entries;
777                 }
778         }
779         *vp = 0;

781         return (tokv);
782 }

784 /*
```

```
 * Free a vector allocated by split()
 */
static void
freev(char **v)
{
        int i;
        if (v != NULL) {
                for (i = 0; v[i] != NULL; i++) {
                        free(v[i]);
                }
                free(v);
        }
}

/*
 * Convert paths to full paths if necessary
 */
static void
fullpaths(struct man_node **manp_head)
{
        char            *cwd = NULL;
        char            *p;
        int             cwd_gotten = 0;
        struct man_node *manp = *manp_head;
        struct man_node *b;
        struct man_node *prev = NULL;

        for (b = manp; b != NULL; b = b->next) {
                if (*(b->path) == '/') {
                        prev = b;
                        continue;
                }

                if (!cwd_gotten) {
                        cwd = getcwd(NULL, MAXPATHLEN);
                        cwd_gotten = 1;
                }

                if (cwd) {
                        /* Relative manpath with cwd: make absolute */
                        if (asprintf(&p, "%s/%s", cwd, b->path) == -1)
                                err(1, "asprintf");
                        free(b->path);
                        b->path = p;
                } else {
                        /* Relative manpath but no cwd: omit path entry */
                        if (prev)
                                prev->next = b->next;
                        else
                                *manp_head = b->next;

                        free_manp(b);
                }
        }
        free(cwd);
}

/*
 * Free a man_node structure and its contents
 */
static void
free_manp(struct man_node *manp)
{
        char    **p;

        free(manp->path);
```



```
        p = manp->secv;
        while ((p != NULL) && (*p != NULL)) {
                free(*p);
                p++;
        }
        free(manp->secv);
        free(manp);
}


/*
 * Map (in place) to lower case.
 */
static void
lower(char *s)
{

        if (s == 0)
                return;
        while (*s) {
                if (isupper(*s))
                        *s = tolower(*s);
                s++;
        }
}


/*
 * Compare function for qsort().
 * Sort first by section, then by prefix.
 */
static int
cmp(const void *arg1, const void *arg2)
{
        int     n;
        char    **p1 = (char **)arg1;
        char    **p2 = (char **)arg2;

        /* By section */
        if ((n = strcmp(*p1 + 3, *p2 + 3)) != 0)
                return (n);

        /* By prefix reversed */
        return (strncmp(*p2, *p1, 3));
}


/*
 * Find a manpage.
 */
static int
manual(struct man_node *manp, char *name)
{
        struct man_node *p;
        struct man_node *local;
        int             ndirs = 0;
        char            *ldir;
        char            *ldirs[2];
        char            *fullname = name;
        char            *slash;

        if ((slash = strrchr(name, '/')) != NULL)
                name = slash + 1;

        /* For each path in MANPATH */
        found = 0;
```

```
        for (p = manp; p != NULL; p = p->next) {
                DPRINTF("-- Searching mandir: %s\n", p->path);

                if (*localedir != '\0') {
                        ldir = addlocale(p->path);
                        ndirs = getdirs(ldir, NULL, 0);
                        if (ndirs != 0) {
                                ldirs[0] = ldir;
                                ldirs[1] = NULL;
                                local = build_manpath(ldirs, 0);
                                DPRINTF("-- Locale specific subdir: %s\n",
                                    ldir);
                                mandir(local->secv, ldir, name, 1);
                                free_manp(local);
                        }
                        free(ldir);
                }

                /*
                 * Locale mandir not valid, man page in locale
                 * mandir not found, or -a option present
                 */
                if (ndirs == 0 || !found || all)
                        mandir(p->secv, p->path, name, 0);

                if (found && !all)
                        break;
        }

        if (!found) {
                if (sargs) {
                        (void) fprintf(stderr, gettext(
                            "No manual entry for %s in section(s) %s\n"),
                            fullname, mansec);
                } else {
                        (void) fprintf(stderr,
                            gettext("No manual entry for %s\n"), fullname);
                }

        }

        return (!found);
}


/*
 * For a specified manual directory, read, store and sort section subdirs.
 * For each section specified, find and search matching subdirs.
 */
static void
mandir(char **secv, char *path, char *name, int lspec)
{
        DIR     *dp;
        char    **dirv;
        char    **dv, **pdv;
        int     len, dslen;

        if ((dp = opendir(path)) == NULL)
                return;

        if (lspec)
                DPRINTF("-- Searching mandir: %s\n", path);

        sortdir(dp, &dirv);
```



```
        /* Search in the order specified by MANSECTS */
        for (; *secv; secv++) {
                len = strlen(*secv);
                for (dv = dirv; *dv; dv++) {
                        dslen = strlen(*dv + 3);
                        if (dslen > len)
                                len = dslen;
                        if (**secv == '\\') {
                                if (strcmp(*secv + 1, *dv + 3) != 0)
                                        continue;
                        } else if (strncasecmp(*secv, *dv + 3, len) != 0) {
                                if (!all &&
                                    (newsection = map_section(*secv, path))
                                    == NULL) {
                                        continue;
                                }
                                if (newsection == NULL)
                                        newsection = "";
                                if (strncmp(newsection, *dv + 3, len) != 0) {
                                        continue;
                                }
                        }

                        if (searchdir(path, *dv, name) == 0)
                                continue;

                        if (!all) {
                                pdv = dirv;
                                while (*pdv) {
                                        free(*pdv);
                                        pdv++;
                                }
                                (void) closedir(dp);
                                free(dirv);
                                return;
                        }

                        if (all && **dv == 'm' && *(dv + 1) &&
                            strcmp(*(dv + 1) + 3, *dv + 3) == 0)
                                        dv++;
                }
        }
        pdv = dirv;
        while (*pdv != NULL) {
                free(*pdv);
                pdv++;
        }
        free(dirv);
        (void) closedir(dp);
}

/*
 * Sort directories.
 */
static void
sortdir(DIR *dp, char ***dirv)
{
        struct dirent   *d;
        char            **dv;
        int             maxentries = MAXDIRS;
        int             entries = 0;

        if ((dv = *dirv = malloc(sizeof (char *) *
            maxentries)) == NULL)
                err(1, "malloc");
        dv = *dirv;
```

```
1050         while ((d = readdir(dp))) {
1051                 if (strcmp(d->d_name, ".") == 0 ||
1052                     strcmp(d->d_name, "..") == 0)
1053                         continue;

1055                 if (strncmp(d->d_name, "man", 3) == 0 ||
1056                     strncmp(d->d_name, "cat", 3) == 0) {
1057                         if ((*dv = strdup(d->d_name)) == NULL)
1058                                 err(1, "strdup");
1059                         dv++;
1060                         entries++;
1061                         if (entries == maxentries) {
1062                                 maxentries += MAXDIRS;
1063                                 if ((*dirv = realloc(*dirv,
1064                                     sizeof (char *) * maxentries)) == NULL)
1065                                         err(1, "realloc");
1066                                 dv = *dirv + entries;
1067                         }
1068                 }
1069         }
1070         *dv = 0;

1072         qsort((void *)*dirv, dv - *dirv, sizeof (char *), cmp);

1074 }


1077 /*
1078  * Search a section subdir for a given manpage.
1079  */
1080 static int
1081 searchdir(char *path, char *dir, char *name)
1082 {
1083         DIR             *sdp;
1084         struct dirent   *sd;
1085         char            sectpath[MAXPATHLEN];
1086         char            file[MAXNAMLEN];
1087         char            dname[MAXPATHLEN];
1088         char            *last;
1089         int             nlen;

1091         (void) snprintf(sectpath, sizeof (sectpath), "%s/%s", path, dir);
1092         (void) snprintf(file, sizeof (file), "%s.", name);

1094         if ((sdp = opendir(sectpath)) == NULL)
1095                 return (0);

1097         while ((sd = readdir(sdp))) {
1098                 char    *pname;

1100                 if ((pname = strdup(sd->d_name)) == NULL)
1101                         err(1, "strdup");
1102                 if ((last = strrchr(pname, '.')) != NULL &&
1103                     (strcmp(last, ".gz") == 0 || strcmp(last, ".bz2") == 0))
1104                         *last = '\0';
1105                 last = strrchr(pname, '.');
1106                 nlen = last - pname;
1107                 (void) snprintf(dname, sizeof (dname), "%.*s.", nlen, pname);
1108                 if (strcmp(dname, file) == 0 ||
1109                     strcmp(pname, name) == 0) {
1110                         (void) format(path, dir, name, sd->d_name);
1111                         (void) closedir(sdp);
1112                         free(pname);
1113                         return (1);
1114                 }
```



```
1115                 free(pname);
1116         }
1117         (void) closedir(sdp);

1119         return (0);
1120 }

1122 /*
1123  * Check the hash table of old directory names to see if there is a
1124  * new directory name.
1125  */
1126 static char *
1127 map_section(char *section, char *path)
1128 {
1129         int     i;
1130         int     len;
1131         char    fullpath[MAXPATHLEN];

1133         if (list)  /* -l option fall through */
1134                 return (NULL);

1136         for (i = 0; i <= ((sizeof (map)/sizeof (map[0]) - 1)); i++) {
1137                 if (strlen(section) > strlen(map[i].new_name)) {
1138                         len = strlen(section);
1139                 } else {
1140                         len = strlen(map[i].new_name);
1141                 }
1142                 if (strncmp(section, map[i].old_name, len) == 0) {
1143                         (void) snprintf(fullpath, sizeof (fullpath),
1144                             "%s/sman%s", path, map[i].new_name);
1145                         if (!access(fullpath, R_OK | X_OK)) {
1146                                 return (map[i].new_name);
1147                         } else {
1148                                 return (NULL);
1149                         }
1150                 }
1151         }

1153         return (NULL);
1154 }

1156 /*
1157  * Format the manpage.
1158  */
1159 static int
1160 format(char *path, char *dir, char *name, char *pg)
1161 {
1162         char            manpname[MAXPATHLEN], catpname[MAXPATHLEN];
1163         char            cmdbuf[BUFSIZ], tmpbuf[BUFSIZ];
1164         char            *cattool;
1165         int             utf8 = 0;
1166         struct stat     sbman, sbcat;

1168         found++;

1170         if (list) {
1171                 (void) printf(gettext("%s(%s)\t-M %s\n"), name, dir + 3, path);
1172                 return (-1);
1173         }

1175         (void) snprintf(manpname, sizeof (manpname), "%s/man%s/%s", path,
1176             dir + 3, pg);
1177         (void) snprintf(catpname, sizeof (catpname), "%s/cat%s/%s", path,
1178             dir + 3, pg);

1180         /* Can't do PS output if manpage doesn't exist */
```

```
1181         if (stat(manpname, &sbman) != 0 && psoutput)
1182                 return (-1);
1183
1184         /*
1185          * If both manpage and catpage do not exist, manpname is
1186          * broken symlink, most likely.
1187          */
1188         if (stat(catpname, &sbcat) != 0 && stat(manpname, &sbman) != 0)
1189                 err(1, "%s", manpname);
1190
1191         /* Setup cattool */
1192         if (fnmatch("*.gz", manpname, 0) == 0)
1193                 cattool = "gzcat";
1194         else if (fnmatch("*.bz2", manpname, 0) == 0)
1195                 cattool = "bzcat";
1196         else
1197                 cattool = "gzcat -f";
1198
1199         /* Preprocess UTF-8 input with preconv (could be smarter) */
1200         if (strstr(path, "UTF-8") != NULL)
1201                 utf8 = 1;
1202
1203         if (psoutput) {
1204                 (void) snprintf(cmdbuf, BUFSIZ,
1205                     "cd %s; %s %s%s | mandoc -Tps | lp -Tpostscript",
1206                     path, cattool, manpname,
1207                     utf8 ? " | " PRECONV " -e UTF-8" : "");
1208                 DPRINTF("-- Using manpage: %s\n", manpname);
1209                 goto cmd;
1210         }
1211
1212         /*
1213          * Output catpage if:
1214          * - manpage doesn't exist
1215          * - output width is standard and catpage is recent enough
1216          */
1217         if (stat(manpname, &sbman) != 0 || (manwidth == 0 &&
1218             stat(catpname, &sbcat) == 0 && sbcat.st_mtime >= sbman.st_mtime)) {
1219                 DPRINTF("-- Using catpage: %s\n", catpname);
1220                 (void) snprintf(cmdbuf, BUFSIZ, "%s %s", pager, catpname);
1221                 goto cmd;
1222         }
1223
1224         DPRINTF("-- Using manpage: %s\n", manpname);
1225         if (manwidth > 0)
1226                 (void) snprintf(tmpbuf, BUFSIZ, "-Owidth=%d ", manwidth);
1227         (void) snprintf(cmdbuf, BUFSIZ, "cd %s; %s %s%s | mandoc -T%s %s| %s",
1228             path, cattool, manpname,
1229             utf8 ? " | " PRECONV " -e UTF-8 " : "",
1230             utf8 ? "utf8" : "ascii", (manwidth > 0) ? tmpbuf : "", pager);
1231
1232 cmd:
1233         DPRINTF("-- Command: %s\n", cmdbuf);
1234
1235         if (!debug)
1236                 return (system(cmdbuf) == 0);
1237         else
1238                 return (0);
1239 }
1240
1241 /*
1242  * Add <localedir> to the path.
1243  */
1244 static char *
1245 addlocale(char *path)
1246 {
```



```
1247         char    *tmp;
1248
1249         if (asprintf(&tmp, "%s/%s", path, localedir) == -1)
1250                 err(1, "asprintf");
1251
1252         return (tmp);
1253 }
1254
1255 /*
1256  * Get the order of sections from man.cf.
1257  */
1258 static char *
1259 check_config(char *path)
1260 {
1261         FILE            *fp;
1262         char            *rc = NULL;
1263         char            *sect;
1264         char            fname[MAXPATHLEN];
1265         char            *line = NULL;
1266         size_t          linecap = 0;
1267
1268         (void) snprintf(fname, MAXPATHLEN, "%s/%s", path, CONFIG);
1269
1270         if ((fp = fopen(fname, "r")) == NULL)
1271                 return (NULL);
1272
1273         while (getline(&line, &linecap, fp) > 0) {
1274                 if ((rc = strstr(line, "MANSECTS")) != NULL)
1275                         break;
1276         }
1277
1278         (void) fclose(fp);
1279
1280         if (rc == NULL || (sect = strchr(line, '=')) == NULL)
1281                 return (NULL);
1282         else
1283                 return (++sect);
1284 }
1285
1286
1287 /*
1288  * Initialize the bintoman array with appropriate device and inode info.
1289  */
1290 static void
1291 init_bintoman(void)
1292 {
1293         int i;
1294         struct stat sb;
1295
1296         for (i = 0; bintoman[i].bindir != NULL; i++) {
1297                 if (stat(bintoman[i].bindir, &sb) == 0) {
1298                         bintoman[i].dev = sb.st_dev;
1299                         bintoman[i].ino = sb.st_ino;
1300                 } else {
1301                         bintoman[i].dev = NODEV;
1302                 }
1303         }
1304 }
1305
1306 /*
1307  * If a duplicate is found, return 1.
1308  * If a duplicate is not found, add it to the dupnode list and return 0.
1309  */
1310 static int
1311 dupcheck(struct man_node *mnp, struct dupnode **dnp)
1312 {
```

```
1313         struct dupnode  *curdnp;
1314         struct secnode  *cursnp;
1315         struct stat     sb;
1316         int             i;
1317         int             rv = 1;
1318         int             dupfound;

1320         /* If the path doesn't exist, treat it as a duplicate */
1321         if (stat(mnp->path, &sb) != 0)
1322                 return (1);

1324         /* If no sections were found in the man dir, treat it as duplicate */
1325         if (mnp->secv == NULL)
1326                 return (1);

1328         /*
1329          * Find the dupnode structure for the previous time this directory
1330          * was looked at.  Device and inode numbers are compared so that
1331          * directories that are reached via different paths (e.g. /usr/man and
1332          * /usr/share/man) are treated as equivalent.
1333          */
1334         for (curdnp = *dnp; curdnp != NULL; curdnp = curdnp->next) {
1335                 if (curdnp->dev == sb.st_dev && curdnp->ino == sb.st_ino)
1336                         break;
1337         }

1339         /*
1340          * First time this directory has been seen. Add a new node to the
1341          * head of the list. Since all entries are guaranteed to be unique
1342          * copy all sections to new node.
1343          */
1344         if (curdnp == NULL) {
1345                 if ((curdnp = calloc(1, sizeof (struct dupnode))) == NULL)
1346                         err(1, "calloc");
1347                 for (i = 0; mnp->secv[i] != NULL; i++) {
1348                         if ((cursnp = calloc(1, sizeof (struct secnode)))
1349                             == NULL)
1350                                 err(1, "calloc");
1351                         cursnp->next = curdnp->secl;
1352                         curdnp->secl = cursnp;
1353                         if ((cursnp->secp = strdup(mnp->secv[i])) == NULL)
1354                                 err(1, "strdup");
1355                 }
1356                 curdnp->dev = sb.st_dev;
1357                 curdnp->ino = sb.st_ino;
1358                 curdnp->next = *dnp;
1359                 *dnp = curdnp;
1360                 return (0);
1361         }

1363         /*
1364          * Traverse the section vector in the man_node and the section list
1365          * in dupnode cache to eliminate all duplicates from man_node.
1366          */
1367         for (i = 0; mnp->secv[i] != NULL; i++) {
1368                 dupfound = 0;
1369                 for (cursnp = curdnp->secl; cursnp != NULL;
1370                     cursnp = cursnp->next) {
1371                         if (strcmp(mnp->secv[i], cursnp->secp) == 0) {
1372                                 dupfound = 1;
1373                                 break;
1374                         }
1375                 }
1376                 if (dupfound) {
1377                         mnp->secv[i][0] = '\0';
1378                         continue;
```



```
1379                 }


1382                 /*
1383                  * Update curdnp and set return value to indicate that this
1384                  * was not all duplicates.
1385                  */
1386                 if ((cursnp = calloc(1, sizeof (struct secnode))) == NULL)
1387                         err(1, "calloc");
1388                 cursnp->next = curdnp->secl;
1389                 curdnp->secl = cursnp;
1390                 if ((cursnp->secp = strdup(mnp->secv[i])) == NULL)
1391                         err(1, "strdup");
1392                 rv = 0;
1393         }

1395         return (rv);
1396 }

1398 /*
1399  * Given a bindir, return corresponding mandir.
1400  */
1401 static char *
1402 path_to_manpath(char *bindir)
1403 {
1404         char            *mand, *p;
1405         int             i;
1406         struct stat     sb;

1408         /* First look for known translations for specific bin paths */
1409         if (stat(bindir, &sb) != 0) {
1410                 return (NULL);
1411         }
1412         for (i = 0; bintoman[i].bindir != NULL; i++) {
1413                 if (sb.st_dev == bintoman[i].dev &&
1414                     sb.st_ino == bintoman[i].ino) {
1415                         if ((mand = strdup(bintoman[i].mandir)) == NULL)
1416                                 err(1, "strdup");
1417                         if ((p = strchr(mand, ',')) != NULL)
1418                                 *p = '\0';
1419                         if (stat(mand, &sb) != 0) {
1420                                 free(mand);
1421                                 return (NULL);
1422                         }
1423                         if (p != NULL)
1424                                 *p = ',';
1425                         return (mand);
1426                 }
1427         }

1429         /*
1430          * No specific translation found.  Try 'dirname $bindir'/share/man
1431          * and 'dirname $bindir'/man
1432          */
1433         if ((mand = malloc(MAXPATHLEN)) == NULL)
1434                 err(1, "malloc");
1435         if (strlcpy(mand, bindir, MAXPATHLEN) >= MAXPATHLEN) {
1436                 free(mand);
1437                 return (NULL);
1438         }

1440         /*
1441          * Advance to end of buffer, strip trailing /'s then remove last
1442          * directory component.
1443          */
1444         for (p = mand; *p != '\0'; p++)
```

```
1445                 ;
1446         for (; p > mand && *p == '/'; p--)
1447                 ;
1448         for (; p > mand && *p != '/'; p--)
1449                 ;
1450         if (p == mand && *p == '.') {
1451                 if (realpath("..", mand) == NULL) {
1452                         free(mand);
1453                         return (NULL);
1454                 }
1455                 for (; *p != '\0'; p++)
1456                         ;
1457         } else {
1458                 *p = '\0';
1459         }

1461         if (strlcat(mand, "/share/man", MAXPATHLEN) >= MAXPATHLEN) {
1462                 free(mand);
1463                 return (NULL);
1464         }

1466         if ((stat(mand, &sb) == 0) && S_ISDIR(sb.st_mode)) {
1467                 return (mand);
1468         }

1470         /*
1471          * Strip the /share/man off and try /man
1472          */
1473         *p = '\0';
1474         if (strlcat(mand, "/man", MAXPATHLEN) >= MAXPATHLEN) {
1475                 free(mand);
1476                 return (NULL);
1477         }
1478         if ((stat(mand, &sb) == 0) && S_ISDIR(sb.st_mode)) {
1479                 return (mand);
1480         }

1482         /*
1483          * No man or share/man directory found
1484          */
1485         free(mand);
1486         return (NULL);
1487 }

1489 /*
1490  * Free a linked list of dupnode structs.
1491  */
1492 void
1493 free_dupnode(struct dupnode *dnp) {
1494         struct dupnode *dnp2;
1495         struct secnode *snp;

1497         while (dnp != NULL) {
1498                 dnp2 = dnp;
1499                 dnp = dnp->next;
1500                 while (dnp2->secl != NULL) {
1501                         snp = dnp2->secl;
1502                         dnp2->secl = dnp2->secl->next;
1503                         free(snp->secp);
1504                         free(snp);
1505                 }
1506                 free(dnp2);
1507         }
1508 }

1510 /*
```



```
1511  * Print manp linked list to stdout.
1512  */
1513 void
1514 print_manpath(struct man_node *manp)
1515 {
1516         char    colon[2] = "\0\0";
1517         char    **secp;

1519         for (; manp != NULL; manp = manp->next) {
1520                 (void) printf("%s%s", colon, manp->path);
1521                 colon[0] = ':';

1523                 /*
1524                  * If man.cf or a directory scan was used to create section
1525                  * list, do not print section list again.  If the output of
1526                  * man -p is used to set MANPATH, subsequent runs of man
1527                  * will re-read man.cf and/or scan man directories as
1528                  * required.
1529                  */
1530                 if (manp->defsrch != 0)
1531                         continue;

1533                 for (secp = manp->secv; *secp != NULL; secp++) {
1534                         /*
1535                          * Section deduplication may have eliminated some
1536                          * sections from the vector. Avoid displaying this
1537                          * detail which would appear as ",," in output
1538                          */
1539                         if ((*secp)[0] != '\0')
1540                                 (void) printf(",%s", *secp);
1541                 }
1542         }
1543         (void) printf("\n");
1544 }

1546 static void
1547 usage_man(void)
1548 {

1550         (void) fprintf(stderr, gettext(
1551 "usage: man [-alptw] [-M path] [-s section] name ...\n"
1552 "       man [-M path] [-s section] -k keyword -- emulate apropos\n"
1553 "       man [-M path] [-s section] -f keyword -- emulate whatis\n"));

1555         exit(1);
1556 }

1558 static void
1559 usage_whatapro(void)
1560 {

1562         (void) fprintf(stderr, gettext(
1563 "usage: %s [-M path] [-s section] keyword ...\n"),
1564             whatis ? "whatis" : "apropos");

1566         exit(1);
1567 }
```

```
**********************************************************
     939 Tue Jul 15 13:48:03 2014
new/usr/src/cmd/man/man.h
Finish integration.  Use mandoc_preconv, etc.
Initial import of man functionality.
**********************************************************
```

```
   1 /*
   2  * This file and its contents are supplied under the terms of the
   3  * Common Development and Distribution License ("CDDL"), version 1.0.
   4  * You may only use this file in accordance with the terms of version
   5  * 1.0 of the CDDL.
   6  *
   7  * A full copy of the text of the CDDL should have accompanied this
   8  * source.  A copy of the CDDL is also available via the Internet at
   9  * http://www.illumos.org/license/CDDL.
  10  */

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

  17 /*
  18  * Common definitions
  19  */

  21 #ifndef _MAN_H_
  22 #define _MAN_H_

  24 #define CONFIG          "man.cf"
  25 #define DEFMANDIR       "/usr/share/man"
  26 #define INDENT          24
  27 #define PAGER           "less -ins"
  28 #define WHATIS          "whatis"
  29 #define PRECONV         "/usr/lib/mandoc_preconv"

  31 #define LINE_ALLOC      4096
  32 #define MAXDIRS         128
  33 #define MAXTOKENS       64

  35 #define DPRINTF         if (debug) \
  36                                 (void) printf

  38 void    mwpath(char *path);

  40 #endif  /* _MAN_H_ */
```

```
**********************************************************
    2609 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/man/stringlist.c
Initial import of man functionality.
**********************************************************
```

```
/*
 * Copyright (c) 1994 Christos Zoulas
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/*
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#include <err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "stringlist.h"

#define _SL_CHUNKSIZE   20

stringlist *
sl_init(void)
{
        stringlist      *sl;

        if ((sl = malloc(sizeof (stringlist))) == NULL)
                err(1, "malloc");

        sl->sl_cur = 0;
        sl->sl_max = _SL_CHUNKSIZE;
        sl->sl_str = malloc(sl->sl_max * sizeof (char *));
        if (sl->sl_str == NULL)
                err(1, "malloc");

        return (sl);
}

int
sl_add(stringlist *sl, char *name)
{
```



```
        if (sl->sl_cur == sl->sl_max - 1) {
                sl->sl_max += _SL_CHUNKSIZE;
                sl->sl_str = realloc(sl->sl_str, sl->sl_max * sizeof (char *));
                if (sl->sl_str == NULL)
                        return (-1);
        }
        sl->sl_str[sl->sl_cur++] = name;

        return (0);
}


void
sl_free(stringlist *sl, int all)
{
        size_t i;

        if (sl == NULL)
                return;
        if (sl->sl_str) {
                if (all)
                        for (i = 0; i < sl->sl_cur; i++)
                                free(sl->sl_str[i]);
                free(sl->sl_str);
        }
        free(sl);
}


char *
sl_find(stringlist *sl, char *name)
{
        size_t  i;

        for (i = 0; i < sl->sl_cur; i++)
                if (strcmp(sl->sl_str[i], name) == 0)
                        return (sl->sl_str[i]);

        return (NULL);
}
```

```
**********************************************************
    2061 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/man/stringlist.h
Initial import of man functionality.
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 1994 Christos Zoulas
   3  * All rights reserved.
   4  *
   5  * Redistribution and use in source and binary forms, with or without
   6  * modification, are permitted provided that the following conditions
   7  * are met:
   8  * 1. Redistributions of source code must retain the above copyright
   9  *    notice, this list of conditions and the following disclaimer.
  10  * 2. Redistributions in binary form must reproduce the above copyright
  11  *    notice, this list of conditions and the following disclaimer in the
  12  *    documentation and/or other materials provided with the distribution.
  13  * 3. All advertising materials mentioning features or use of this software
  14  *    must display the following acknowledgement:
  15  *      This product includes software developed by Christos Zoulas.
  16  * 4. The name of the author may not be used to endorse or promote products
  17  *    derived from this software without specific prior written permission.
  18  *
  19  * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS
  20  * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
  21  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  22  * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
  23  * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  24  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  25  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  26  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  27  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  28  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  29  * SUCH DAMAGE.
  30  */

  32 /*
  33  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  34  */

  36 #ifndef _STRINGLIST_H_
  37 #define _STRINGLIST_H_

  39 #include <sys/types.h>

  41 typedef struct _stringlist {
  42         char    **sl_str;
  43         size_t  sl_max;
  44         size_t  sl_cur;
  45 } stringlist;

  47 stringlist      *sl_init(void);
  48 int             sl_add(stringlist *, char *);
  49 void            sl_free(stringlist *, int);
  50 char            *sl_find(stringlist *, char *);

  52 #endif  /* _STRINGLIST_H_ */
```

```
**********************************************************
    1770 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/Makefile
Finish integration.  Use mandoc_preconv, etc.
import complete (hopefully)
Initial import of man functionality.
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

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

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

  27 preconv_OBJS =  preconv.o

  29 # We place preconv in /usr/lib.  This is done to avoid conflicting with
  30 # GNU groff, which puts it into /usr/bin.  We also rename it so that it
  31 # will only be seen by mandoc -- it isn't intended for general end-user use.

  33 ROOTPROGS =     $(ROOTBIN)/mandoc $(ROOTLIB)/mandoc_preconv

  35 OBJS=           $(preconv_OBJS) $(mandoc_OBJS)

  37 include         $(SRC)/cmd/Makefile.cmd

  39 CFLAGS +=       $(CC_VERBOSE)

  41 CPPFLAGS +=     -DHAVE_CONFIG_H -DUSE_WCHAR     \
  42                 -DOSNAME="\"illumos\""  \
  43                 -DVERSION="\"1.12.1\""

  45 .KEEP_STATE:

  47 all:            $(PROGS)

  49 mandoc_preconv: $(preconv_OBJS)
  50                 $(LINK.c) $(preconv_OBJS) -o $@ $(LDLIBS)
  51                 $(POST_PROCESS)

  53 mandoc:         $(mandoc_OBJS)
  54                 $(LINK.c) $(mandoc_OBJS) -o $@ $(LDLIBS)
  55                 $(POST_PROCESS)

  57 clean:
  58                 $(RM) $(OBJS)
```



```
  60 install:        all $(ROOTPROGS)

  62 include         $(SRC)/cmd/Makefile.targ
```

****
825 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/THIRDPARTYLICENSE
Initial import of man functionality.
****

```
Copyright (c) 2008, 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 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.
```

```
**********************************************************
      41 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/THIRDPARTYLICENSE.descrip
Initial import of man functionality.
**********************************************************
```

```
   1 MANDOC - FORMAT AND DISPLAY UNIX MANUALS
```

```
**********************************************************
    1159 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/arch.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: arch.c,v 1.9 2011/03/22 14:33:05 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 <stdlib.h>
#include <string.h>
#include <time.h>

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

#define LINE(x, y) \
        if (0 == strcmp(p, x)) return(y);

const char *
mdoc_a2arch(const char *p)
{

#include "arch.in"

        return(NULL);
}
```

```
**********************************************************
    3281 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/arch.in
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: arch.in,v 1.12 2012/01/28 14:02:17 joerg Exp $ */
/*
 * Copyright (c) 2009 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 architecture token of the .Dt prologue macro.
 * All architectures that your system supports (or the manuals of your
 * system) should be included here.  The right-hand-side is the
 * formatted output.
 *
 * Be sure to escape strings.
 *
 * REMEMBER TO ADD NEW ARCHITECTURES TO MDOC.7!
 */

LINE("acorn26",         "Acorn26")
LINE("acorn32",         "Acorn32")
LINE("algor",           "Algor")
LINE("alpha",           "Alpha")
LINE("amd64",           "AMD64")
LINE("amiga",           "Amiga")
LINE("amigappc",        "AmigaPPC")
LINE("arc",             "ARC")
LINE("arm",             "ARM")
LINE("arm26",           "ARM26")
LINE("arm32",           "ARM32")
LINE("armish",          "ARMISH")
LINE("aviion",          "AViiON")
LINE("atari",           "ATARI")
LINE("beagle",          "Beagle")
LINE("bebox",           "BeBox")
LINE("cats",            "cats")
LINE("cesfic",          "CESFIC")
LINE("cobalt",          "Cobalt")
LINE("dreamcast",       "Dreamcast")
LINE("emips",           "EMIPS")
LINE("evbarm",          "evbARM")
LINE("evbmips",         "evbMIPS")
LINE("evbppc",          "evbPPC")
LINE("evbsh3",          "evbSH3")
LINE("ews4800mips",     "EWS4800MIPS")
LINE("hp300",           "HP300")
LINE("hp700",           "HP700")
LINE("hpcarm",          "HPCARM")
LINE("hpcmips",         "HPCMIPS")
LINE("hpcsh",           "HPCSH")
LINE("hppa",            "HPPA")
LINE("hppa64",          "HPPA64")
```



```
LINE("ia64",            "ia64")
LINE("i386",            "i386")
LINE("ibmnws",          "IBMNWS")
LINE("iyonix",          "Iyonix")
LINE("landisk",         "LANDISK")
LINE("loongson",        "Loongson")
LINE("luna68k",         "Luna68k")
LINE("luna88k",         "Luna88k")
LINE("m68k",            "m68k")
LINE("mac68k",          "Mac68k")
LINE("macppc",          "MacPPC")
LINE("mips",            "MIPS")
LINE("mips64",          "MIPS64")
LINE("mipsco",          "MIPSCo")
LINE("mmeye",           "mmEye")
LINE("mvme68k",         "MVME68k")
LINE("mvme88k",         "MVME88k")
LINE("mvmeppc",         "MVMEPPC")
LINE("netwinder",       "NetWinder")
LINE("news68k",         "NeWS68k")
LINE("newsmips",        "NeWSMIPS")
LINE("next68k",         "NeXT68k")
LINE("ofppc",           "OFPPC")
LINE("palm",            "Palm")
LINE("pc532",           "PC532")
LINE("playstation2",    "PlayStation2")
LINE("pmax",            "PMAX")
LINE("pmppc",           "pmPPC")
LINE("powerpc",         "PowerPC")
LINE("prep",            "PReP")
LINE("rs6000",          "RS6000")
LINE("sandpoint",       "Sandpoint")
LINE("sbmips",          "SBMIPS")
LINE("sgi",             "SGI")
LINE("sgimips",         "SGIMIPS")
LINE("sh3",             "SH3")
LINE("shark",           "Shark")
LINE("socppc",          "SOCPPC")
LINE("solbourne",       "Solbourne")
LINE("sparc",           "SPARC")
LINE("sparc64",         "SPARC64")
LINE("sun2",            "Sun2")
LINE("sun3",            "Sun3")
LINE("tahoe",           "Tahoe")
LINE("vax",             "VAX")
LINE("x68k",            "X68k")
LINE("x86",             "x86")
LINE("x86_64",          "x86_64")
LINE("xen",             "Xen")
LINE("zaurus",          "Zaurus")
```

```
**********************************************************
    1156 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/att.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: att.c,v 1.9 2011/03/22 14:33:05 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 <stdlib.h>
#include <string.h>
#include <time.h>

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

#define LINE(x, y) \
        if (0 == strcmp(p, x)) return(y);

const char *
mdoc_a2att(const char *p)
{

#include "att.in"

        return(NULL);
}
```

```
**********************************************************
    1738 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/att.in
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: att.in,v 1.8 2011/07/31 17:30:33 schwarze Exp $ */
/*
 * Copyright (c) 2009 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 AT&T versions of the .At macro.  This probably
 * isn't going to change.  The right-hand side is the formatted 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.
 */

LINE("v1",              "Version\\~1 AT&T UNIX")
LINE("v2",              "Version\\~2 AT&T UNIX")
LINE("v3",              "Version\\~3 AT&T UNIX")
LINE("v4",              "Version\\~4 AT&T UNIX")
LINE("v5",              "Version\\~5 AT&T UNIX")
LINE("v6",              "Version\\~6 AT&T UNIX")
LINE("v7",              "Version\\~7 AT&T UNIX")
LINE("32v",             "Version\\~32V AT&T UNIX")
LINE("III",             "AT&T System\\~III UNIX")
LINE("V",               "AT&T System\\~V UNIX")
LINE("V.1",             "AT&T System\\~V Release\\~1 UNIX")
LINE("V.2",             "AT&T System\\~V Release\\~2 UNIX")
LINE("V.3",             "AT&T System\\~V Release\\~3 UNIX")
LINE("V.4",             "AT&T System\\~V Release\\~4 UNIX")
```

```
**********************************************************
    3566 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/chars.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: chars.c,v 1.52 2011/11/08 00:15:23 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 "mandoc.h"
#include "libmandoc.h"

#define PRINT_HI         126
#define PRINT_LO         32

struct  ln {
        struct ln        *next;
        const char       *code;
        const char       *ascii;
        int               unicode;
};

#define LINES_MAX         328

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

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

#include "chars.in"

struct  mchars {
        struct ln       **htab;
};

static  const struct ln  *find(const struct mchars *,
                               const char *, size_t);

void
mchars_free(struct mchars *arg)
{

        free(arg->htab);
```



```
        free(arg);
}

struct mchars *
mchars_alloc(void)
{
        struct mchars    *tab;
        struct ln       **htab;
        struct ln        *pp;
        int               i, hash;

        /*
         * Constructs a very basic chaining hashtable.  The hash routine
         * is simply the integral value of the first character.
         * Subsequent entries are chained in the order they're processed.
         */

        tab = mandoc_malloc(sizeof(struct mchars));
        htab = mandoc_calloc(PRINT_HI - PRINT_LO + 1, sizeof(struct ln **));

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

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

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

        tab->htab = htab;
        return(tab);
}

int
mchars_spec2cp(const struct mchars *arg, const char *p, size_t sz)
{
        const struct ln *ln;

        ln = find(arg, p, sz);
        if (NULL == ln)
                return(-1);
        return(ln->unicode);
}

char
mchars_num2char(const char *p, size_t sz)
{
        int               i;

        if ((i = mandoc_strntoi(p, sz, 10)) < 0)
                return('\0');
        return(i > 0 && i < 256 && isprint(i) ?
                        /* LINTED */ i : '\0');
}

int
mchars_num2uc(const char *p, size_t sz)
{
        int               i;

        if ((i = mandoc_strntoi(p, sz, 16)) < 0)
                return('\0');
```

```
128         /* FIXME: make sure we're not in a bogus range. */
129         return(i > 0x80 && i <= 0x10FFFF ? i : '\0');
130 }

132 const char *
133 mchars_spec2str(const struct mchars *arg,
134                 const char *p, size_t sz, size_t *rsz)
135 {
136         const struct ln *ln;

138         ln = find(arg, p, sz);
139         if (NULL == ln) {
140                 *rsz = 1;
141                 return(NULL);
142         }

144         *rsz = strlen(ln->ascii);
145         return(ln->ascii);
146 }

148 static const struct ln *
149 find(const struct mchars *tab, const char *p, size_t sz)
150 {
151         const struct ln  *pp;
152         int               hash;

154         assert(p);

156         if (0 == sz || p[0] < PRINT_LO || p[0] > PRINT_HI)
157                 return(NULL);

159         hash = (int)p[0] - PRINT_LO;

161         for (pp = tab->htab[hash]; pp; pp = pp->next)
162                 if (0 == strncmp(pp->code, p, sz) &&
163                                 '\0' == pp->code[(int)sz])
164                         return(pp);

166         return(NULL);
167 }
```

```
**********************************************************
   10032 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/chars.in
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: chars.in,v 1.42 2011/10/02 10:02:26 kristaps Exp $ */
/*
 * 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.
 */

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

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

CHAR_TBL_START

/* Spacing. */
CHAR("c",                       "",             0)
CHAR("0",                       " ",            8194)
CHAR(" ",                       ascii_nbrsp,    160)
CHAR("~",                       ascii_nbrsp,    160)
CHAR("%",                       "",             0)
CHAR("&",                       "",             0)
CHAR("^",                       "",             0)
CHAR("|",                       "",             0)
CHAR("}",                       "",             0)

/* Accents. */
CHAR("a\"",                     "\"",           779)
CHAR("a-",                      "-",            175)
CHAR("a.",                      ".",            729)
CHAR("a^",                      "^",            770)
CHAR("\'",                      "\'",           769)
CHAR("aa",                      "\'",           769)
CHAR("ga",                      "`",            768)
CHAR("`",                       "`",            768)
CHAR("ab",                      "_",            774)
CHAR("ac",                      ",",            807)
CHAR("ad",                      "\"",           776)
CHAR("ah",                      "v",            711)
CHAR("ao",                      "o",            730)
CHAR("a~",                      "~",            771)
CHAR("ho",                      ",",            808)
```



```
CHAR("ha",                      "^",            94)
CHAR("ti",                      "~",            126)

/* Quotes. */
CHAR("Bq",                      ",,",           8222)
CHAR("bq",                      ",",            8218)
CHAR("lq",                      "``",           8220)
CHAR("rq",                      "\'\'",         8221)
CHAR("oq",                      "`",            8216)
CHAR("cq",                      "\'",           8217)
CHAR("aq",                      "\'",           39)
CHAR("dq",                      "\"",           34)
CHAR("Fo",                      "<<",           171)
CHAR("Fc",                      ">>",           187)
CHAR("fo",                      "<",            8249)
CHAR("fc",                      ">",            8250)

/* Brackets. */
CHAR("lB",                      "[",            91)
CHAR("rB",                      "]",            93)
CHAR("lC",                      "{",            123)
CHAR("rC",                      "}",            125)
CHAR("la",                      "<",            60)
CHAR("ra",                      ">",            62)
CHAR("bv",                      "|",            9130)
CHAR("braceex",                 "|",            9130)
CHAR("bracketlefttp",           "|",            9121)
CHAR("bracketleftbp",           "|",            9123)
CHAR("bracketleftex",           "|",            9122)
CHAR("bracketrighttp",          "|",            9124)
CHAR("bracketrightbp",          "|",            9126)
CHAR("bracketrightex",          "|",            9125)
CHAR("lt",                      ",-",           9127)
CHAR("braceleftтp",             ",-",           9127)
CHAR("lk",                      "{",            9128)
CHAR("braceleftmid",            "{",            9128)
CHAR("lb",                      ",-",           9129)
CHAR("braceleftbp",             "`-",           9129)
CHAR("braceleftex",             "|",            9130)
CHAR("rt",                      "-.",           9131)
CHAR("bracerighttp",            "-.",           9131)
CHAR("rk",                      "}",            9132)
CHAR("bracerightmid",           "}",            9132)
CHAR("rb",                      "-\'",          9133)
CHAR("bracerightbp",            "-\'",          9133)
CHAR("bracerightex",            "|",            9130)
CHAR("parenlefttp",             "/",            9115)
CHAR("parenleftbp",             "\\",           9117)
CHAR("parenleftex",             "|",            9116)
CHAR("parenrighttp",            "\\",           9118)
CHAR("parenrightbp",            "/",            9120)
CHAR("parenrightex",            "|",            9119)

/* Greek characters. */
CHAR("*A",                      "A",            913)
CHAR("*B",                      "B",            914)
CHAR("*G",                      "|",            915)
CHAR("*D",                      "/\\",          916)
CHAR("*E",                      "E",            917)
CHAR("*Z",                      "Z",            918)
CHAR("*Y",                      "H",            919)
CHAR("*H",                      "O",            920)
CHAR("*I",                      "I",            921)
CHAR("*K",                      "K",            922)
CHAR("*L",                      "/\\",          923)
CHAR("*M",                      "M",            924)
```

```
128 CHAR("*N",                  "N",            925)
129 CHAR("*C",                  "H",            926)
130 CHAR("*O",                  "O",            927)
131 CHAR("*P",                  "TT",           928)
132 CHAR("*R",                  "P",            929)
133 CHAR("*S",                  ">",            931)
134 CHAR("*T",                  "T",            932)
135 CHAR("*U",                  "Y",            933)
136 CHAR("*F",                  "O_",           934)
137 CHAR("*X",                  "X",            935)
138 CHAR("*Q",                  "Y",            936)
139 CHAR("*W",                  "O",            937)
140 CHAR("*a",                  "a",            945)
141 CHAR("*b",                  "B",            946)
142 CHAR("*g",                  "y",            947)
143 CHAR("*d",                  "d",            948)
144 CHAR("*e",                  "e",            949)
145 CHAR("*z",                  "C",            950)
146 CHAR("*y",                  "n",            951)
147 CHAR("*h",                  "0",            952)
148 CHAR("*i",                  "i",            953)
149 CHAR("*k",                  "k",            954)
150 CHAR("*l",                  "\\",           955)
151 CHAR("*m",                  "u",            956)
152 CHAR("*n",                  "v",            957)
153 CHAR("*c",                  "E",            958)
154 CHAR("*o",                  "o",            959)
155 CHAR("*p",                  "n",            960)
156 CHAR("*r",                  "p",            961)
157 CHAR("*s",                  "o",            963)
158 CHAR("*t",                  "t",            964)
159 CHAR("*u",                  "u",            965)
160 CHAR("*f",                  "o",            981)
161 CHAR("*x",                  "x",            967)
162 CHAR("*q",                  "u",            968)
163 CHAR("*w",                  "w",            969)
164 CHAR("+h",                  "0",            977)
165 CHAR("+f",                  "o",            966)
166 CHAR("+p",                  "w",            982)
167 CHAR("+e",                  "e",            1013)
168 CHAR("ts",                  "s",            962)

170 /* Accented letters. */
171 CHAR(",C",                  "C",            199)
172 CHAR(",c",                  "c",            231)
173 CHAR("/L",                  "L",            321)
174 CHAR("/O",                  "O",            216)
175 CHAR("/l",                  "l",            322)
176 CHAR("/o",                  "o",            248)
177 CHAR("oA",                  "A",            197)
178 CHAR("oa",                  "a",            229)
179 CHAR(":A",                  "A",            196)
180 CHAR(":E",                  "E",            203)
181 CHAR(":I",                  "I",            207)
182 CHAR(":O",                  "O",            214)
183 CHAR(":U",                  "U",            220)
184 CHAR(":a",                  "a",            228)
185 CHAR(":e",                  "e",            235)
186 CHAR(":i",                  "i",            239)
187 CHAR(":o",                  "o",            246)
188 CHAR(":u",                  "u",            252)
189 CHAR(":y",                  "y",            255)
190 CHAR("\'A",                 "A",            193)
191 CHAR("\'E",                 "E",            201)
192 CHAR("\'I",                 "I",            205)
193 CHAR("\'O",                 "O",            211)
```



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

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

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

253 /* Mathematicals. */
254 CHAR("pl",                  "+",            43)
255 CHAR("mi",                  "-",            8722)
256 CHAR("-",                   "-",            45)
257 CHAR("-+",                  "-+",           8723)
258 CHAR("+-",                  "+-",           177)
259 CHAR("t+-",                 "+-",           177)
```

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

319 /* Ligatures. */
320 CHAR("ff",                  "ff",         64256)
321 CHAR("fi",                  "fi",         64257)
322 CHAR("fl",                  "fl",         64258)
323 CHAR("Fi",                  "ffi",        64259)
324 CHAR("Fl",                  "ffl",        64260)
325 CHAR("AE",                  "AE",         198)
```



```
326 CHAR("ae",                  "ae",         230)
327 CHAR("OE",                  "OE",         338)
328 CHAR("oe",                  "oe",         339)
329 CHAR("ss",                  "ss",         223)
330 CHAR("IJ",                  "IJ",         306)
331 CHAR("ij",                  "ij",         307)

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

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

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

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

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

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

390 /* Units. */
391 CHAR("de",                  "o",          176)
```

```
392 CHAR("%0",                  "%o",           8240)
393 CHAR("fm",                  "\'",           8242)
394 CHAR("sd",                  "\"",           8243)
395 CHAR("mc",                  "mu",           181)

397 CHAR_TBL_END
```

```
**********************************************************
    1208 Tue Jul 15 13:48:04 2014
new/usr/src/cmd/mandoc/config.h
Initial import of man functionality.
**********************************************************
```

```
   1 #ifndef MANDOC_CONFIG_H
   2 #define MANDOC_CONFIG_H

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

   8 #include <stdio.h>

  10 #define HAVE_STRPTIME
  11 #define HAVE_GETSUBOPT
  12 #define HAVE_STRLCAT
  13 #define HAVE_STRLCPY

  15 #include <sys/types.h>

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

  32 #if defined(__APPLE__)
  33 # define htobe32(x) OSSwapHostToBigInt32(x)
  34 # define betoh32(x) OSSwapBigToHostInt32(x)
  35 # define htobe64(x) OSSwapHostToBigInt64(x)
  36 # define betoh64(x) OSSwapBigToHostInt64(x)
  37 #elif defined(__linux__)
  38 # define betoh32(x) be32toh(x)
  39 # define betoh64(x) be64toh(x)
  40 #endif

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

  56 #endif /* MANDOC_CONFIG_H */
```

```
**********************************************************
   21161 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/eqn.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: eqn.c,v 1.38 2011/07/25 15:37:00 kristaps Exp $ */
/*
 * Copyright (c) 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 <assert.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

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

#define EQN_NEST_MAX     128 /* maximum nesting of defines */
#define EQN_MSG(t, x)    mandoc_msg((t), (x)->parse, (x)->eqn.ln, (x)->eqn.pos,

enum    eqn_rest {
        EQN_DESCOPE,
        EQN_ERR,
        EQN_OK,
        EQN_EOF
};

enum    eqn_symt {
        EQNSYM_alpha,
        EQNSYM_beta,
        EQNSYM_chi,
        EQNSYM_delta,
        EQNSYM_epsilon,
        EQNSYM_eta,
        EQNSYM_gamma,
        EQNSYM_iota,
        EQNSYM_kappa,
        EQNSYM_lambda,
        EQNSYM_mu,
        EQNSYM_nu,
        EQNSYM_omega,
        EQNSYM_omicron,
        EQNSYM_phi,
        EQNSYM_pi,
        EQNSYM_ps,
        EQNSYM_rho,
        EQNSYM_sigma,
```



```
        EQNSYM_tau,
        EQNSYM_theta,
        EQNSYM_upsilon,
        EQNSYM_xi,
        EQNSYM_zeta,
        EQNSYM_DELTA,
        EQNSYM_GAMMA,
        EQNSYM_LAMBDA,
        EQNSYM_OMEGA,
        EQNSYM_PHI,
        EQNSYM_PI,
        EQNSYM_PSI,
        EQNSYM_SIGMA,
        EQNSYM_THETA,
        EQNSYM_UPSILON,
        EQNSYM_XI,
        EQNSYM_inter,
        EQNSYM_union,
        EQNSYM_prod,
        EQNSYM_int,
        EQNSYM_sum,
        EQNSYM_grad,
        EQNSYM_del,
        EQNSYM_times,
        EQNSYM_cdot,
        EQNSYM_nothing,
        EQNSYM_approx,
        EQNSYM_prime,
        EQNSYM_half,
        EQNSYM_partial,
        EQNSYM_inf,
        EQNSYM_muchgreat,
        EQNSYM_muchless,
        EQNSYM_larrow,
        EQNSYM_rarrow,
        EQNSYM_pm,
        EQNSYM_nequal,
        EQNSYM_equiv,
        EQNSYM_lessequal,
        EQNSYM_moreequal,
        EQNSYM__MAX
};

enum    eqnpartt {
        EQN_DEFINE = 0,
        EQN_NDEFINE,
        EQN_TDEFINE,
        EQN_SET,
        EQN_UNDEF,
        EQN_GFONT,
        EQN_GSIZE,
        EQN_BACK,
        EQN_FWD,
        EQN_UP,
        EQN_DOWN,
        EQN__MAX
};

struct  eqnstr {
        const char      *name;
        size_t           sz;
};

#define STRNEQ(p1, sz1, p2, sz2) \
        ((sz1) == (sz2) && 0 == strncmp((p1), (p2), (sz1)))
#define EQNSTREQ(x, p, sz) \
```

```
128         STRNEQ((x)->name, (x)->sz, (p), (sz))

130 struct  eqnpart {
131         struct eqnstr    str;
132         int             (*fp)(struct eqn_node *);
133 };

135 struct  eqnsym {
136         struct eqnstr    str;
137         const char      *sym;
138 };


141 static  enum eqn_rest    eqn_box(struct eqn_node *, struct eqn_box *);
142 static  struct eqn_box  *eqn_box_alloc(struct eqn_node *,
143                                 struct eqn_box *);
144 static  void             eqn_box_free(struct eqn_box *);
145 static  struct eqn_def  *eqn_def_find(struct eqn_node *,
146                                 const char *, size_t);
147 static  int              eqn_do_gfont(struct eqn_node *);
148 static  int              eqn_do_gsize(struct eqn_node *);
149 static  int              eqn_do_define(struct eqn_node *);
150 static  int              eqn_do_ign1(struct eqn_node *);
151 static  int              eqn_do_ign2(struct eqn_node *);
152 static  int              eqn_do_tdefine(struct eqn_node *);
153 static  int              eqn_do_undef(struct eqn_node *);
154 static  enum eqn_rest    eqn_eqn(struct eqn_node *, struct eqn_box *);
155 static  enum eqn_rest    eqn_list(struct eqn_node *, struct eqn_box *);
156 static  enum eqn_rest    eqn_matrix(struct eqn_node *, struct eqn_box *);
157 static  const char      *eqn_nexttok(struct eqn_node *, size_t *);
158 static  const char      *eqn_nextrawtok(struct eqn_node *, size_t *);
159 static  const char      *eqn_next(struct eqn_node *,
160                                 char, size_t *, int);
161 static  void             eqn_rewind(struct eqn_node *);

163 static  const struct eqnpart eqnparts[EQN__MAX] = {
164         { { "define", 6 }, eqn_do_define }, /* EQN_DEFINE */
165         { { "ndefine", 7 }, eqn_do_define }, /* EQN_NDEFINE */
166         { { "tdefine", 7 }, eqn_do_tdefine }, /* EQN_TDEFINE */
167         { { "set", 3 }, eqn_do_ign2 }, /* EQN_SET */
168         { { "undef", 5 }, eqn_do_undef }, /* EQN_UNDEF */
169         { { "gfont", 5 }, eqn_do_gfont }, /* EQN_GFONT */
170         { { "gsize", 5 }, eqn_do_gsize }, /* EQN_GSIZE */
171         { { "back", 4 }, eqn_do_ign1 }, /* EQN_BACK */
172         { { "fwd", 3 }, eqn_do_ign1 }, /* EQN_FWD */
173         { { "up", 2 }, eqn_do_ign1 }, /* EQN_UP */
174         { { "down", 4 }, eqn_do_ign1 }, /* EQN_DOWN */
175 };

177 static  const struct eqnstr eqnmarks[EQNMARK__MAX] = {
178         { "", 0 }, /* EQNMARK_NONE */
179         { "dot", 3 }, /* EQNMARK_DOT */
180         { "dotdot", 6 }, /* EQNMARK_DOTDOT */
181         { "hat", 3 }, /* EQNMARK_HAT */
182         { "tilde", 5 }, /* EQNMARK_TILDE */
183         { "vec", 3 }, /* EQNMARK_VEC */
184         { "dyad", 4 }, /* EQNMARK_DYAD */
185         { "bar", 3 }, /* EQNMARK_BAR */
186         { "under", 5 }, /* EQNMARK_UNDER */
187 };

189 static  const struct eqnstr eqnfonts[EQNFONT__MAX] = {
190         { "", 0 }, /* EQNFONT_NONE */
191         { "roman", 5 }, /* EQNFONT_ROMAN */
192         { "bold", 4 }, /* EQNFONT_BOLD */
193         { "fat", 3 }, /* EQNFONT_FAT */
```



```
194         { "italic", 6 }, /* EQNFONT_ITALIC */
195 };

197 static  const struct eqnstr eqnposs[EQNPOS__MAX] = {
198         { "", 0 }, /* EQNPOS_NONE */
199         { "over", 4 }, /* EQNPOS_OVER */
200         { "sup", 3 }, /* EQNPOS_SUP */
201         { "sub", 3 }, /* EQNPOS_SUB */
202         { "to", 2 }, /* EQNPOS_TO */
203         { "from", 4 }, /* EQNPOS_FROM */
204 };

206 static  const struct eqnstr eqnpiles[EQNPILE__MAX] = {
207         { "", 0 }, /* EQNPILE_NONE */
208         { "pile", 4 }, /* EQNPILE_PILE */
209         { "cpile", 5 }, /* EQNPILE_CPILE */
210         { "rpile", 5 }, /* EQNPILE_RPILE */
211         { "lpile", 5 }, /* EQNPILE_LPILE */
212         { "col", 3 }, /* EQNPILE_COL */
213         { "ccol", 4 }, /* EQNPILE_CCOL */
214         { "rcol", 4 }, /* EQNPILE_RCOL */
215         { "lcol", 4 }, /* EQNPILE_LCOL */
216 };

218 static  const struct eqnsym eqnsyms[EQNSYM__MAX] = {
219         { { "alpha", 5 }, "*a" }, /* EQNSYM_alpha */
220         { { "beta", 4 }, "*b" }, /* EQNSYM_beta */
221         { { "chi", 3 }, "*x" }, /* EQNSYM_chi */
222         { { "delta", 5 }, "*d" }, /* EQNSYM_delta */
223         { { "epsilon", 7 }, "*e" }, /* EQNSYM_epsilon */
224         { { "eta", 3 }, "*y" }, /* EQNSYM_eta */
225         { { "gamma", 5 }, "*g" }, /* EQNSYM_gamma */
226         { { "iota", 4 }, "*i" }, /* EQNSYM_iota */
227         { { "kappa", 5 }, "*k" }, /* EQNSYM_kappa */
228         { { "lambda", 6 }, "*l" }, /* EQNSYM_lambda */
229         { { "mu", 2 }, "*m" }, /* EQNSYM_mu */
230         { { "nu", 2 }, "*n" }, /* EQNSYM_nu */
231         { { "omega", 5 }, "*w" }, /* EQNSYM_omega */
232         { { "omicron", 7 }, "*o" }, /* EQNSYM_omicron */
233         { { "phi", 3 }, "*f" }, /* EQNSYM_phi */
234         { { "pi", 2 }, "*p" }, /* EQNSYM_pi */
235         { { "psi", 2 }, "*q" }, /* EQNSYM_psi */
236         { { "rho", 3 }, "*r" }, /* EQNSYM_rho */
237         { { "sigma", 5 }, "*s" }, /* EQNSYM_sigma */
238         { { "tau", 3 }, "*t" }, /* EQNSYM_tau */
239         { { "theta", 5 }, "*h" }, /* EQNSYM_theta */
240         { { "upsilon", 7 }, "*u" }, /* EQNSYM_upsilon */
241         { { "xi", 2 }, "*c" }, /* EQNSYM_xi */
242         { { "zeta", 4 }, "*z" }, /* EQNSYM_zeta */
243         { { "DELTA", 5 }, "*D" }, /* EQNSYM_DELTA */
244         { { "GAMMA", 5 }, "*G" }, /* EQNSYM_GAMMA */
245         { { "LAMBDA", 6 }, "*L" }, /* EQNSYM_LAMBDA */
246         { { "OMEGA", 5 }, "*W" }, /* EQNSYM_OMEGA */
247         { { "PHI", 3 }, "*F" }, /* EQNSYM_PHI */
248         { { "PI", 2 }, "*P" }, /* EQNSYM_PI */
249         { { "PSI", 3 }, "*Q" }, /* EQNSYM_PSI */
250         { { "SIGMA", 5 }, "*S" }, /* EQNSYM_SIGMA */
251         { { "THETA", 5 }, "*H" }, /* EQNSYM_THETA */
252         { { "UPSILON", 7 }, "*U" }, /* EQNSYM_UPSILON */
253         { { "XI", 2 }, "*C" }, /* EQNSYM_XI */
254         { { "inter", 5 }, "ca" }, /* EQNSYM_inter */
255         { { "union", 5 }, "cu" }, /* EQNSYM_union */
256         { { "prod", 4 }, "product" }, /* EQNSYM_prod */
257         { { "int", 3 }, "integral" }, /* EQNSYM_int */
258         { { "sum", 3 }, "sum" }, /* EQNSYM_sum */
259         { { "grad", 4 }, "gr" }, /* EQNSYM_grad */
```

```
260         { { "del", 3 }, "gr" }, /* EQNSYM_del */
261         { { "times", 5 }, "mu" }, /* EQNSYM_times */
262         { { "cdot", 4 }, "pc" }, /* EQNSYM_cdot */
263         { { "nothing", 7 }, "&" }, /* EQNSYM_nothing */
264         { { "approx", 6 }, "~~" }, /* EQNSYM_approx */
265         { { "prime", 5 }, "aq" }, /* EQNSYM_prime */
266         { { "half", 4 }, "12" }, /* EQNSYM_half */
267         { { "partial", 7 }, "pd" }, /* EQNSYM_partial */
268         { { "inf", 3 }, "if" }, /* EQNSYM_inf */
269         { { ">>", 2 }, ">>" }, /* EQNSYM_muchgreat */
270         { { "<<", 2 }, "<<" }, /* EQNSYM_muchless */
271         { { "<-", 2 }, "<-" }, /* EQNSYM_larrow */
272         { { "->", 2 }, "->" }, /* EQNSYM_rarrow */
273         { { "+-", 2 }, "+-" }, /* EQNSYM_pm */
274         { { "!=", 2 }, "!=" }, /* EQNSYM_nequal */
275         { { "==", 2 }, "==" }, /* EQNSYM_equiv */
276         { { "<=", 2 }, "<=" }, /* EQNSYM_lessequal */
277         { { ">=", 2 }, ">=" }, /* EQNSYM_moreequal */
278 };

280 /* ARGSUSED */
281 enum rofferr
282 eqn_read(struct eqn_node **epp, int ln,
283                 const char *p, int pos, int *offs)
284 {
285         size_t           sz;
286         struct eqn_node *ep;
287         enum rofferr     er;

289         ep = *epp;

291         /*
292          * If we're the terminating mark, unset our equation status and
293          * validate the full equation.
294          */

296         if (0 == strncmp(p, ".EN", 3)) {
297                 er = eqn_end(epp);
298                 p += 3;
299                 while (' ' == *p || '\t' == *p)
300                         p++;
301                 if ('\0' == *p)
302                         return(er);
303                 mandoc_msg(MANDOCERR_ARGSLOST, ep->parse, ln, pos, NULL);
304                 return(er);
305         }

307         /*
308          * Build up the full string, replacing all newlines with regular
309          * whitespace.
310          */

312         sz = strlen(p + pos) + 1;
313         ep->data = mandoc_realloc(ep->data, ep->sz + sz + 1);

315         /* First invocation: nil terminate the string. */

317         if (0 == ep->sz)
318                 *ep->data = '\0';

320         ep->sz += sz;
321         strlcat(ep->data, p + pos, ep->sz + 1);
322         strlcat(ep->data, " ", ep->sz + 1);
323         return(ROFF_IGN);
324 }
```



```
326 struct eqn_node *
327 eqn_alloc(const char *name, int pos, int line, struct mparse *parse)
328 {
329         struct eqn_node *p;
330         size_t           sz;
331         const char      *end;

333         p = mandoc_calloc(1, sizeof(struct eqn_node));

335         if (name && '\0' != *name) {
336                 sz = strlen(name);
337                 assert(sz);
338                 do {
339                         sz--;
340                         end = name + (int)sz;
341                 } while (' ' == *end || '\t' == *end);
342                 p->eqn.name = mandoc_strndup(name, sz + 1);
343         }

345         p->parse = parse;
346         p->eqn.ln = line;
347         p->eqn.pos = pos;
348         p->gsize = EQN_DEFSIZE;

350         return(p);
351 }

353 enum rofferr
354 eqn_end(struct eqn_node **epp)
355 {
356         struct eqn_node *ep;
357         struct eqn_box  *root;
358         enum eqn_rest    c;

360         ep = *epp;
361         *epp = NULL;

363         ep->eqn.root = mandoc_calloc(1, sizeof(struct eqn_box));

365         root = ep->eqn.root;
366         root->type = EQN_ROOT;

368         if (0 == ep->sz)
369                 return(ROFF_IGN);

371         if (EQN_DESCOPE == (c = eqn_eqn(ep, root))) {
372                 EQN_MSG(MANDOCERR_EQNNSCOPE, ep);
373                 c = EQN_ERR;
374         }

376         return(EQN_EOF == c ? ROFF_EQN : ROFF_IGN);
377 }

379 static enum eqn_rest
380 eqn_eqn(struct eqn_node *ep, struct eqn_box *last)
381 {
382         struct eqn_box  *bp;
383         enum eqn_rest    c;

385         bp = eqn_box_alloc(ep, last);
386         bp->type = EQN_SUBEXPR;

388         while (EQN_OK == (c = eqn_box(ep, bp)))
389                 /* Spin! */ ;

391         return(c);
```

```
392 }

394 static enum eqn_rest
395 eqn_matrix(struct eqn_node *ep, struct eqn_box *last)
396 {
397         struct eqn_box  *bp;
398         const char      *start;
399         size_t           sz;
400         enum eqn_rest    c;

402         bp = eqn_box_alloc(ep, last);
403         bp->type = EQN_MATRIX;

405         if (NULL == (start = eqn_nexttok(ep, &sz))) {
406                 EQN_MSG(MANDOCERR_EQNEOF, ep);
407                 return(EQN_ERR);
408         }
409         if ( ! STRNEQ(start, sz, "{", 1)) {
410                 EQN_MSG(MANDOCERR_EQNSYNT, ep);
411                 return(EQN_ERR);
412         }

414         while (EQN_OK == (c = eqn_box(ep, bp)))
415                 switch (bp->last->pile) {
416                 case (EQNPILE_LCOL):
417                         /* FALLTHROUGH */
418                 case (EQNPILE_CCOL):
419                         /* FALLTHROUGH */
420                 case (EQNPILE_RCOL):
421                         continue;
422                 default:
423                         EQN_MSG(MANDOCERR_EQNSYNT, ep);
424                         return(EQN_ERR);
425                 };

427         if (EQN_DESCOPE != c) {
428                 if (EQN_EOF == c)
429                         EQN_MSG(MANDOCERR_EQNEOF, ep);
430                 return(EQN_ERR);
431         }

433         eqn_rewind(ep);
434         start = eqn_nexttok(ep, &sz);
435         assert(start);
436         if (STRNEQ(start, sz, "}", 1))
437                 return(EQN_OK);

439         EQN_MSG(MANDOCERR_EQNBADSCOPE, ep);
440         return(EQN_ERR);
441 }

443 static enum eqn_rest
444 eqn_list(struct eqn_node *ep, struct eqn_box *last)
445 {
446         struct eqn_box  *bp;
447         const char      *start;
448         size_t           sz;
449         enum eqn_rest    c;

451         bp = eqn_box_alloc(ep, last);
452         bp->type = EQN_LIST;

454         if (NULL == (start = eqn_nexttok(ep, &sz))) {
455                 EQN_MSG(MANDOCERR_EQNEOF, ep);
456                 return(EQN_ERR);
457         }
```



```
458         if ( ! STRNEQ(start, sz, "{", 1)) {
459                 EQN_MSG(MANDOCERR_EQNSYNT, ep);
460                 return(EQN_ERR);
461         }

463         while (EQN_DESCOPE == (c = eqn_eqn(ep, bp))) {
464                 eqn_rewind(ep);
465                 start = eqn_nexttok(ep, &sz);
466                 assert(start);
467                 if ( ! STRNEQ(start, sz, "above", 5))
468                         break;
469         }

471         if (EQN_DESCOPE != c) {
472                 if (EQN_ERR != c)
473                         EQN_MSG(MANDOCERR_EQNSCOPE, ep);
474                 return(EQN_ERR);
475         }

477         eqn_rewind(ep);
478         start = eqn_nexttok(ep, &sz);
479         assert(start);
480         if (STRNEQ(start, sz, "}", 1))
481                 return(EQN_OK);

483         EQN_MSG(MANDOCERR_EQNBADSCOPE, ep);
484         return(EQN_ERR);
485 }

487 static enum eqn_rest
488 eqn_box(struct eqn_node *ep, struct eqn_box *last)
489 {
490         size_t           sz;
491         const char      *start;
492         char            *left;
493         char             sym[64];
494         enum eqn_rest    c;
495         int              i, size;
496         struct eqn_box  *bp;

498         if (NULL == (start = eqn_nexttok(ep, &sz)))
499                 return(EQN_EOF);

501         if (STRNEQ(start, sz, "}", 1))
502                 return(EQN_DESCOPE);
503         else if (STRNEQ(start, sz, "right", 5))
504                 return(EQN_DESCOPE);
505         else if (STRNEQ(start, sz, "above", 5))
506                 return(EQN_DESCOPE);
507         else if (STRNEQ(start, sz, "mark", 4))
508                 return(EQN_OK);
509         else if (STRNEQ(start, sz, "lineup", 6))
510                 return(EQN_OK);

512         for (i = 0; i < (int)EQN__MAX; i++) {
513                 if ( ! EQNSTREQ(&eqnparts[i].str, start, sz))
514                         continue;
515                 return((*eqnparts[i].fp)(ep) ?
516                                 EQN_OK : EQN_ERR);
517         }

519         if (STRNEQ(start, sz, "{", 1)) {
520                 if (EQN_DESCOPE != (c = eqn_eqn(ep, last))) {
521                         if (EQN_ERR != c)
522                                 EQN_MSG(MANDOCERR_EQNSCOPE, ep);
523                         return(EQN_ERR);
```

```
524                 }
525                 eqn_rewind(ep);
526                 start = eqn_nexttok(ep, &sz);
527                 assert(start);
528                 if (STRNEQ(start, sz, "}", 1))
529                         return(EQN_OK);
530                 EQN_MSG(MANDOCERR_EQNBADSCOPE, ep);
531                 return(EQN_ERR);
532         }
533
534         for (i = 0; i < (int)EQNPILE__MAX; i++) {
535                 if ( ! EQNSTREQ(&eqnpiles[i], start, sz))
536                         continue;
537                 if (EQN_OK == (c = eqn_list(ep, last)))
538                         last->last->pile = (enum eqn_pilet)i;
539                 return(c);
540         }
541
542         if (STRNEQ(start, sz, "matrix", 6))
543                 return(eqn_matrix(ep, last));
544
545         if (STRNEQ(start, sz, "left", 4)) {
546                 if (NULL == (start = eqn_nexttok(ep, &sz))) {
547                         EQN_MSG(MANDOCERR_EQNEOF, ep);
548                         return(EQN_ERR);
549                 }
550                 left = mandoc_strndup(start, sz);
551                 c = eqn_eqn(ep, last);
552                 if (last->last)
553                         last->last->left = left;
554                 else
555                         free(left);
556                 if (EQN_DESCOPE != c)
557                         return(c);
558                 assert(last->last);
559                 eqn_rewind(ep);
560                 start = eqn_nexttok(ep, &sz);
561                 assert(start);
562                 if ( ! STRNEQ(start, sz, "right", 5))
563                         return(EQN_DESCOPE);
564                 if (NULL == (start = eqn_nexttok(ep, &sz))) {
565                         EQN_MSG(MANDOCERR_EQNEOF, ep);
566                         return(EQN_ERR);
567                 }
568                 last->last->right = mandoc_strndup(start, sz);
569                 return(EQN_OK);
570         }
571
572         for (i = 0; i < (int)EQNPOS__MAX; i++) {
573                 if ( ! EQNSTREQ(&eqnposs[i], start, sz))
574                         continue;
575                 if (NULL == last->last) {
576                         EQN_MSG(MANDOCERR_EQNSYNT, ep);
577                         return(EQN_ERR);
578                 }
579                 last->last->pos = (enum eqn_post)i;
580                 if (EQN_EOF == (c = eqn_box(ep, last))) {
581                         EQN_MSG(MANDOCERR_EQNEOF, ep);
582                         return(EQN_ERR);
583                 }
584                 return(c);
585         }
586
587         for (i = 0; i < (int)EQNMARK__MAX; i++) {
588                 if ( ! EQNSTREQ(&eqnmarks[i], start, sz))
589                         continue;
```



```
590                 if (NULL == last->last) {
591                         EQN_MSG(MANDOCERR_EQNSYNT, ep);
592                         return(EQN_ERR);
593                 }
594                 last->last->mark = (enum eqn_markt)i;
595                 if (EQN_EOF == (c = eqn_box(ep, last))) {
596                         EQN_MSG(MANDOCERR_EQNEOF, ep);
597                         return(EQN_ERR);
598                 }
599                 return(c);
600         }
601
602         for (i = 0; i < (int)EQNFONT__MAX; i++) {
603                 if ( ! EQNSTREQ(&eqnfonts[i], start, sz))
604                         continue;
605                 if (EQN_EOF == (c = eqn_box(ep, last))) {
606                         EQN_MSG(MANDOCERR_EQNEOF, ep);
607                         return(EQN_ERR);
608                 } else if (EQN_OK == c)
609                         last->last->font = (enum eqn_fontt)i;
610                 return(c);
611         }
612
613         if (STRNEQ(start, sz, "size", 4)) {
614                 if (NULL == (start = eqn_nexttok(ep, &sz))) {
615                         EQN_MSG(MANDOCERR_EQNEOF, ep);
616                         return(EQN_ERR);
617                 }
618                 size = mandoc_strntoi(start, sz, 10);
619                 if (EQN_EOF == (c = eqn_box(ep, last))) {
620                         EQN_MSG(MANDOCERR_EQNEOF, ep);
621                         return(EQN_ERR);
622                 } else if (EQN_OK != c)
623                         return(c);
624                 last->last->size = size;
625         }
626
627         bp = eqn_box_alloc(ep, last);
628         bp->type = EQN_TEXT;
629         for (i = 0; i < (int)EQNSYM__MAX; i++)
630                 if (EQNSTREQ(&eqnsyms[i].str, start, sz)) {
631                         sym[63] = '\0';
632                         snprintf(sym, 62, "\\[%s]", eqnsyms[i].sym);
633                         bp->text = mandoc_strdup(sym);
634                         return(EQN_OK);
635                 }
636
637         bp->text = mandoc_strndup(start, sz);
638         return(EQN_OK);
639 }
640
641 void
642 eqn_free(struct eqn_node *p)
643 {
644         int              i;
645
646         eqn_box_free(p->eqn.root);
647
648         for (i = 0; i < (int)p->defsz; i++) {
649                 free(p->defs[i].key);
650                 free(p->defs[i].val);
651         }
652
653         free(p->eqn.name);
654         free(p->data);
655         free(p->defs);
```

```
656         free(p);
657 }

659 static struct eqn_box *
660 eqn_box_alloc(struct eqn_node *ep, struct eqn_box *parent)
661 {
662         struct eqn_box  *bp;

664         bp = mandoc_calloc(1, sizeof(struct eqn_box));
665         bp->parent = parent;
666         bp->size = ep->gsize;

668         if (NULL == parent->first)
669                 parent->first = bp;
670         else
671                 parent->last->next = bp;

673         parent->last = bp;
674         return(bp);
675 }

677 static void
678 eqn_box_free(struct eqn_box *bp)
679 {

681         if (bp->first)
682                 eqn_box_free(bp->first);
683         if (bp->next)
684                 eqn_box_free(bp->next);

686         free(bp->text);
687         free(bp->left);
688         free(bp->right);
689         free(bp);
690 }

692 static const char *
693 eqn_nextrawtok(struct eqn_node *ep, size_t *sz)
694 {

696         return(eqn_next(ep, '"', sz, 0));
697 }

699 static const char *
700 eqn_nexttok(struct eqn_node *ep, size_t *sz)
701 {

703         return(eqn_next(ep, '"', sz, 1));
704 }

706 static void
707 eqn_rewind(struct eqn_node *ep)
708 {

710         ep->cur = ep->rew;
711 }

713 static const char *
714 eqn_next(struct eqn_node *ep, char quote, size_t *sz, int repl)
715 {
716         char            *start, *next;
717         int              q, diff, lim;
718         size_t           ssz, dummy;
719         struct eqn_def  *def;

721         if (NULL == sz)
```



```
722                 sz = &dummy;

724         lim = 0;
725         ep->rew = ep->cur;
726 again:
727         /* Prevent self-definitions. */

729         if (lim >= EQN_NEST_MAX) {
730                 EQN_MSG(MANDOCERR_ROFFLOOP, ep);
731                 return(NULL);
732         }

734         ep->cur = ep->rew;
735         start = &ep->data[(int)ep->cur];
736         q = 0;

738         if ('\0' == *start)
739                 return(NULL);

741         if (quote == *start) {
742                 ep->cur++;
743                 q = 1;
744         }

746         start = &ep->data[(int)ep->cur];

748         if ( ! q) {
749                 if ('{' == *start || '}' == *start)
750                         ssz = 1;
751                 else
752                         ssz = strcspn(start + 1, " ^~\"{}\t") + 1;
753                 next = start + (int)ssz;
754                 if ('\0' == *next)
755                         next = NULL;
756         } else
757                 next = strchr(start, quote);

759         if (NULL != next) {
760                 *sz = (size_t)(next - start);
761                 ep->cur += *sz;
762                 if (q)
763                         ep->cur++;
764                 while (' ' == ep->data[(int)ep->cur] ||
765                                 '\t' == ep->data[(int)ep->cur] ||
766                                 '^' == ep->data[(int)ep->cur] ||
767                                 '~' == ep->data[(int)ep->cur])
768                         ep->cur++;
769         } else {
770                 if (q)
771                         EQN_MSG(MANDOCERR_BADQUOTE, ep);
772                 next = strchr(start, '\0');
773                 *sz = (size_t)(next - start);
774                 ep->cur += *sz;
775         }

777         /* Quotes aren't expanded for values. */

779         if (q || ! repl)
780                 return(start);

782         if (NULL != (def = eqn_def_find(ep, start, *sz))) {
783                 diff = def->valsz - *sz;

785                 if (def->valsz > *sz) {
786                         ep->sz += diff;
787                         ep->data = mandoc_realloc(ep->data, ep->sz + 1);
```

```
788                         ep->data[ep->sz] = '\0';
789                         start = &ep->data[(int)ep->rew];
790                 }

792                 diff = def->valsz - *sz;
793                 memmove(start + *sz + diff, start + *sz,
794                                 (strlen(start) - *sz) + 1);
795                 memcpy(start, def->val, def->valsz);
796                 goto again;
797         }

799         return(start);
800 }

802 static int
803 eqn_do_ign1(struct eqn_node *ep)
804 {

806         if (NULL == eqn_nextrawtok(ep, NULL))
807                 EQN_MSG(MANDOCERR_EQNEOF, ep);
808         else
809                 return(1);

811         return(0);
812 }

814 static int
815 eqn_do_ign2(struct eqn_node *ep)
816 {

818         if (NULL == eqn_nextrawtok(ep, NULL))
819                 EQN_MSG(MANDOCERR_EQNEOF, ep);
820         else if (NULL == eqn_nextrawtok(ep, NULL))
821                 EQN_MSG(MANDOCERR_EQNEOF, ep);
822         else
823                 return(1);

825         return(0);
826 }

828 static int
829 eqn_do_tdefine(struct eqn_node *ep)
830 {

832         if (NULL == eqn_nextrawtok(ep, NULL))
833                 EQN_MSG(MANDOCERR_EQNEOF, ep);
834         else if (NULL == eqn_next(ep, ep->data[(int)ep->cur], NULL, 0))
835                 EQN_MSG(MANDOCERR_EQNEOF, ep);
836         else
837                 return(1);

839         return(0);
840 }

842 static int
843 eqn_do_define(struct eqn_node *ep)
844 {
845         const char      *start;
846         size_t           sz;
847         struct eqn_def  *def;
848         int              i;

850         if (NULL == (start = eqn_nextrawtok(ep, &sz))) {
851                 EQN_MSG(MANDOCERR_EQNEOF, ep);
852                 return(0);
853         }
```



```
855         /*
856          * Search for a key that already exists.
857          * Create a new key if none is found.
858          */

860         if (NULL == (def = eqn_def_find(ep, start, sz))) {
861                 /* Find holes in string array. */
862                 for (i = 0; i < (int)ep->defsz; i++)
863                         if (0 == ep->defs[i].keysz)
864                                 break;

866                 if (i == (int)ep->defsz) {
867                         ep->defsz++;
868                         ep->defs = mandoc_realloc
869                                 (ep->defs, ep->defsz *
870                                  sizeof(struct eqn_def));
871                         ep->defs[i].key = ep->defs[i].val = NULL;
872                 }

874                 ep->defs[i].keysz = sz;
875                 ep->defs[i].key = mandoc_realloc
876                         (ep->defs[i].key, sz + 1);

878                 memcpy(ep->defs[i].key, start, sz);
879                 ep->defs[i].key[(int)sz] = '\0';
880                 def = &ep->defs[i];
881         }

883         start = eqn_next(ep, ep->data[(int)ep->cur], &sz, 0);

885         if (NULL == start) {
886                 EQN_MSG(MANDOCERR_EQNEOF, ep);
887                 return(0);
888         }

890         def->valsz = sz;
891         def->val = mandoc_realloc(def->val, sz + 1);
892         memcpy(def->val, start, sz);
893         def->val[(int)sz] = '\0';
894         return(1);
895 }

897 static int
898 eqn_do_gfont(struct eqn_node *ep)
899 {

901         if (NULL == eqn_nextrawtok(ep, NULL)) {
902                 EQN_MSG(MANDOCERR_EQNEOF, ep);
903                 return(0);
904         }
905         return(1);
906 }

908 static int
909 eqn_do_gsize(struct eqn_node *ep)
910 {
911         const char      *start;
912         size_t           sz;

914         if (NULL == (start = eqn_nextrawtok(ep, &sz))) {
915                 EQN_MSG(MANDOCERR_EQNEOF, ep);
916                 return(0);
917         }
918         ep->gsize = mandoc_strntoi(start, sz, 10);
919         return(1);
```

```
 920 }

 922 static int
 923 eqn_do_undef(struct eqn_node *ep)
 924 {
 925         const char      *start;
 926         struct eqn_def  *def;
 927         size_t           sz;

 929         if (NULL == (start = eqn_nextrawtok(ep, &sz))) {
 930                 EQN_MSG(MANDOCERR_EQNEOF, ep);
 931                 return(0);
 932         } else if (NULL != (def = eqn_def_find(ep, start, sz)))
 933                 def->keysz = 0;

 935         return(1);
 936 }

 938 static struct eqn_def *
 939 eqn_def_find(struct eqn_node *ep, const char *key, size_t sz)
 940 {
 941         int              i;

 943         for (i = 0; i < (int)ep->defsz; i++)
 944                 if (ep->defs[i].keysz && STRNEQ(ep->defs[i].key,
 945                                         ep->defs[i].keysz, key, sz))
 946                         return(&ep->defs[i]);

 948         return(NULL);
 949 }
```

```
**********************************************************
    1996 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/eqn_html.c
Initial import of man functionality.
**********************************************************
```

```
  1 /*      $Id: eqn_html.c,v 1.2 2011/07/24 10:09:03 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 #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  const enum htmltag fontmap[EQNFONT__MAX] = {
 31         TAG_SPAN, /* EQNFONT_NONE */
 32         TAG_SPAN, /* EQNFONT_ROMAN */
 33         TAG_B, /* EQNFONT_BOLD */
 34         TAG_B, /* EQNFONT_FAT */
 35         TAG_I /* EQNFONT_ITALIC */
 36 };


 39 static void     eqn_box(struct html *, const struct eqn_box *);

 41 void
 42 print_eqn(struct html *p, const struct eqn *ep)
 43 {
 44         struct htmlpair  tag;
 45         struct tag      *t;

 47         PAIR_CLASS_INIT(&tag, "eqn");
 48         t = print_otag(p, TAG_SPAN, 1, &tag);

 50         p->flags |= HTML_NONOSPACE;
 51         eqn_box(p, ep->root);
 52         p->flags &= ~HTML_NONOSPACE;

 54         print_tagq(p, t);
 55 }

 57 static void
 58 eqn_box(struct html *p, const struct eqn_box *bp)
 59 {
 60         struct tag      *t;
```



```
 62         t = EQNFONT_NONE == bp->font ? NULL :
 63                 print_otag(p, fontmap[(int)bp->font], 0, NULL);

 65         if (bp->left)
 66                 print_text(p, bp->left);
 67
 68         if (bp->text)
 69                 print_text(p, bp->text);

 71         if (bp->first)
 72                 eqn_box(p, bp->first);

 74         if (NULL != t)
 75                 print_tagq(p, t);
 76         if (bp->right)
 77                 print_text(p, bp->right);

 79         if (bp->next)
 80                 eqn_box(p, bp->next);
 81 }
```

```
**********************************************************
    2006 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/eqn_term.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: eqn_term.c,v 1.4 2011/07/24 10:09:03 kristaps Exp $ */
/*
 * Copyright (c) 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 <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

static  const enum termfont fontmap[EQNFONT__MAX] = {
        TERMFONT_NONE, /* EQNFONT_NONE */
        TERMFONT_NONE, /* EQNFONT_ROMAN */
        TERMFONT_BOLD, /* EQNFONT_BOLD */
        TERMFONT_BOLD, /* EQNFONT_FAT */
        TERMFONT_UNDER /* EQNFONT_ITALIC */
};

static void     eqn_box(struct termp *, const struct eqn_box *);

void
term_eqn(struct termp *p, const struct eqn *ep)
{

        p->flags |= TERMP_NONOSPACE;
        eqn_box(p, ep->root);
        term_word(p, " ");
        p->flags &= ~TERMP_NONOSPACE;
}

static void
eqn_box(struct termp *p, const struct eqn_box *bp)
{

        if (EQNFONT_NONE != bp->font)
                term_fontpush(p, fontmap[(int)bp->font]);
        if (bp->left)
                term_word(p, bp->left);
        if (EQN_SUBEXPR == bp->type)
                term_word(p, "(");

        if (bp->text)
```



```
                term_word(p, bp->text);

        if (bp->first)
                eqn_box(p, bp->first);

        if (EQN_SUBEXPR == bp->type)
                term_word(p, ")");
        if (bp->right)
                term_word(p, bp->right);
        if (EQNFONT_NONE != bp->font)
                term_fontpop(p);

        if (bp->next)
                eqn_box(p, bp->next);
}
```

```
**********************************************************
   14142 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/html.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: html.c,v 1.150 2011/10/05 21:35:17 kristaps Exp $ */
/*
 * Copyright (c) 2008, 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 <sys/types.h>

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

#include "mandoc.h"
#include "libmandoc.h"
#include "out.h"
#include "html.h"
#include "main.h"

struct  htmldata {
        const char       *name;
        int               flags;
#define HTML_CLRLINE     (1 << 0)
#define HTML_NOSTACK     (1 << 1)
#define HTML_AUTOCLOSE   (1 << 2) /* Tag has auto-closure. */
};

static  const struct htmldata htmltags[TAG_MAX] = {
        {"html",        HTML_CLRLINE}, /* TAG_HTML */
        {"head",        HTML_CLRLINE}, /* TAG_HEAD */
        {"body",        HTML_CLRLINE}, /* TAG_BODY */
        {"meta",        HTML_CLRLINE | HTML_NOSTACK | HTML_AUTOCLOSE}, /* TAG_ME
        {"title",       HTML_CLRLINE}, /* TAG_TITLE */
        {"div",         HTML_CLRLINE}, /* TAG_DIV */
        {"h1",          0}, /* TAG_H1 */
        {"h2",          0}, /* TAG_H2 */
        {"span",        0}, /* TAG_SPAN */
        {"link",        HTML_CLRLINE | HTML_NOSTACK | HTML_AUTOCLOSE}, /* TAG_LI
        {"br",          HTML_CLRLINE | HTML_NOSTACK | HTML_AUTOCLOSE}, /* TAG_BR
        {"a",           0}, /* TAG_A */
        {"table",       HTML_CLRLINE}, /* TAG_TABLE */
        {"tbody",       HTML_CLRLINE}, /* TAG_TBODY */
```



```
        {"col",         HTML_CLRLINE | HTML_NOSTACK | HTML_AUTOCLOSE}, /* TAG_CO
        {"tr",          HTML_CLRLINE}, /* TAG_TR */
        {"td",          HTML_CLRLINE}, /* TAG_TD */
        {"li",          HTML_CLRLINE}, /* TAG_LI */
        {"ul",          HTML_CLRLINE}, /* TAG_UL */
        {"ol",          HTML_CLRLINE}, /* TAG_OL */
        {"dl",          HTML_CLRLINE}, /* TAG_DL */
        {"dt",          HTML_CLRLINE}, /* TAG_DT */
        {"dd",          HTML_CLRLINE}, /* TAG_DD */
        {"blockquote",  HTML_CLRLINE}, /* TAG_BLOCKQUOTE */
        {"p",           HTML_CLRLINE | HTML_NOSTACK | HTML_AUTOCLOSE}, /* TAG_P
        {"pre",         HTML_CLRLINE }, /* TAG_PRE */
        {"b",           0 }, /* TAG_B */
        {"i",           0 }, /* TAG_I */
        {"code",        0 }, /* TAG_CODE */
        {"small",       0 }, /* TAG_SMALL */
};

static  const char      *const htmlattrs[ATTR_MAX] = {
        "http-equiv", /* ATTR_HTTPEQUIV */
        "content", /* ATTR_CONTENT */
        "name", /* ATTR_NAME */
        "rel", /* ATTR_REL */
        "href", /* ATTR_HREF */
        "type", /* ATTR_TYPE */
        "media", /* ATTR_MEDIA */
        "class", /* ATTR_CLASS */
        "style", /* ATTR_STYLE */
        "width", /* ATTR_WIDTH */
        "id", /* ATTR_ID */
        "summary", /* ATTR_SUMMARY */
        "align", /* ATTR_ALIGN */
        "colspan", /* ATTR_COLSPAN */
};

static  const char      *const roffscales[SCALE_MAX] = {
        "cm", /* SCALE_CM */
        "in", /* SCALE_IN */
        "pc", /* SCALE_PC */
        "pt", /* SCALE_PT */
        "em", /* SCALE_EM */
        "em", /* SCALE_MM */
        "ex", /* SCALE_EN */
        "ex", /* SCALE_BU */
        "em", /* SCALE_VS */
        "ex", /* SCALE_FS */
};

static  void     bufncat(struct html *, const char *, size_t);
static  void     print_ctag(struct html *, enum htmltag);
static  int      print_encode(struct html *, const char *, int);
static  void     print_metaf(struct html *, enum mandoc_esc);
static  void     print_attr(struct html *, const char *, const char *);
static  void     *ml_alloc(char *, enum htmltype);

static void *
ml_alloc(char *outopts, enum htmltype type)
{
        struct html     *h;
        const char      *toks[5];
        char            *v;

        toks[0] = "style";
        toks[1] = "man";
        toks[2] = "includes";
        toks[3] = "fragment";
```

```
128         toks[4] = NULL;

130         h = mandoc_calloc(1, sizeof(struct html));

132         h->type = type;
133         h->tags.head = NULL;
134         h->symtab = mchars_alloc();

136         while (outopts && *outopts)
137                 switch (getsubopt(&outopts, UNCONST(toks), &v)) {
138                 case (0):
139                         h->style = v;
140                         break;
141                 case (1):
142                         h->base_man = v;
143                         break;
144                 case (2):
145                         h->base_includes = v;
146                         break;
147                 case (3):
148                         h->oflags |= HTML_FRAGMENT;
149                         break;
150                 default:
151                         break;
152                 }

154         return(h);
155 }

157 void *
158 html_alloc(char *outopts)
159 {

161         return(ml_alloc(outopts, HTML_HTML_4_01_STRICT));
162 }


165 void *
166 xhtml_alloc(char *outopts)
167 {

169         return(ml_alloc(outopts, HTML_XHTML_1_0_STRICT));
170 }


173 void
174 html_free(void *p)
175 {
176         struct tag      *tag;
177         struct html     *h;

179         h = (struct html *)p;

181         while ((tag = h->tags.head) != NULL) {
182                 h->tags.head = tag->next;
183                 free(tag);
184         }
185
186         if (h->symtab)
187                 mchars_free(h->symtab);

189         free(h);
190 }


193 void
```



```
194 print_gen_head(struct html *h)
195 {
196         struct htmlpair  tag[4];

198         tag[0].key = ATTR_HTTPEQUIV;
199         tag[0].val = "Content-Type";
200         tag[1].key = ATTR_CONTENT;
201         tag[1].val = "text/html; charset=utf-8";
202         print_otag(h, TAG_META, 2, tag);

204         tag[0].key = ATTR_NAME;
205         tag[0].val = "resource-type";
206         tag[1].key = ATTR_CONTENT;
207         tag[1].val = "document";
208         print_otag(h, TAG_META, 2, tag);

210         if (h->style) {
211                 tag[0].key = ATTR_REL;
212                 tag[0].val = "stylesheet";
213                 tag[1].key = ATTR_HREF;
214                 tag[1].val = h->style;
215                 tag[2].key = ATTR_TYPE;
216                 tag[2].val = "text/css";
217                 tag[3].key = ATTR_MEDIA;
218                 tag[3].val = "all";
219                 print_otag(h, TAG_LINK, 4, tag);
220         }
221 }

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_FONT):
239                 /* FALLTHROUGH */
240         case (ESCAPE_FONTROMAN):
241                 font = HTMLFONT_NONE;
242                 break;
243         default:
244                 abort();
245                 /* NOTREACHED */
246         }

248         if (h->metaf) {
249                 print_tagq(h, h->metaf);
250                 h->metaf = NULL;
251         }

253         h->metal = h->metac;
254         h->metac = font;

256         if (HTMLFONT_NONE != font)
257                 h->metaf = HTMLFONT_BOLD == font ?
258                         print_otag(h, TAG_B, 0, NULL) :
259                         print_otag(h, TAG_I, 0, NULL);
```

```
}

int
html_strlen(const char *cp)
{
        int              ssz, sz;
        const char      *seq, *p;

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

        sz = 0;
        while (NULL != (p = strchr(cp, '\\'))) {
                sz += (int)(p - cp);
                ++cp;
                switch (mandoc_escape(&cp, &seq, &ssz)) {
                case (ESCAPE_ERROR):
                        return(sz);
                case (ESCAPE_UNICODE):
                        /* FALLTHROUGH */
                case (ESCAPE_NUMBERED):
                        /* FALLTHROUGH */
                case (ESCAPE_SPECIAL):
                        sz++;
                        break;
                default:
                        break;
                }
        }

        assert(sz >= 0);
        return(sz + strlen(cp));
}

static int
print_encode(struct html *h, const char *p, int norecurse)
{
        size_t           sz;
        int              c, len, nospace;
        const char      *seq;
        enum mandoc_esc  esc;
        static const char rejs[6] = { '\\', '<', '>', '&', ASCII_HYPH, '\0' };

        nospace = 0;

        while ('\0' != *p) {
                sz = strcspn(p, rejs);

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

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

                switch (*p++) {
                case ('<'):
                        printf("&lt;");
                        continue;
                case ('>'):
                        printf("&gt;");
                        continue;
```



```
                case ('&'):
                        printf("&amp;");
                        continue;
                case (ASCII_HYPH):
                        putchar('-');
                        continue;
                default:
                        break;
                }

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

                switch (esc) {
                case (ESCAPE_UNICODE):
                        /* Skip passed "u" header. */
                        c = mchars_num2uc(seq + 1, len - 1);
                        if ('\0' != c)
                                printf("&#x%x;", c);
                        break;
                case (ESCAPE_NUMBERED):
                        c = mchars_num2char(seq, len);
                        if ('\0' != c)
                                putchar(c);
                        break;
                case (ESCAPE_SPECIAL):
                        c = mchars_spec2cp(h->symtab, seq, len);
                        if (c > 0)
                                printf("&#%d;", c);
                        else if (-1 == c && 1 == len)
                                putchar((int)*seq);
                        break;
                case (ESCAPE_FONT):
                        /* FALLTHROUGH */
                case (ESCAPE_FONTPREV):
                        /* FALLTHROUGH */
                case (ESCAPE_FONTBOLD):
                        /* FALLTHROUGH */
                case (ESCAPE_FONTITALIC):
                        /* FALLTHROUGH */
                case (ESCAPE_FONTROMAN):
                        if (norecurse)
                                break;
                        print_metaf(h, esc);
                        break;
                case (ESCAPE_NOSPACE):
                        if ('\0' == *p)
                                nospace = 1;
                        break;
                default:
                        break;
                }
        }

        return(nospace);
}


static void
print_attr(struct html *h, const char *key, const char *val)
{
        printf(" %s=\"", key);
        (void)print_encode(h, val, 1);
        putchar('\"');
}
```

```
394 struct tag *
395 print_otag(struct html *h, enum htmltag tag,
396                 int sz, const struct htmlpair *p)
397 {
398         int              i;
399         struct tag      *t;

401         /* Push this tags onto the stack of open scopes. */

403         if ( ! (HTML_NOSTACK & htmltags[tag].flags)) {
404                 t = mandoc_malloc(sizeof(struct tag));
405                 t->tag = tag;
406                 t->next = h->tags.head;
407                 h->tags.head = t;
408         } else
409                 t = NULL;

411         if ( ! (HTML_NOSPACE & h->flags))
412                 if ( ! (HTML_CLRLINE & htmltags[tag].flags)) {
413                         /* Manage keeps! */
414                         if ( ! (HTML_KEEP & h->flags)) {
415                                 if (HTML_PREKEEP & h->flags)
416                                         h->flags |= HTML_KEEP;
417                                 putchar(' ');
418                         } else
419                                 printf(" ");
420                 }

422         if ( ! (h->flags & HTML_NONOSPACE))
423                 h->flags &= ~HTML_NOSPACE;
424         else
425                 h->flags |= HTML_NOSPACE;

427         /* Print out the tag name and attributes. */

429         printf("<%s", htmltags[tag].name);
430         for (i = 0; i < sz; i++)
431                 print_attr(h, htmlattrs[p[i].key], p[i].val);

433         /* Add non-overridable attributes. */

435         if (TAG_HTML == tag && HTML_XHTML_1_0_STRICT == h->type) {
436                 print_attr(h, "xmlns", "http://www.w3.org/1999/xhtml");
437                 print_attr(h, "xml:lang", "en");
438                 print_attr(h, "lang", "en");
439         }

441         /* Accommodate for XML "well-formed" singleton escaping. */

443         if (HTML_AUTOCLOSE & htmltags[tag].flags)
444                 switch (h->type) {
445                 case (HTML_XHTML_1_0_STRICT):
446                         putchar('/');
447                         break;
448                 default:
449                         break;
450                 }

452         putchar('>');

454         h->flags |= HTML_NOSPACE;

456         if ((HTML_AUTOCLOSE | HTML_CLRLINE) & htmltags[tag].flags)
457                 putchar('\n');
```



```
459         return(t);
460 }


463 static void
464 print_ctag(struct html *h, enum htmltag tag)
465 {
466
467         printf("</%s>", htmltags[tag].name);
468         if (HTML_CLRLINE & htmltags[tag].flags) {
469                 h->flags |= HTML_NOSPACE;
470                 putchar('\n');
471         }
472 }

474 void
475 print_gen_decls(struct html *h)
476 {
477         const char      *doctype;
478         const char      *dtd;
479         const char      *name;

481         switch (h->type) {
482         case (HTML_HTML_4_01_STRICT):
483                 name = "HTML";
484                 doctype = "-//W3C//DTD HTML 4.01//EN";
485                 dtd = "http://www.w3.org/TR/html4/strict.dtd";
486                 break;
487         default:
488                 puts("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
489                 name = "html";
490                 doctype = "-//W3C//DTD XHTML 1.0 Strict//EN";
491                 dtd = "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd";
492                 break;
493         }

495         printf("<!DOCTYPE %s PUBLIC \"%s\" \"%s\">\n",
496                         name, doctype, dtd);
497 }

499 void
500 print_text(struct html *h, const char *word)
501 {

503         if ( ! (HTML_NOSPACE & h->flags)) {
504                 /* Manage keeps! */
505                 if ( ! (HTML_KEEP & h->flags)) {
506                         if (HTML_PREKEEP & h->flags)
507                                 h->flags |= HTML_KEEP;
508                         putchar(' ');
509                 } else
510                         printf(" ");
511         }

513         assert(NULL == h->metaf);
514         if (HTMLFONT_NONE != h->metac)
515                 h->metaf = HTMLFONT_BOLD == h->metac ?
516                         print_otag(h, TAG_B, 0, NULL) :
517                         print_otag(h, TAG_I, 0, NULL);

519         assert(word);
520         if ( ! print_encode(h, word, 0)) {
521                 if ( ! (h->flags & HTML_NONOSPACE))
522                         h->flags &= ~HTML_NOSPACE;
523         } else
```

```
524                 h->flags |= HTML_NOSPACE;

526         if (h->metaf) {
527                 print_tagq(h, h->metaf);
528                 h->metaf = NULL;
529         }

531         h->flags &= ~HTML_IGNDELIM;
532 }


535 void
536 print_tagq(struct html *h, const struct tag *until)
537 {
538         struct tag      *tag;

540         while ((tag = h->tags.head) != NULL) {
541                 /*
542                  * Remember to close out and nullify the current
543                  * meta-font and table, if applicable.
544                  */
545                 if (tag == h->metaf)
546                         h->metaf = NULL;
547                 if (tag == h->tblt)
548                         h->tblt = NULL;
549                 print_ctag(h, tag->tag);
550                 h->tags.head = tag->next;
551                 free(tag);
552                 if (until && tag == until)
553                         return;
554         }
555 }


558 void
559 print_stagq(struct html *h, const struct tag *suntil)
560 {
561         struct tag      *tag;

563         while ((tag = h->tags.head) != NULL) {
564                 if (suntil && tag == suntil)
565                         return;
566                 /*
567                  * Remember to close out and nullify the current
568                  * meta-font and table, if applicable.
569                  */
570                 if (tag == h->metaf)
571                         h->metaf = NULL;
572                 if (tag == h->tblt)
573                         h->tblt = NULL;
574                 print_ctag(h, tag->tag);
575                 h->tags.head = tag->next;
576                 free(tag);
577         }
578 }

580 void
581 bufinit(struct html *h)
582 {

584         h->buf[0] = '\0';
585         h->buflen = 0;
586 }

588 void
589 bufcat_style(struct html *h, const char *key, const char *val)
```



```
590 {

592         bufcat(h, key);
593         bufcat(h, ":");
594         bufcat(h, val);
595         bufcat(h, ";");
596 }

598 void
599 bufcat(struct html *h, const char *p)
600 {

602         h->buflen = strlcat(h->buf, p, BUFSIZ);
603         assert(h->buflen < BUFSIZ);
604 }

606 void
607 bufcat_fmt(struct html *h, const char *fmt, ...)
608 {
609         va_list          ap;

611         va_start(ap, fmt);
612         (void)vsnprintf(h->buf + (int)h->buflen,
613                         BUFSIZ - h->buflen - 1, fmt, ap);
614         va_end(ap);
615         h->buflen = strlen(h->buf);
616 }

618 static void
619 bufncat(struct html *h, const char *p, size_t sz)
620 {

622         assert(h->buflen + sz + 1 < BUFSIZ);
623         strncat(h->buf, p, sz);
624         h->buflen += sz;
625 }

627 void
628 buffmt_includes(struct html *h, const char *name)
629 {
630         const char      *p, *pp;

632         pp = h->base_includes;
633
634         bufinit(h);
635         while (NULL != (p = strchr(pp, '%'))) {
636                 bufncat(h, pp, (size_t)(p - pp));
637                 switch (*(p + 1)) {
638                 case('I'):
639                         bufcat(h, name);
640                         break;
641                 default:
642                         bufncat(h, p, 2);
643                         break;
644                 }
645                 pp = p + 2;
646         }
647         if (pp)
648                 bufcat(h, pp);
649 }

651 void
652 buffmt_man(struct html *h,
653                 const char *name, const char *sec)
654 {
655         const char      *p, *pp;
```

```
657         pp = h->base_man;
658
659         bufinit(h);
660         while (NULL != (p = strchr(pp, '%'))) {
661                 bufncat(h, pp, (size_t)(p - pp));
662                 switch (*(p + 1)) {
663                 case('S'):
664                         bufcat(h, sec ? sec : "1");
665                         break;
666                 case('N'):
667                         bufcat_fmt(h, name);
668                         break;
669                 default:
670                         bufncat(h, p, 2);
671                         break;
672                 }
673                 pp = p + 2;
674         }
675         if (pp)
676                 bufcat(h, pp);
677 }

679 void
680 bufcat_su(struct html *h, const char *p, const struct roffsu *su)
681 {
682         double           v;

684         v = su->scale;
685         if (SCALE_MM == su->unit && 0.0 == (v /= 100.0))
686                 v = 1.0;

688         bufcat_fmt(h, "%s: %.2f%s;", p, v, roffscales[su->unit]);
689 }

691 void
692 bufcat_id(struct html *h, const char *src)
693 {

695         /* Cf. <http://www.w3.org/TR/html4/types.html#h-6.2>. */

697         while ('\0' != *src)
698                 bufcat_fmt(h, "%.2x", *src++);
699 }
```

```
************************************************************
    4165 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/html.h
Initial import of man functionality.
************************************************************
```

```
/*      $Id: html.h,v 1.47 2011/10/05 21:35:17 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.
 */
#ifndef HTML_H
#define HTML_H

__BEGIN_DECLS

enum    htmltag {
        TAG_HTML,
        TAG_HEAD,
        TAG_BODY,
        TAG_META,
        TAG_TITLE,
        TAG_DIV,
        TAG_H1,
        TAG_H2,
        TAG_SPAN,
        TAG_LINK,
        TAG_BR,
        TAG_A,
        TAG_TABLE,
        TAG_TBODY,
        TAG_COL,
        TAG_TR,
        TAG_TD,
        TAG_LI,
        TAG_UL,
        TAG_OL,
        TAG_DL,
        TAG_DT,
        TAG_DD,
        TAG_BLOCKQUOTE,
        TAG_P,
        TAG_PRE,
        TAG_B,
        TAG_I,
        TAG_CODE,
        TAG_SMALL,
        TAG_MAX
};

enum    htmlattr {
        ATTR_HTTPEQUIV,
        ATTR_CONTENT,
        ATTR_NAME,
        ATTR_REL,
        ATTR_HREF,
```



```
        ATTR_TYPE,
        ATTR_MEDIA,
        ATTR_CLASS,
        ATTR_STYLE,
        ATTR_WIDTH,
        ATTR_ID,
        ATTR_SUMMARY,
        ATTR_ALIGN,
        ATTR_COLSPAN,
        ATTR_MAX
};

enum    htmlfont {
        HTMLFONT_NONE = 0,
        HTMLFONT_BOLD,
        HTMLFONT_ITALIC,
        HTMLFONT_MAX
};

struct  tag {
        struct tag       *next;
        enum htmltag      tag;
};

struct tagq {
        struct tag       *head;
};

struct  htmlpair {
        enum htmlattr     key;
        const char       *val;
};

#define PAIR_INIT(p, t, v) \
        do { \
                (p)->key = (t); \
                (p)->val = (v); \
        } while (/* CONSTCOND */ 0)

#define PAIR_ID_INIT(p, v)      PAIR_INIT(p, ATTR_ID, v)
#define PAIR_CLASS_INIT(p, v)   PAIR_INIT(p, ATTR_CLASS, v)
#define PAIR_HREF_INIT(p, v)    PAIR_INIT(p, ATTR_HREF, v)
#define PAIR_STYLE_INIT(p, h)   PAIR_INIT(p, ATTR_STYLE, (h)->buf)
#define PAIR_SUMMARY_INIT(p, v) PAIR_INIT(p, ATTR_SUMMARY, v)

enum    htmltype {
        HTML_HTML_4_01_STRICT,
        HTML_XHTML_1_0_STRICT
};

struct  html {
        int               flags;
#define HTML_NOSPACE     (1 << 0) /* suppress next space */
#define HTML_IGNDELIM    (1 << 1)
#define HTML_KEEP        (1 << 2)
#define HTML_PREKEEP     (1 << 3)
#define HTML_NONOSPACE   (1 << 4) /* never add spaces */
#define HTML_LITERAL     (1 << 5) /* literal (e.g., <PRE>) context */
        struct tagq       tags; /* stack of open tags */
        struct rofftbl    tbl; /* current table */
        struct tag       *tblt; /* current open table scope */
        struct mchars    *symtab; /* character-escapes */
        char             *base_man; /* base for manpage href */
        char             *base_includes; /* base for include href */
        char             *style; /* style-sheet URI */
        char              buf[BUFSIZ]; /* see bufcat and friends */
```

```
128         size_t            buflen;
129         struct tag       *metaf; /* current open font scope */
130         enum htmlfont     metal; /* last used font */
131         enum htmlfont     metac; /* current font mode */
132         enum htmltype     type; /* output media type */
133         int               oflags; /* output options */
134 #define HTML_FRAGMENT    (1 << 0) /* don't emit HTML/HEAD/BODY */
135 };

137 void              print_gen_decls(struct html *);
138 void              print_gen_head(struct html *);
139 struct tag       *print_otag(struct html *, enum htmltag,
140                              int, const struct htmlpair *);
141 void              print_tagq(struct html *, const struct tag *);
142 void              print_stagq(struct html *, const struct tag *);
143 void              print_text(struct html *, const char *);
144 void              print_tblclose(struct html *);
145 void              print_tbl(struct html *, const struct tbl_span *);
146 void              print_eqn(struct html *, const struct eqn *);

148 void              bufcat_fmt(struct html *, const char *, ...);
149 void              bufcat(struct html *, const char *);
150 void              bufcat_id(struct html *, const char *);
151 void              bufcat_style(struct html *,
152                           const char *, const char *);
153 void              bufcat_su(struct html *, const char *,
154                           const struct roffsu *);
155 void              bufinit(struct html *);
156 void              buffmt_man(struct html *,
157                           const char *, const char *);
158 void              buffmt_includes(struct html *, const char *);

160 int               html_strlen(const char *);

162 __END_DECLS

164 #endif /*!HTML_H*/
```

```
**********************************************************
    1155 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/lib.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: lib.c,v 1.9 2011/03/22 14:33:05 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 <stdlib.h>
#include <string.h>
#include <time.h>

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

#define LINE(x, y) \
        if (0 == strcmp(p, x)) return(y);

const char *
mdoc_a2lib(const char *p)
{

#include "lib.in"

        return(NULL);
}
```

```
**********************************************************
     497 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/lib.in
Initial import of man functionality.
**********************************************************
```

```
   1 /*
   2  * This file and its contents are supplied under the terms of the
   3  * Common Development and Distribution License ("CDDL"), version 1.0.
   4  * You may only use this file in accordance with the terms of version
   5  * 1.0 of the CDDL.
   6  *
   7  * A full copy of the text of the CDDL should have accompanied this
   8  * source.  A copy of the CDDL is also available via the Internet at
   9  * http://www.illumos.org/license/CDDL.
  10  */

  12 /*
  13  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  14  */

  16 /*
  17  * TBD
  18  */
```

```
**********************************************************
    3102 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/libman.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: libman.h,v 1.55 2011/11/07 01:24:40 schwarze Exp $ */
/*
 * 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 LIBMAN_H
#define LIBMAN_H

enum    man_next {
        MAN_NEXT_SIBLING = 0,
        MAN_NEXT_CHILD
};

struct  man {
        struct mparse   *parse; /* parse pointer */
        int              flags; /* parse flags */
#define MAN_HALT        (1 << 0) /* badness happened: die */
#define MAN_ELINE       (1 << 1) /* Next-line element scope. */
#define MAN_BLINE       (1 << 2) /* Next-line block scope. */
#define MAN_ILINE       (1 << 3) /* Ignored in next-line scope. */
#define MAN_LITERAL     (1 << 4) /* Literal input. */
#define MAN_BPLINE      (1 << 5)
#define MAN_NEWLINE     (1 << 6) /* first macro/text in a line */
        enum man_next    next; /* where to put the next node */
        struct man_node *last; /* the last parsed node */
        struct man_node *first; /* the first parsed node */
        struct man_meta  meta; /* document meta-data */
        struct roff     *roff;
};

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

struct  man_macro {
        int             (*fp)(MACRO_PROT_ARGS);
        int               flags;
#define MAN_SCOPED       (1 << 0)
#define MAN_EXPLICIT     (1 << 1)       /* See blk_imp(). */
#define MAN_FSCOPED      (1 << 2)       /* See blk_imp(). */
#define MAN_NSCOPED      (1 << 3)       /* See in_line_eoln(). */
#define MAN_NOCLOSE      (1 << 4)       /* See blk_exp(). */
#define MAN_BSCOPE       (1 << 5)       /* Break BLINE scope. */
};

extern  const struct man_macro *const man_macros;
```



```
__BEGIN_DECLS

#define           man_pmsg(m, l, p, t) \
                  mandoc_msg((t), (m)->parse, (l), (p), NULL)
#define           man_nmsg(m, n, t) \
                  mandoc_msg((t), (m)->parse, (n)->line, (n)->pos, NULL)
int               man_word_alloc(struct man *, int, int, const char *);
int               man_block_alloc(struct man *, int, int, enum mant);
int               man_head_alloc(struct man *, int, int, enum mant);
int               man_tail_alloc(struct man *, int, int, enum mant);
int               man_body_alloc(struct man *, int, int, enum mant);
int               man_elem_alloc(struct man *, int, int, enum mant);
void              man_node_delete(struct man *, struct man_node *);
void              man_hash_init(void);
enum mant         man_hash_find(const char *);
int               man_macroend(struct man *);
int               man_valid_post(struct man *);
int               man_valid_pre(struct man *, struct man_node *);
int               man_unscope(struct man *,
                        const struct man_node *, enum mandocerr);

__END_DECLS

#endif /*!LIBMAN_H*/
```

```
**********************************************************
    3343 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/libmandoc.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: libmandoc.h,v 1.29 2011/12/02 01:37:14 schwarze Exp $ */
/*
 * 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 *);
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(struct mparse *);
void             roff_reset(struct roff *);
enum rofferr     roff_parseln(struct roff *, int,
                       char **, size_t *, int, int *);
void             roff_endparse(struct roff *);
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 *);
#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*/
```

```
**********************************************************
    4877 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/libmdoc.h
Initial import of man functionality.
**********************************************************
```

```
/*      $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>
 *
 * 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 */
        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 */
        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 *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)
        /* Reserved words in arguments treated as text. */
};
```



```
enum    margserr {
        ARGS_ERROR,
        ARGS_EOLN, /* end-of-line */
        ARGS_WORD, /* normal word */
        ARGS_PUNCT, /* series of punctuation */
        ARGS_QWORD, /* quoted word */
        ARGS_PHRASE, /* Ta’d phrase (-column) */
        ARGS_PPHRASE, /* tabbed phrase (-column) */
        ARGS_PEND /* last phrase (-column) */
};

enum    margverr {
        ARGV_ERROR,
        ARGV_EOLN, /* end of line */
        ARGV_ARG, /* valid argument */
        ARGV_WORD /* normal word (or bad argument---same thing) */
};

/*
 * A punctuation delimiter is opening, closing, or "middle mark"
 * punctuation.  These govern spacing.
 * Opening punctuation (e.g., the opening parenthesis) suppresses the
 * following space; closing punctuation (e.g., the closing parenthesis)
 * suppresses the leading space; middle punctuation (e.g., the vertical
 * bar) can do either.  The middle punctuation delimiter bends the rules
 * depending on usage.
 */
enum    mdelim {
        DELIM_NONE = 0,
        DELIM_OPEN,
        DELIM_MIDDLE,
        DELIM_CLOSE,
        DELIM_MAX
};

extern  const struct mdoc_macro *const mdoc_macros;

__BEGIN_DECLS

#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 *);
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 *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 *);
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 *);
```

```
128 int                 mdoc_valid_post(struct mdoc *);
129 enum margverr       mdoc_argv(struct mdoc *, int, enum mdoct,
130                             struct mdoc_arg **, int *, char *);
131 void                mdoc_argv_free(struct mdoc_arg *);
132 enum margserr       mdoc_args(struct mdoc *, int,
133                             int *, char *, enum mdoct, char **);
134 enum margserr       mdoc_zargs(struct mdoc *, int,
135                             int *, char *, char **);
136 int                 mdoc_macroend(struct mdoc *);
137 enum mdelim         mdoc_isdelim(const char *);

139 __END_DECLS

141 #endif /*!LIBMDOC_H*/
```

```
**********************************************************
    2642 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/libroff.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: libroff.h,v 1.27 2011/07/25 15:37:00 kristaps Exp $ */
/*
 * 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 LIBROFF_H
#define LIBROFF_H

__BEGIN_DECLS

enum    tbl_part {
        TBL_PART_OPTS, /* in options (first line) */
        TBL_PART_LAYOUT, /* describing layout */
        TBL_PART_DATA, /* creating data rows */
        TBL_PART_CDATA /* continue previous row */
};

struct  tbl_node {
        struct mparse    *parse; /* parse point */
        int               pos; /* invocation column */
        int               line; /* invocation line */
        enum tbl_part     part;
        struct tbl        opts;
        struct tbl_row   *first_row;
        struct tbl_row   *last_row;
        struct tbl_span  *first_span;
        struct tbl_span  *current_span;
        struct tbl_span  *last_span;
        struct tbl_head  *first_head;
        struct tbl_head  *last_head;
        struct tbl_node  *next;
};

struct  eqn_node {
        struct eqn_def   *defs;
        size_t            defsz;
        char             *data;
        size_t            rew;
        size_t            cur;
        size_t            sz;
        int               gsize;
        struct eqn        eqn;
        struct mparse    *parse;
        struct eqn_node  *next;
};

struct  eqn_def {
        char             *key;
        size_t            keysz;
        char             *val;
```



```
        size_t            valsz;
};

struct tbl_node *tbl_alloc(int, int, struct mparse *);
void             tbl_restart(int, int, struct tbl_node *);
void             tbl_free(struct tbl_node *);
void             tbl_reset(struct tbl_node *);
enum rofferr     tbl_read(struct tbl_node *, int, const char *, int);
int              tbl_option(struct tbl_node *, int, const char *);
int              tbl_layout(struct tbl_node *, int, const char *);
int              tbl_data(struct tbl_node *, int, const char *);
int              tbl_cdata(struct tbl_node *, int, const char *);
const struct tbl_span   *tbl_span(struct tbl_node *);
void             tbl_end(struct tbl_node **);
struct eqn_node *eqn_alloc(const char *, int, int, struct mparse *);
enum rofferr     eqn_end(struct eqn_node **);
void             eqn_free(struct eqn_node *);
enum rofferr     eqn_read(struct eqn_node **, int,
                         const char *, int, int *);

__END_DECLS

#endif /*LIBROFF_H*/
```

```
**********************************************************
    8872 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/main.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: main.c,v 1.165 2011/10/06 22:29:12 kristaps Exp $ */
/*
 * 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 <assert.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

#if !defined(__GNUC__) || (__GNUC__ < 2)
# if !defined(lint)
#  define __attribute__(x)
# endif
#endif /* !defined(__GNUC__) || (__GNUC__ < 2) */

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

enum    outt {
        OUTT_ASCII = 0, /* -Tascii */
        OUTT_LOCALE,    /* -Tlocale */
        OUTT_UTF8,      /* -Tutf8 */
        OUTT_TREE,      /* -Ttree */
        OUTT_MAN,       /* -Tman */
        OUTT_HTML,      /* -Thtml */
        OUTT_XHTML,     /* -Txhtml */
        OUTT_LINT,      /* -Tlint */
        OUTT_PS,        /* -Tps */
        OUTT_PDF        /* -Tpdf */
};

struct  curparse {
        struct mparse    *mp;
        enum mandoclevel  wlevel;       /* ignore messages below this */
        int               wstop;        /* stop after a file with a warning */
        enum outt         outtype;      /* which output to use */
```



```
        out_mdoc          outmdoc;      /* mdoc output ptr */
        out_man           outman;       /* man output ptr */
        out_free          outfree;      /* free output ptr */
        void             *outdata;      /* data for output */
        char              outopts[BUFSIZ]; /* buf of output opts */
};

static  int               moptions(enum mparset *, char *);
static  void              mmsg(enum mandocerr, enum mandoclevel,
                               const char *, int, int, const char *);
static  void              parse(struct curparse *, int,
                               const char *, enum mandoclevel *);
static  int               toptions(struct curparse *, char *);
static  void              usage(void) __attribute__((noreturn));
static  void              version(void) __attribute__((noreturn));
static  int               woptions(struct curparse *, char *);

static  const char       *progname;

int
main(int argc, char *argv[])
{
        int              c;
        struct curparse  curp;
        enum mparset     type;
        enum mandoclevel rc;

        progname = strrchr(argv[0], '/');
        if (progname == NULL)
                progname = argv[0];
        else
                ++progname;

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

        type = MPARSE_AUTO;
        curp.outtype = OUTT_ASCII;
        curp.wlevel  = MANDOCLEVEL_FATAL;

        /* LINTED */
        while (-1 != (c = getopt(argc, argv, "m:O:T:VW:")))
                switch (c) {
                case ('m'):
                        if ( ! moptions(&type, optarg))
                                return((int)MANDOCLEVEL_BADARG);
                        break;
                case ('O'):
                        (void)strlcat(curp.outopts, optarg, BUFSIZ);
                        (void)strlcat(curp.outopts, ",", BUFSIZ);
                        break;
                case ('T'):
                        if ( ! toptions(&curp, optarg))
                                return((int)MANDOCLEVEL_BADARG);
                        break;
                case ('W'):
                        if ( ! woptions(&curp, optarg))
                                return((int)MANDOCLEVEL_BADARG);
                        break;
                case ('V'):
                        version();
                        /* NOTREACHED */
                default:
                        usage();
                        /* NOTREACHED */
                }
```

```
128         curp.mp = mparse_alloc(type, curp.wlevel, mmsg, &curp);
129
130         /*
131          * Conditionally start up the lookaside buffer before parsing.
132          */
133         if (OUTT_MAN == curp.outtype)
134                 mparse_keep(curp.mp);
135
136         argc -= optind;
137         argv += optind;
138
139         rc = MANDOCLEVEL_OK;
140
141         if (NULL == *argv)
142                 parse(&curp, STDIN_FILENO, "<stdin>", &rc);
143
144         while (*argv) {
145                 parse(&curp, -1, *argv, &rc);
146                 if (MANDOCLEVEL_OK != rc && curp.wstop)
147                         break;
148                 ++argv;
149         }
150
151         if (curp.outfree)
152                 (*curp.outfree)(curp.outdata);
153         if (curp.mp)
154                 mparse_free(curp.mp);
155
156         return((int)rc);
157 }
158
159 static void
160 version(void)
161 {
162
163         printf("%s %s\n", progname, VERSION);
164         exit((int)MANDOCLEVEL_OK);
165 }
166
167 static void
168 usage(void)
169 {
170
171         fprintf(stderr, "usage: %s "
172                         "[-V] "
173                         "[-foption] "
174                         "[-mformat] "
175                         "[-Ooption] "
176                         "[-Toutput] "
177                         "[-Wlevel] "
178                         "[file...]\n",
179                         progname);
180
181         exit((int)MANDOCLEVEL_BADARG);
182 }
183
184 static void
185 parse(struct curparse *curp, int fd,
186                 const char *file, enum mandoclevel *level)
187 {
188         enum mandoclevel  rc;
189         struct mdoc      *mdoc;
190         struct man       *man;
191
192         /* Begin by parsing the file itself. */
```



```
193
194         assert(file);
195         assert(fd >= -1);
196
197         rc = mparse_readfd(curp->mp, fd, file);
198
199         /* Stop immediately if the parse has failed. */
200
201         if (MANDOCLEVEL_FATAL <= rc)
202                 goto cleanup;
203
204         /*
205          * With -Wstop and warnings or errors of at least the requested
206          * level, do not produce output.
207          */
208
209         if (MANDOCLEVEL_OK != rc && curp->wstop)
210                 goto cleanup;
211
212         /* If unset, allocate output dev now (if applicable). */
213
214         if ( ! (curp->outman && curp->outmdoc)) {
215                 switch (curp->outtype) {
216                 case (OUTT_XHTML):
217                         curp->outdata = xhtml_alloc(curp->outopts);
218                         curp->outfree = html_free;
219                         break;
220                 case (OUTT_HTML):
221                         curp->outdata = html_alloc(curp->outopts);
222                         curp->outfree = html_free;
223                         break;
224                 case (OUTT_UTF8):
225                         curp->outdata = utf8_alloc(curp->outopts);
226                         curp->outfree = ascii_free;
227                         break;
228                 case (OUTT_LOCALE):
229                         curp->outdata = locale_alloc(curp->outopts);
230                         curp->outfree = ascii_free;
231                         break;
232                 case (OUTT_ASCII):
233                         curp->outdata = ascii_alloc(curp->outopts);
234                         curp->outfree = ascii_free;
235                         break;
236                 case (OUTT_PDF):
237                         curp->outdata = pdf_alloc(curp->outopts);
238                         curp->outfree = pspdf_free;
239                         break;
240                 case (OUTT_PS):
241                         curp->outdata = ps_alloc(curp->outopts);
242                         curp->outfree = pspdf_free;
243                         break;
244                 default:
245                         break;
246                 }
247
248                 switch (curp->outtype) {
249                 case (OUTT_HTML):
250                         /* FALLTHROUGH */
251                 case (OUTT_XHTML):
252                         curp->outman = html_man;
253                         curp->outmdoc = html_mdoc;
254                         break;
255                 case (OUTT_TREE):
256                         curp->outman = tree_man;
257                         curp->outmdoc = tree_mdoc;
258                         break;
259                 case (OUTT_MAN):
```

```
260                         curp->outmdoc = man_mdoc;
261                         curp->outman = man_man;
262                         break;
263                 case (OUTT_PDF):
264                         /* FALLTHROUGH */
265                 case (OUTT_ASCII):
266                         /* FALLTHROUGH */
267                 case (OUTT_UTF8):
268                         /* FALLTHROUGH */
269                 case (OUTT_LOCALE):
270                         /* FALLTHROUGH */
271                 case (OUTT_PS):
272                         curp->outman = terminal_man;
273                         curp->outmdoc = terminal_mdoc;
274                         break;
275                 default:
276                         break;
277                 }
278         }

280         mparse_result(curp->mp, &mdoc, &man);

282         /* Execute the out device, if it exists. */

284         if (man && curp->outman)
285                 (*curp->outman)(curp->outdata, man);
286         if (mdoc && curp->outmdoc)
287                 (*curp->outmdoc)(curp->outdata, mdoc);

289  cleanup:

291         mparse_reset(curp->mp);

293         if (*level < rc)
294                 *level = rc;
295 }

297 static int
298 moptions(enum mparset *tflags, char *arg)
299 {

301         if (0 == strcmp(arg, "doc"))
302                 *tflags = MPARSE_MDOC;
303         else if (0 == strcmp(arg, "andoc"))
304                 *tflags = MPARSE_AUTO;
305         else if (0 == strcmp(arg, "an"))
306                 *tflags = MPARSE_MAN;
307         else {
308                 fprintf(stderr, "%s: Bad argument\n", arg);
309                 return(0);
310         }

312         return(1);
313 }

315 static int
316 toptions(struct curparse *curp, char *arg)
317 {

319         if (0 == strcmp(arg, "ascii"))
320                 curp->outtype = OUTT_ASCII;
321         else if (0 == strcmp(arg, "lint")) {
322                 curp->outtype = OUTT_LINT;
323                 curp->wlevel  = MANDOCLEVEL_WARNING;
324         } else if (0 == strcmp(arg, "tree"))
325                 curp->outtype = OUTT_TREE;
```



```
326         else if (0 == strcmp(arg, "man"))
327                 curp->outtype = OUTT_MAN;
328         else if (0 == strcmp(arg, "html"))
329                 curp->outtype = OUTT_HTML;
330         else if (0 == strcmp(arg, "utf8"))
331                 curp->outtype = OUTT_UTF8;
332         else if (0 == strcmp(arg, "locale"))
333                 curp->outtype = OUTT_LOCALE;
334         else if (0 == strcmp(arg, "xhtml"))
335                 curp->outtype = OUTT_XHTML;
336         else if (0 == strcmp(arg, "ps"))
337                 curp->outtype = OUTT_PS;
338         else if (0 == strcmp(arg, "pdf"))
339                 curp->outtype = OUTT_PDF;
340         else {
341                 fprintf(stderr, "%s: Bad argument\n", arg);
342                 return(0);
343         }

345         return(1);
346 }

348 static int
349 woptions(struct curparse *curp, char *arg)
350 {
351         char            *v, *o;
352         const char      *toks[6];

354         toks[0] = "stop";
355         toks[1] = "all";
356         toks[2] = "warning";
357         toks[3] = "error";
358         toks[4] = "fatal";
359         toks[5] = NULL;

361         while (*arg) {
362                 o = arg;
363                 switch (getsubopt(&arg, UNCONST(toks), &v)) {
364                 case (0):
365                         curp->wstop = 1;
366                         break;
367                 case (1):
368                         /* FALLTHROUGH */
369                 case (2):
370                         curp->wlevel = MANDOCLEVEL_WARNING;
371                         break;
372                 case (3):
373                         curp->wlevel = MANDOCLEVEL_ERROR;
374                         break;
375                 case (4):
376                         curp->wlevel = MANDOCLEVEL_FATAL;
377                         break;
378                 default:
379                         fprintf(stderr, "-W%s: Bad argument\n", o);
380                         return(0);
381                 }
382         }

384         return(1);
385 }

387 static void
388 mmsg(enum mandocerr t, enum mandoclevel lvl,
389                 const char *file, int line, int col, const char *msg)
390 {
```

```
392         fprintf(stderr, "%s:%d:%d: %s: %s",
393                         file, line, col + 1,
394                         mparse_strlevel(lvl),
395                         mparse_strerror(t));
396
397         if (msg)
398                 fprintf(stderr, ": %s", msg);
399
400         fputc('\n', stderr);
401 }
```

```
************************************************************
    1974 Tue Jul 15 13:48:05 2014
new/usr/src/cmd/mandoc/main.h
Initial import of man functionality.
************************************************************
```

```
/*      $Id: main.h,v 1.15 2011/10/06 22:29:12 kristaps Exp $ */
/*
 * 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 MAIN_H
#define MAIN_H

__BEGIN_DECLS

struct  mdoc;
struct  man;

#define UNCONST(a)      ((void *)(uintptr_t)(const void *)(a))


/*
 * Definitions for main.c-visible output device functions, e.g., -Thtml
 * and -Tascii.  Note that ascii_alloc() is named as such in
 * anticipation of latin1_alloc() and so on, all of which map into the
 * terminal output routines with different character settings.
 */

void             *html_alloc(char *);
void             *xhtml_alloc(char *);
void              html_mdoc(void *, const struct mdoc *);
void              html_man(void *, const struct man *);
void              html_free(void *);

void              tree_mdoc(void *, const struct mdoc *);
void              tree_man(void *, const struct man *);

void              man_mdoc(void *, const struct mdoc *);
void              man_man(void *, const struct man *);

void             *locale_alloc(char *);
void             *utf8_alloc(char *);
void             *ascii_alloc(char *);
void              ascii_free(void *);

void             *pdf_alloc(char *);
void             *ps_alloc(char *);
void              pspdf_free(void *);

void              terminal_mdoc(void *, const struct mdoc *);
void              terminal_man(void *, const struct man *);

__END_DECLS

#endif /*!MAIN_H*/
```

```
**********************************************************
   13782 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man.c
Initial import of man functionality.
**********************************************************
```

```
  1 /*      $Id: man.c,v 1.115 2012/01/03 15:16:24 kristaps Exp $ */
  2 /*
  3  * Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
  4  *
  5  * Permission to use, copy, modify, and distribute this software for any
  6  * purpose with or without fee is hereby granted, provided that the above
  7  * copyright notice and this permission notice appear in all copies.
  8  *
  9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 10  * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 11  * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 12  * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 13  * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 14  * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 15  * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 16  */
 17 #ifdef HAVE_CONFIG_H
 18 #include "config.h"
 19 #endif

 21 #include <sys/types.h>

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

 29 #include "man.h"
 30 #include "mandoc.h"
 31 #include "libman.h"
 32 #include "libmandoc.h"

 34 const   char *const __man_macronames[MAN_MAX] = {
 35         "br",           "TH",           "SH",           "SS",
 36         "TP",           "LP",           "PP",           "P",
 37         "IP",           "HP",           "SM",           "SB",
 38         "BI",           "IB",           "BR",           "RB",
 39         "R",            "B",            "I",            "IR",
 40         "RI",           "na",           "sp",           "nf",
 41         "fi",           "RE",           "RS",           "DT",
 42         "UC",           "PD",           "AT",           "in",
 43         "ft",           "OP"
 44         };

 46 const   char * const *man_macronames = __man_macronames;

 48 static  struct man_node *man_node_alloc(struct man *, int, int,
 49                                 enum man_type, enum mant);
 50 static  int              man_node_append(struct man *,
 51                                 struct man_node *);
 52 static  void             man_node_free(struct man_node *);
 53 static  void             man_node_unlink(struct man *,
 54                                 struct man_node *);
 55 static  int              man_ptext(struct man *, int, char *, int);
 56 static  int              man_pmacro(struct man *, int, char *, int);
 57 static  void             man_free1(struct man *);
 58 static  void             man_alloc1(struct man *);
 59 static  int              man_descope(struct man *, int, int);
```



```
 62 const struct man_node *
 63 man_node(const struct man *m)
 64 {

 66         assert( ! (MAN_HALT & m->flags));
 67         return(m->first);
 68 }


 71 const struct man_meta *
 72 man_meta(const struct man *m)
 73 {

 75         assert( ! (MAN_HALT & m->flags));
 76         return(&m->meta);
 77 }


 80 void
 81 man_reset(struct man *man)
 82 {

 84         man_free1(man);
 85         man_alloc1(man);
 86 }


 89 void
 90 man_free(struct man *man)
 91 {

 93         man_free1(man);
 94         free(man);
 95 }


 98 struct man *
 99 man_alloc(struct roff *roff, struct mparse *parse)
100 {
101         struct man      *p;

103         p = mandoc_calloc(1, sizeof(struct man));

105         man_hash_init();
106         p->parse = parse;
107         p->roff = roff;

109         man_alloc1(p);
110         return(p);
111 }


114 int
115 man_endparse(struct man *m)
116 {

118         assert( ! (MAN_HALT & m->flags));
119         if (man_macroend(m))
120                 return(1);
121         m->flags |= MAN_HALT;
122         return(0);
123 }


126 int
127 man_parseln(struct man *m, int ln, char *buf, int offs)
```

```
128 {
130         m->flags |= MAN_NEWLINE;
132         assert( ! (MAN_HALT & m->flags));
134         return (mandoc_getcontrol(buf, &offs) ?
135                         man_pmacro(m, ln, buf, offs) :
136                         man_ptext(m, ln, buf, offs));
137 }
140 static void
141 man_free1(struct man *man)
142 {
144         if (man->first)
145                 man_node_delete(man, man->first);
146         if (man->meta.title)
147                 free(man->meta.title);
148         if (man->meta.source)
149                 free(man->meta.source);
150         if (man->meta.date)
151                 free(man->meta.date);
152         if (man->meta.vol)
153                 free(man->meta.vol);
154         if (man->meta.msec)
155                 free(man->meta.msec);
156 }
159 static void
160 man_alloc1(struct man *m)
161 {
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;
170 }
173 static int
174 man_node_append(struct man *man, struct man_node *p)
175 {
177         assert(man->last);
178         assert(man->first);
179         assert(MAN_ROOT != p->type);
181         switch (man->next) {
182         case (MAN_NEXT_SIBLING):
183                 man->last->next = p;
184                 p->prev = man->last;
185                 p->parent = man->last->parent;
186                 break;
187         case (MAN_NEXT_CHILD):
188                 man->last->child = p;
189                 p->parent = man->last;
190                 break;
191         default:
192                 abort();
193                 /* NOTREACHED */
```



```
194         }
195
196         assert(p->parent);
197         p->parent->nchild++;
199         if ( ! man_valid_pre(man, p))
200                 return(0);
202         switch (p->type) {
203         case (MAN_HEAD):
204                 assert(MAN_BLOCK == p->parent->type);
205                 p->parent->head = p;
206                 break;
207         case (MAN_TAIL):
208                 assert(MAN_BLOCK == p->parent->type);
209                 p->parent->tail = p;
210                 break;
211         case (MAN_BODY):
212                 assert(MAN_BLOCK == p->parent->type);
213                 p->parent->body = p;
214                 break;
215         default:
216                 break;
217         }
219         man->last = p;
221         switch (p->type) {
222         case (MAN_TBL):
223                 /* FALLTHROUGH */
224         case (MAN_TEXT):
225                 if ( ! man_valid_post(man))
226                         return(0);
227                 break;
228         default:
229                 break;
230         }
232         return(1);
233 }
236 static struct man_node *
237 man_node_alloc(struct man *m, int line, int pos,
238                 enum man_type type, enum mant tok)
239 {
240         struct man_node *p;
242         p = mandoc_calloc(1, sizeof(struct man_node));
243         p->line = line;
244         p->pos = pos;
245         p->type = type;
246         p->tok = tok;
248         if (MAN_NEWLINE & m->flags)
249                 p->flags |= MAN_LINE;
250         m->flags &= ~MAN_NEWLINE;
251         return(p);
252 }
255 int
256 man_elem_alloc(struct man *m, int line, int pos, enum mant tok)
257 {
258         struct man_node *p;
```

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


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

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


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

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


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

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


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

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

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

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



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

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


335 /*
336  * Free all of the resources held by a node.  This does NOT unlink a
337  * node from its context; for that, see man_node_unlink().
338  */
339 static void
340 man_node_free(struct man_node *p)
341 {

343         if (p->string)
344                 free(p->string);
345         free(p);
346 }


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

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

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

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

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

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

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

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

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

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

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

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

390         m->next = MAN_NEXT_SIBLING;
391         return(man_descope(m, sp->line, 0));
```

```
}

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

        if (MAN_ELINE & m->flags) {
                m->flags &= ~MAN_ELINE;
                if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
                        return(0);
        }

        if ( ! (MAN_BLINE & m->flags))
                return(1);
        m->flags &= ~MAN_BLINE;

        if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
                return(0);
        return(man_body_alloc(m, line, offs, m->last->tok));
}

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

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

        if (MAN_LITERAL & m->flags) {
                if ( ! man_word_alloc(m, line, offs, buf + offs))
                        return(0);
                return(man_descope(m, line, offs));
        }

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

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

        if ('\0' == buf[i]) {
                /* Allocate a blank entry. */
                if ( ! man_word_alloc(m, line, offs, ""))
                        return(0);
                return(man_descope(m, line, offs));
        }

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

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

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

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



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

                buf[i] = '\0';
        }

        if ( ! man_word_alloc(m, line, offs, buf + offs))
                return(0);

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

        assert(i);
        if (mandoc_eos(buf, (size_t)i, 0))
                m->last->flags |= MAN_EOS;

        return(man_descope(m, line, offs));
}

static int
man_pmacro(struct man *m, int ln, char *buf, int offs)
{
        int              i, ppos;
        enum mant        tok;
        char             mac[5];
        struct man_node *n;

        if ('"' == buf[offs]) {
                man_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
                return(1);
        } else if ('\0' == buf[offs])
                return(1);

        ppos = offs;

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

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

        mac[i] = '\0';

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

        if (MAN_MAX == tok) {
                mandoc_vmsg(MANDOCERR_MACRO, m->parse, ln,
                                ppos, "%s", buf + ppos - 1);
                return(1);
        }

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

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

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

```
524          */
526         if ('\0' == buf[offs] && ' ' == buf[offs - 1])
527                 man_pmsg(m, ln, offs - 1, MANDOCERR_EOLNSPACE);
529         /*
530          * Remove prior ELINE macro, as it's being clobbered by a new
531          * macro.  Note that NSCOPED macros do not close out ELINE
532          * macros---they don't print text---so we let those slip by.
533          */
535         if ( ! (MAN_NSCOPED & man_macros[tok].flags) &&
536                         m->flags & MAN_ELINE) {
537                 n = m->last;
538                 assert(MAN_TEXT != n->type);
540                 /* Remove repeated NSCOPED macros causing ELINE. */
542                 if (MAN_NSCOPED & man_macros[n->tok].flags)
543                         n = n->parent;
545                 mandoc_vmsg(MANDOCERR_LINESCOPE, m->parse, n->line,
546                     n->pos, "%s breaks %s", man_macronames[tok],
547                     man_macronames[n->tok]);
549                 man_node_delete(m, n);
550                 m->flags &= ~MAN_ELINE;
551         }
553         /*
554          * Remove prior BLINE macro that is being clobbered.
555          */
556         if ((m->flags & MAN_BLINE) &&
557             (MAN_BSCOPE & man_macros[tok].flags)) {
558                 n = m->last;
560                 /* Might be a text node like 8 in
561                  * .TP 8
562                  * .SH foo
563                  */
564                 if (MAN_TEXT == n->type)
565                         n = n->parent;
567                 /* Remove element that didn't end BLINE, if any. */
568                 if ( ! (MAN_BSCOPE & man_macros[n->tok].flags))
569                         n = n->parent;
571                 assert(MAN_HEAD == n->type);
572                 n = n->parent;
573                 assert(MAN_BLOCK == n->type);
574                 assert(MAN_SCOPED & man_macros[n->tok].flags);
576                 mandoc_vmsg(MANDOCERR_LINESCOPE, m->parse, n->line,
577                     n->pos, "%s breaks %s", man_macronames[tok],
578                     man_macronames[n->tok]);
580                 man_node_delete(m, n);
581                 m->flags &= ~MAN_BLINE;
582         }
584         /*
585          * Save the fact that we're in the next-line for a block.  In
586          * this way, embedded roff instructions can "remember" state
587          * when they exit.
588          */
```



```
590         if (MAN_BLINE & m->flags)
591                 m->flags |= MAN_BPLINE;
593         /* Call to handler... */
595         assert(man_macros[tok].fp);
596         if ( ! (*man_macros[tok].fp)(m, tok, ln, ppos, &offs, buf))
597                 goto err;
599         /*
600          * We weren't in a block-line scope when entering the
601          * above-parsed macro, so return.
602          */
604         if ( ! (MAN_BPLINE & m->flags)) {
605                 m->flags &= ~MAN_ILINE;
606                 return(1);
607         }
608         m->flags &= ~MAN_BPLINE;
610         /*
611          * If we're in a block scope, then allow this macro to slip by
612          * without closing scope around it.
613          */
615         if (MAN_ILINE & m->flags) {
616                 m->flags &= ~MAN_ILINE;
617                 return(1);
618         }
620         /*
621          * If we've opened a new next-line element scope, then return
622          * now, as the next line will close out the block scope.
623          */
625         if (MAN_ELINE & m->flags)
626                 return(1);
628         /* Close out the block scope opened in the prior line.  */
630         assert(MAN_BLINE & m->flags);
631         m->flags &= ~MAN_BLINE;
633         if ( ! man_unscope(m, m->last->parent, MANDOCERR_MAX))
634                 return(0);
635         return(man_body_alloc(m, ln, ppos, m->last->tok));
637 err:    /* Error out. */
639         m->flags |= MAN_HALT;
640         return(0);
641 }
643 /*
644  * Unlink a node from its context.  If "m" is provided, the last parse
645  * point will also be adjusted accordingly.
646  */
647 static void
648 man_node_unlink(struct man *m, struct man_node *n)
649 {
651         /* Adjust siblings. */
653         if (n->prev)
654                 n->prev->next = n->next;
655         if (n->next)
```

```
656                 n->next->prev = n->prev;

658         /* Adjust parent. */

660         if (n->parent) {
661                 n->parent->nchild--;
662                 if (n->parent->child == n)
663                         n->parent->child = n->prev ? n->prev : n->next;
664         }

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

668         if (m && m->last == n) {
669                 /*XXX: this can occur when bailing from validation. */
670                 /*assert(NULL == n->next);*/
671                 if (n->prev) {
672                         m->last = n->prev;
673                         m->next = MAN_NEXT_SIBLING;
674                 } else {
675                         m->last = n->parent;
676                         m->next = MAN_NEXT_CHILD;
677                 }
678         }

680         if (m && m->first == n)
681                 m->first = NULL;
682 }

684 const struct mparse *
685 man_mparse(const struct man *m)
686 {

688         assert(m && m->parse);
689         return(m->parse);
690 }
```

```
**********************************************************
    2695 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: man.h,v 1.60 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * 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 MAN_H
#define MAN_H

enum    mant {
        MAN_br = 0,
        MAN_TH,
        MAN_SH,
        MAN_SS,
        MAN_TP,
        MAN_LP,
        MAN_PP,
        MAN_P,
        MAN_IP,
        MAN_HP,
        MAN_SM,
        MAN_SB,
        MAN_BI,
        MAN_IB,
        MAN_BR,
        MAN_RB,
        MAN_R,
        MAN_B,
        MAN_I,
        MAN_IR,
        MAN_RI,
        MAN_na,
        MAN_sp,
        MAN_nf,
        MAN_fi,
        MAN_RE,
        MAN_RS,
        MAN_DT,
        MAN_UC,
        MAN_PD,
        MAN_AT,
        MAN_in,
        MAN_ft,
        MAN_OP,
        MAN_MAX
};

enum    man_type {
        MAN_TEXT,
        MAN_ELEM,
        MAN_ROOT,
```



```
        MAN_BLOCK,
        MAN_HEAD,
        MAN_BODY,
        MAN_TAIL,
        MAN_TBL,
        MAN_EQN
};

struct  man_meta {
        char            *msec; /* 'TH' section (1, 3p, etc.) */
        char            *date; /* 'TH' normalised date */
        char            *vol; /* 'TH' volume */
        char            *title; /* 'TH' title (e.g., FOO) */
        char            *source; /* 'TH' source (e.g., GNU) */
};

struct  man_node {
        struct man_node *parent; /* parent AST node */
        struct man_node *child; /* first child AST node */
        struct man_node *next; /* sibling AST node */
        struct man_node *prev; /* prior sibling AST node */
        int              nchild; /* number children */
        int              line;
        int              pos;
        enum mant        tok; /* tok or MAN__MAX if none */
        int              flags;
#define MAN_VALID       (1 << 0) /* has been validated */
#define MAN_EOS         (1 << 2) /* at sentence boundary */
#define MAN_LINE        (1 << 3) /* first macro/text on line */
        enum man_type    type; /* AST node type */
        char            *string; /* TEXT node argument */
        struct man_node *head; /* BLOCK node HEAD ptr */
        struct man_node *tail; /* BLOCK node TAIL ptr */
        struct man_node *body; /* BLOCK node BODY ptr */
        const struct tbl_span *span; /* TBL */
        const struct eqn *eqn; /* EQN */
};

/* Names of macros.  Index is enum mant. */
extern  const char *const *man_macronames;

__BEGIN_DECLS

struct  man;

const struct man_node *man_node(const struct man *);
const struct man_meta *man_meta(const struct man *);
const struct mparse   *man_mparse(const struct man *);

__END_DECLS

#endif /*!MAN_H*/
```

```
**********************************************************
    2586 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man_hash.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: man_hash.c,v 1.25 2011/07/24 18:15:14 kristaps Exp $ */
/*
 * Copyright (c) 2008, 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.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <sys/types.h>

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

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

#define HASH_DEPTH       6

#define HASH_ROW(x) do { \
                if (isupper((unsigned char)(x))) \
                        (x) -= 65; \
                else \
                        (x) -= 97; \
                (x) *= HASH_DEPTH; \
        } while (/* CONSTCOND */ 0)

/*
 * Lookup table is indexed first by lower-case first letter (plus one
 * for the period, which is stored in the last row), then by lower or
 * uppercase second letter.  Buckets correspond to the index of the
 * macro (the integer value of the enum stored as a char to save a bit
 * of space).
 */
static  unsigned char    table[26 * HASH_DEPTH];

/*
 * XXX - this hash has global scope, so if intended for use as a library
 * with multiple callers, it will need re-invocation protection.
 */
void
man_hash_init(void)
{
        int              i, j, x;

        memset(table, UCHAR_MAX, sizeof(table));
```



```
        assert(/* LINTED */
                        MAN_MAX < UCHAR_MAX);

        for (i = 0; i < (int)MAN_MAX; i++) {
                x = man_macronames[i][0];

                assert(isalpha((unsigned char)x));

                HASH_ROW(x);

                for (j = 0; j < HASH_DEPTH; j++)
                        if (UCHAR_MAX == table[x + j]) {
                                table[x + j] = (unsigned char)i;
                                break;
                        }

                assert(j < HASH_DEPTH);
        }
}


enum mant
man_hash_find(const char *tmp)
{
        int              x, y, i;
        enum mant        tok;

        if ('\0' == (x = tmp[0]))
                return(MAN_MAX);
        if ( ! (isalpha((unsigned char)x)))
                return(MAN_MAX);

        HASH_ROW(x);

        for (i = 0; i < HASH_DEPTH; i++) {
                if (UCHAR_MAX == (y = table[x + i]))
                        return(MAN_MAX);

                tok = (enum mant)y;
                if (0 == strcmp(tmp, man_macronames[tok]))
                        return(tok);
        }

        return(MAN_MAX);
}
```

```
**********************************************************
   13891 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man_html.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: man_html.c,v 1.86 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 <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

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

#define INDENT            5

#define MAN_ARGS          const struct man_meta *m, \
                          const struct man_node *n, \
                          struct mhtml *mh, \
                          struct html *h

struct  mhtml {
        int               fl;
#define MANH_LITERAL     (1 << 0) /* literal context */
};

struct  htmlman {
        int             (*pre)(MAN_ARGS);
        int             (*post)(MAN_ARGS);
};

static  void              print_bvspace(struct html *,
                                const struct man_node *);
static  void              print_man(MAN_ARGS);
static  void              print_man_head(MAN_ARGS);
static  void              print_man_nodelist(MAN_ARGS);
static  void              print_man_node(MAN_ARGS);
static  int               a2width(const struct man_node *,
```



```
                                struct roffsu *);
static  int               man_B_pre(MAN_ARGS);
static  int               man_HP_pre(MAN_ARGS);
static  int               man_IP_pre(MAN_ARGS);
static  int               man_I_pre(MAN_ARGS);
static  int               man_OP_pre(MAN_ARGS);
static  int               man_PP_pre(MAN_ARGS);
static  int               man_RS_pre(MAN_ARGS);
static  int               man_SH_pre(MAN_ARGS);
static  int               man_SM_pre(MAN_ARGS);
static  int               man_SS_pre(MAN_ARGS);
static  int               man_alt_pre(MAN_ARGS);
static  int               man_br_pre(MAN_ARGS);
static  int               man_ign_pre(MAN_ARGS);
static  int               man_in_pre(MAN_ARGS);
static  int               man_literal_pre(MAN_ARGS);
static  void              man_root_post(MAN_ARGS);
static  void              man_root_pre(MAN_ARGS);

static  const struct htmlman mans[MAN_MAX] = {
        { man_br_pre, NULL }, /* br */
        { NULL, NULL }, /* TH */
        { man_SH_pre, NULL }, /* SH */
        { man_SS_pre, NULL }, /* SS */
        { man_IP_pre, NULL }, /* TP */
        { man_PP_pre, NULL }, /* LP */
        { man_PP_pre, NULL }, /* PP */
        { man_PP_pre, NULL }, /* P */
        { man_IP_pre, NULL }, /* IP */
        { man_HP_pre, NULL }, /* HP */
        { man_SM_pre, NULL }, /* SM */
        { man_SM_pre, NULL }, /* SB */
        { man_alt_pre, NULL }, /* BI */
        { man_alt_pre, NULL }, /* IB */
        { man_alt_pre, NULL }, /* BR */
        { man_alt_pre, NULL }, /* RB */
        { NULL, NULL }, /* R */
        { man_B_pre, NULL }, /* B */
        { man_I_pre, NULL }, /* I */
        { man_alt_pre, NULL }, /* IR */
        { man_alt_pre, NULL }, /* RI */
        { man_ign_pre, NULL }, /* na */
        { man_br_pre, NULL }, /* sp */
        { man_literal_pre, NULL }, /* nf */
        { man_literal_pre, NULL }, /* fi */
        { NULL, NULL }, /* RE */
        { man_RS_pre, NULL }, /* RS */
        { man_ign_pre, NULL }, /* DT */
        { man_ign_pre, NULL }, /* UC */
        { man_ign_pre, NULL }, /* PD */
        { man_ign_pre, NULL }, /* AT */
        { man_in_pre, NULL }, /* in */
        { man_ign_pre, NULL }, /* ft */
        { man_OP_pre, NULL }, /* OP */
};

/*
 * Printing leading vertical space before a block.
 * This is used for the paragraph macros.
 * The rules are pretty simple, since there's very little nesting going
 * on here.  Basically, if we're the first within another block (SS/SH),
 * then don't emit vertical space.  If we are (RS), then do.  If not the
 * first, print it.
 */
static void
print_bvspace(struct html *h, const struct man_node *n)
```

```
128 {
130         if (n->body && n->body->child)
131                 if (MAN_TBL == n->body->child->type)
132                         return;
134         if (MAN_ROOT == n->parent->type || MAN_RS != n->parent->tok)
135                 if (NULL == n->prev)
136                         return;
138         print_otag(h, TAG_P, 0, NULL);
139 }
141 void
142 html_man(void *arg, const struct man *m)
143 {
144         struct mhtml     mh;
146         memset(&mh, 0, sizeof(struct mhtml));
147         print_man(man_meta(m), man_node(m), &mh, (struct html *)arg);
148         putchar('\n');
149 }
151 static void
152 print_man(MAN_ARGS)
153 {
154         struct tag      *t, *tt;
155         struct htmlpair  tag;
157         PAIR_CLASS_INIT(&tag, "mandoc");
159         if ( ! (HTML_FRAGMENT & h->oflags)) {
160                 print_gen_decls(h);
161                 t = print_otag(h, TAG_HTML, 0, NULL);
162                 tt = print_otag(h, TAG_HEAD, 0, NULL);
163                 print_man_head(m, n, mh, h);
164                 print_tagq(h, tt);
165                 print_otag(h, TAG_BODY, 0, NULL);
166                 print_otag(h, TAG_DIV, 1, &tag);
167         } else
168                 t = print_otag(h, TAG_DIV, 1, &tag);
170         print_man_nodelist(m, n, mh, h);
171         print_tagq(h, t);
172 }
175 /* ARGSUSED */
176 static void
177 print_man_head(MAN_ARGS)
178 {
180         print_gen_head(h);
181         assert(m->title);
182         assert(m->msec);
183         bufcat_fmt(h, "%s(%s)", m->title, m->msec);
184         print_otag(h, TAG_TITLE, 0, NULL);
185         print_text(h, h->buf);
186 }
189 static void
190 print_man_nodelist(MAN_ARGS)
191 {
193         print_man_node(m, n, mh, h);
```



```
194         if (n->next)
195                 print_man_nodelist(m, n->next, mh, h);
196 }
199 static void
200 print_man_node(MAN_ARGS)
201 {
202         int              child;
203         struct tag      *t;
205         child = 1;
206         t = h->tags.head;
208         switch (n->type) {
209         case (MAN_ROOT):
210                 man_root_pre(m, n, mh, h);
211                 break;
212         case (MAN_TEXT):
213                 /*
214                  * If we have a blank line, output a vertical space.
215                  * If we have a space as the first character, break
216                  * before printing the line's data.
217                  */
218                 if ('\0' == *n->string) {
219                         print_otag(h, TAG_P, 0, NULL);
220                         return;
221                 }
223                 if (' ' == *n->string && MAN_LINE & n->flags)
224                         print_otag(h, TAG_BR, 0, NULL);
225                 else if (MANH_LITERAL & mh->fl && n->prev)
226                         print_otag(h, TAG_BR, 0, NULL);
228                 print_text(h, n->string);
229                 return;
230         case (MAN_EQN):
231                 print_eqn(h, n->eqn);
232                 break;
233         case (MAN_TBL):
234                 /*
235                  * This will take care of initialising all of the table
236                  * state data for the first table, then tearing it down
237                  * for the last one.
238                  */
239                 print_tbl(h, n->span);
240                 return;
241         default:
242                 /*
243                  * Close out scope of font prior to opening a macro
244                  * scope.
245                  */
246                 if (HTMLFONT_NONE != h->metac) {
247                         h->metal = h->metac;
248                         h->metac = HTMLFONT_NONE;
249                 }
251                 /*
252                  * Close out the current table, if it's open, and unset
253                  * the "meta" table state.  This will be reopened on the
254                  * next table element.
255                  */
256                 if (h->tblt) {
257                         print_tblclose(h);
258                         t = h->tags.head;
259                 }
```

```
260                 if (mans[n->tok].pre)
261                         child = (*mans[n->tok].pre)(m, n, mh, h);
262                 break;
263         }

265         if (child && n->child)
266                 print_man_nodelist(m, n->child, mh, h);

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

271         switch (n->type) {
272         case (MAN_ROOT):
273                 man_root_post(m, n, mh, h);
274                 break;
275         case (MAN_EQN):
276                 break;
277         default:
278                 if (mans[n->tok].post)
279                         (*mans[n->tok].post)(m, n, mh, h);
280                 break;
281         }
282 }


285 static int
286 a2width(const struct man_node *n, struct roffsu *su)
287 {

289         if (MAN_TEXT != n->type)
290                 return(0);
291         if (a2roffsu(n->string, su, SCALE_BU))
292                 return(1);

294         return(0);
295 }


298 /* ARGSUSED */
299 static void
300 man_root_pre(MAN_ARGS)
301 {
302         struct htmlpair  tag[3];
303         struct tag      *t, *tt;
304         char             b[BUFSIZ], title[BUFSIZ];

306         b[0] = 0;
307         if (m->vol)
308                 (void)strlcat(b, m->vol, BUFSIZ);

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

314         PAIR_SUMMARY_INIT(&tag[0], "Document Header");
315         PAIR_CLASS_INIT(&tag[1], "head");
316         PAIR_INIT(&tag[2], ATTR_WIDTH, "100%");
317         t = print_otag(h, TAG_TABLE, 3, tag);
318         PAIR_INIT(&tag[0], ATTR_WIDTH, "30%");
319         print_otag(h, TAG_COL, 1, tag);
320         print_otag(h, TAG_COL, 1, tag);
321         print_otag(h, TAG_COL, 1, tag);

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

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



```
327         PAIR_CLASS_INIT(&tag[0], "head-ltitle");
328         print_otag(h, TAG_TD, 1, tag);
329         print_text(h, title);
330         print_stagq(h, tt);

332         PAIR_CLASS_INIT(&tag[0], "head-vol");
333         PAIR_INIT(&tag[1], ATTR_ALIGN, "center");
334         print_otag(h, TAG_TD, 2, tag);
335         print_text(h, b);
336         print_stagq(h, tt);

338         PAIR_CLASS_INIT(&tag[0], "head-rtitle");
339         PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
340         print_otag(h, TAG_TD, 2, tag);
341         print_text(h, title);
342         print_tagq(h, t);
343 }


346 /* ARGSUSED */
347 static void
348 man_root_post(MAN_ARGS)
349 {
350         struct htmlpair  tag[3];
351         struct tag      *t, *tt;

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

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

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

366         assert(m->date);
367         print_text(h, m->date);
368         print_stagq(h, tt);

370         PAIR_CLASS_INIT(&tag[0], "foot-os");
371         PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
372         print_otag(h, TAG_TD, 2, tag);

374         if (m->source)
375                 print_text(h, m->source);
376         print_tagq(h, t);
377 }


380 /* ARGSUSED */
381 static int
382 man_br_pre(MAN_ARGS)
383 {
384         struct roffsu    su;
385         struct htmlpair  tag;

387         SCALE_VS_INIT(&su, 1);

389         if (MAN_sp == n->tok) {
390                 if (NULL != (n = n->child))
391                         if ( ! a2roffsu(n->string, &su, SCALE_VS))
```

```
392                                 SCALE_VS_INIT(&su, atoi(n->string));
393         } else
394                 su.scale = 0;

396         bufinit(h);
397         bufcat_su(h, "height", &su);
398         PAIR_STYLE_INIT(&tag, h);
399         print_otag(h, TAG_DIV, 1, &tag);

401         /* So the div isn't empty: */
402         print_text(h, "\\~");

404         return(0);
405 }

407 /* ARGSUSED */
408 static int
409 man_SH_pre(MAN_ARGS)
410 {
411         struct htmlpair  tag;

413         if (MAN_BLOCK == n->type) {
414                 mh->fl &= ~MANH_LITERAL;
415                 PAIR_CLASS_INIT(&tag, "section");
416                 print_otag(h, TAG_DIV, 1, &tag);
417                 return(1);
418         } else if (MAN_BODY == n->type)
419                 return(1);

421         print_otag(h, TAG_H1, 0, NULL);
422         return(1);
423 }

425 /* ARGSUSED */
426 static int
427 man_alt_pre(MAN_ARGS)
428 {
429         const struct man_node   *nn;
430         int                i, savelit;
431         enum htmltag     fp;
432         struct tag      *t;

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

437         mh->fl &= ~MANH_LITERAL;

439         for (i = 0, nn = n->child; nn; nn = nn->next, i++) {
440                 t = NULL;
441                 switch (n->tok) {
442                 case (MAN_BI):
443                         fp = i % 2 ? TAG_I : TAG_B;
444                         break;
445                 case (MAN_IB):
446                         fp = i % 2 ? TAG_B : TAG_I;
447                         break;
448                 case (MAN_RI):
449                         fp = i % 2 ? TAG_I : TAG_MAX;
450                         break;
451                 case (MAN_IR):
452                         fp = i % 2 ? TAG_MAX : TAG_I;
453                         break;
454                 case (MAN_BR):
455                         fp = i % 2 ? TAG_MAX : TAG_B;
456                         break;
457                 case (MAN_RB):
```



```
458                         fp = i % 2 ? TAG_B : TAG_MAX;
459                         break;
460                 default:
461                         abort();
462                         /* NOTREACHED */
463                 }

465                 if (i)
466                         h->flags |= HTML_NOSPACE;

468                 if (TAG_MAX != fp)
469                         t = print_otag(h, fp, 0, NULL);

471                 print_man_node(m, nn, mh, h);

473                 if (t)
474                         print_tagq(h, t);
475         }

477         if (savelit)
478                 mh->fl |= MANH_LITERAL;

480         return(0);
481 }

483 /* ARGSUSED */
484 static int
485 man_SM_pre(MAN_ARGS)
486 {
487
488         print_otag(h, TAG_SMALL, 0, NULL);
489         if (MAN_SB == n->tok)
490                 print_otag(h, TAG_B, 0, NULL);
491         return(1);
492 }

494 /* ARGSUSED */
495 static int
496 man_SS_pre(MAN_ARGS)
497 {
498         struct htmlpair  tag;

500         if (MAN_BLOCK == n->type) {
501                 mh->fl &= ~MANH_LITERAL;
502                 PAIR_CLASS_INIT(&tag, "subsection");
503                 print_otag(h, TAG_DIV, 1, &tag);
504                 return(1);
505         } else if (MAN_BODY == n->type)
506                 return(1);

508         print_otag(h, TAG_H2, 0, NULL);
509         return(1);
510 }

512 /* ARGSUSED */
513 static int
514 man_PP_pre(MAN_ARGS)
515 {

517         if (MAN_HEAD == n->type)
518                 return(0);
519         else if (MAN_BLOCK == n->type)
520                 print_bvspace(h, n);

522         return(1);
523 }
```

```
525 /* ARGSUSED */
526 static int
527 man_IP_pre(MAN_ARGS)
528 {
529         const struct man_node   *nn;

531         if (MAN_BODY == n->type) {
532                 print_otag(h, TAG_DD, 0, NULL);
533                 return(1);
534         } else if (MAN_HEAD != n->type) {
535                 print_otag(h, TAG_DL, 0, NULL);
536                 return(1);
537         }

539         /* FIXME: width specification. */

541         print_otag(h, TAG_DT, 0, NULL);

543         /* For IP, only print the first header element. */

545         if (MAN_IP == n->tok && n->child)
546                 print_man_node(m, n->child, mh, h);

548         /* For TP, only print next-line header elements. */

550         if (MAN_TP == n->tok)
551                 for (nn = n->child; nn; nn = nn->next)
552                         if (nn->line > n->line)
553                                 print_man_node(m, nn, mh, h);

555         return(0);
556 }

558 /* ARGSUSED */
559 static int
560 man_HP_pre(MAN_ARGS)
561 {
562         struct htmlpair  tag;
563         struct roffsu    su;
564         const struct man_node *np;

566         if (MAN_HEAD == n->type)
567                 return(0);
568         else if (MAN_BLOCK != n->type)
569                 return(1);

571         np = n->head->child;

573         if (NULL == np || ! a2width(np, &su))
574                 SCALE_HS_INIT(&su, INDENT);

576         bufinit(h);

578         print_bvspace(h, n);
579         bufcat_su(h, "margin-left", &su);
580         su.scale = -su.scale;
581         bufcat_su(h, "text-indent", &su);
582         PAIR_STYLE_INIT(&tag, h);
583         print_otag(h, TAG_P, 1, &tag);
584         return(1);
585 }

587 /* ARGSUSED */
588 static int
589 man_OP_pre(MAN_ARGS)
```



```
590 {
591         struct tag      *tt;
592         struct htmlpair  tag;

594         print_text(h, "[");
595         h->flags |= HTML_NOSPACE;
596         PAIR_CLASS_INIT(&tag, "opt");
597         tt = print_otag(h, TAG_SPAN, 1, &tag);

599         if (NULL != (n = n->child)) {
600                 print_otag(h, TAG_B, 0, NULL);
601                 print_text(h, n->string);
602         }

604         print_stagq(h, tt);

606         if (NULL != n && NULL != n->next) {
607                 print_otag(h, TAG_I, 0, NULL);
608                 print_text(h, n->next->string);
609         }

611         print_stagq(h, tt);
612         h->flags |= HTML_NOSPACE;
613         print_text(h, "]");
614         return(0);
615 }


618 /* ARGSUSED */
619 static int
620 man_B_pre(MAN_ARGS)
621 {

623         print_otag(h, TAG_B, 0, NULL);
624         return(1);
625 }

627 /* ARGSUSED */
628 static int
629 man_I_pre(MAN_ARGS)
630 {
631
632         print_otag(h, TAG_I, 0, NULL);
633         return(1);
634 }

636 /* ARGSUSED */
637 static int
638 man_literal_pre(MAN_ARGS)
639 {

641         if (MAN_nf != n->tok) {
642                 print_otag(h, TAG_BR, 0, NULL);
643                 mh->fl &= ~MANH_LITERAL;
644         } else
645                 mh->fl |= MANH_LITERAL;

647         return(0);
648 }

650 /* ARGSUSED */
651 static int
652 man_in_pre(MAN_ARGS)
653 {

655         print_otag(h, TAG_BR, 0, NULL);
```

```
656         return(0);
657 }

659 /* ARGSUSED */
660 static int
661 man_ign_pre(MAN_ARGS)
662 {

664         return(0);
665 }

667 /* ARGSUSED */
668 static int
669 man_RS_pre(MAN_ARGS)
670 {
671         struct htmlpair  tag;
672         struct roffsu    su;

674         if (MAN_HEAD == n->type)
675                 return(0);
676         else if (MAN_BODY == n->type)
677                 return(1);

679         SCALE_HS_INIT(&su, INDENT);
680         if (n->head->child)
681                 a2width(n->head->child, &su);

683         bufinit(h);
684         bufcat_su(h, "margin-left", &su);
685         PAIR_STYLE_INIT(&tag, h);
686         print_otag(h, TAG_DIV, 1, &tag);
687         return(1);
688 }
```

```
**********************************************************
   10870 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man_macro.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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>
 *
 * 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_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 */
};

const   struct man_macro * const man_macros = __man_macros;


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

        assert(to);

        m->next = MAN_NEXT_SIBLING;

        /* LINTED */
        while (m->last != to) {
```

```
128                 /*
129                  * Save the parent here, because we may delete the
130                  * m->last node in the post-validation phase and reset
131                  * it to m->last->parent, causing a step in the closing
132                  * out to be lost.
133                  */
134                 n = m->last->parent;
135                 rew_warn(m, m->last, er);
136                 if ( ! man_valid_post(m))
137                         return(0);
138                 m->last = n;
139                 assert(m->last);
140         }
141
142         rew_warn(m, m->last, er);
143         if ( ! man_valid_post(m))
144                 return(0);
145
146         return(1);
147 }
148
149
150 static enum rew
151 rew_block(enum mant ntok, enum man_type type, const struct man_node *n)
152 {
153
154         if (MAN_BLOCK == type && ntok == n->parent->tok &&
155                         MAN_BODY == n->parent->type)
156                 return(REW_REWIND);
157         return(ntok == n->tok ? REW_HALT : REW_NOHALT);
158 }
159
160
161 /*
162  * There are three scope levels: scoped to the root (all), scoped to the
163  * section (all less sections), and scoped to subsections (all less
164  * sections and subsections).
165  */
166 static enum rew
167 rew_dohalt(enum mant tok, enum man_type type, const struct man_node *n)
168 {
169         enum rew         c;
170
171         /* We cannot progress beyond the root ever. */
172         if (MAN_ROOT == n->type)
173                 return(REW_HALT);
174
175         assert(n->parent);
176
177         /* Normal nodes shouldn't go to the level of the root. */
178         if (MAN_ROOT == n->parent->type)
179                 return(REW_REWIND);
180
181         /* Already-validated nodes should be closed out. */
182         if (MAN_VALID & n->flags)
183                 return(REW_NOHALT);
184
185         /* First: rewind to ourselves. */
186         if (type == n->type && tok == n->tok)
187                 return(REW_REWIND);
188
189         /*
190          * Next follow the implicit scope-smashings as defined by man.7:
191          * section, sub-section, etc.
192          */
```



```
194         switch (tok) {
195         case (MAN_SH):
196                 break;
197         case (MAN_SS):
198                 /* Rewind to a section, if a block. */
199                 if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
200                         return(c);
201                 break;
202         case (MAN_RS):
203                 /* Rewind to a subsection, if a block. */
204                 if (REW_NOHALT != (c = rew_block(MAN_SS, type, n)))
205                         return(c);
206                 /* Rewind to a section, if a block. */
207                 if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
208                         return(c);
209                 break;
210         default:
211                 /* Rewind to an offsetter, if a block. */
212                 if (REW_NOHALT != (c = rew_block(MAN_RS, type, n)))
213                         return(c);
214                 /* Rewind to a subsection, if a block. */
215                 if (REW_NOHALT != (c = rew_block(MAN_SS, type, n)))
216                         return(c);
217                 /* Rewind to a section, if a block. */
218                 if (REW_NOHALT != (c = rew_block(MAN_SH, type, n)))
219                         return(c);
220                 break;
221         }
222
223         return(REW_NOHALT);
224 }
225
226
227 /*
228  * Rewinding entails ascending the parse tree until a coherent point,
229  * for example, the 'SH' macro will close out any intervening 'SS'
230  * scopes.  When a scope is closed, it must be validated and actioned.
231  */
232 static int
233 rew_scope(enum man_type type, struct man *m, enum mant tok)
234 {
235         struct man_node *n;
236         enum rew         c;
237
238         /* LINTED */
239         for (n = m->last; n; n = n->parent) {
240                 /*
241                  * Whether we should stop immediately (REW_HALT), stop
242                  * and rewind until this point (REW_REWIND), or keep
243                  * rewinding (REW_NOHALT).
244                  */
245                 c = rew_dohalt(tok, type, n);
246                 if (REW_HALT == c)
247                         return(1);
248                 if (REW_REWIND == c)
249                         break;
250         }
251
252         /*
253          * Rewind until the current point.  Warn if we're a roff
254          * instruction that's mowing over explicit scopes.
255          */
256         assert(n);
257
258         return(man_unscope(m, n, MANDOCERR_MAX));
259 }
```

```
262 /*
263  * Close out a generic explicit macro.
264  */
265 /* ARGSUSED */
266 int
267 blk_close(MACRO_PROT_ARGS)
268 {
269         enum mant               ntok;
270         const struct man_node   *nn;

272         switch (tok) {
273         case (MAN_RE):
274                 ntok = MAN_RS;
275                 break;
276         default:
277                 abort();
278                 /* NOTREACHED */
279         }

281         for (nn = m->last->parent; nn; nn = nn->parent)
282                 if (ntok == nn->tok)
283                         break;

285         if (NULL == nn)
286                 man_pmsg(m, line, ppos, MANDOCERR_NOSCOPE);

288         if ( ! rew_scope(MAN_BODY, m, ntok))
289                 return(0);
290         if ( ! rew_scope(MAN_BLOCK, m, ntok))
291                 return(0);

293         return(1);
294 }


297 /* ARGSUSED */
298 int
299 blk_exp(MACRO_PROT_ARGS)
300 {
301         int              la;
302         char            *p;

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          */

310         if ( ! man_block_alloc(m, line, ppos, tok))
311                 return(0);
312         if ( ! man_head_alloc(m, line, ppos, tok))
313                 return(0);

315         for (;;) {
316                 la = *pos;
317                 if ( ! man_args(m, line, pos, buf, &p))
318                         break;
319                 if ( ! man_word_alloc(m, line, la, p))
320                         return(0);
321         }

323         assert(m);
324         assert(tok != MAN_MAX);
```



```
326         if ( ! rew_scope(MAN_HEAD, m, tok))
327                 return(0);
328         return(man_body_alloc(m, line, ppos, tok));
329 }



333 /*
334  * Parse an implicit-block macro.  These contain a MAN_HEAD and a
335  * MAN_BODY contained within a MAN_BLOCK.  Rules for closing out other
336  * scopes, such as 'SH' closing out an 'SS', are defined in the rew
337  * routines.
338  */
339 /* ARGSUSED */
340 int
341 blk_imp(MACRO_PROT_ARGS)
342 {
343         int              la;
344         char            *p;
345         struct man_node *n;

347         /* Close out prior scopes. */

349         if ( ! rew_scope(MAN_BODY, m, tok))
350                 return(0);
351         if ( ! rew_scope(MAN_BLOCK, m, tok))
352                 return(0);

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

356         if ( ! man_block_alloc(m, line, ppos, tok))
357                 return(0);
358         if ( ! man_head_alloc(m, line, ppos, tok))
359                 return(0);

361         n = m->last;

363         /* Add line arguments. */

365         for (;;) {
366                 la = *pos;
367                 if ( ! man_args(m, line, pos, buf, &p))
368                         break;
369                 if ( ! man_word_alloc(m, line, la, p))
370                         return(0);
371         }

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

375         if (MAN_SCOPED & man_macros[tok].flags) {
376                 /* If we're forcing scope ('TP'), keep it open. */
377                 if (MAN_FSCOPED & man_macros[tok].flags) {
378                         m->flags |= MAN_BLINE;
379                         return(1);
380                 } else if (n == m->last) {
381                         m->flags |= MAN_BLINE;
382                         return(1);
383                 }
384         }

386         if ( ! rew_scope(MAN_HEAD, m, tok))
387                 return(0);
388         return(man_body_alloc(m, line, ppos, tok));
389 }
```

```
/* ARGSUSED */
int
in_line_eoln(MACRO_PROT_ARGS)
{
        int              la;
        char            *p;
        struct man_node *n;

        if ( ! man_elem_alloc(m, line, ppos, tok))
                return(0);

        n = m->last;

        for (;;) {
                la = *pos;
                if ( ! man_args(m, line, pos, buf, &p))
                        break;
                if ( ! man_word_alloc(m, line, la, p))
                        return(0);
        }

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

        if (n == m->last && MAN_SCOPED & man_macros[tok].flags) {
                assert( ! (MAN_NSCOPED & man_macros[tok].flags));
                m->flags |= MAN_ELINE;
                return(1);
        }

        /* Set ignorable context, if applicable. */

        if (MAN_NSCOPED & man_macros[tok].flags) {
                assert( ! (MAN_SCOPED & man_macros[tok].flags));
                m->flags |= MAN_ILINE;
        }

        assert(MAN_ROOT != m->last->type);
        m->next = MAN_NEXT_SIBLING;

        /*
         * Rewind our element scope.  Note that when TH is pruned, we'll
         * be back at the root, so make sure that we don't clobber as
         * its sibling.
         */

        for ( ; m->last; m->last = m->last->parent) {
                if (m->last == n)
                        break;
                if (m->last->type == MAN_ROOT)
                        break;
                if ( ! man_valid_post(m))
                        return(0);
        }

        assert(m->last);

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

        if (m->last->type != MAN_ROOT && ! man_valid_post(m))
                return(0);
```



```
        return(1);
}


int
man_macroend(struct man *m)
{

        return(man_unscope(m, m->first, MANDOCERR_SCOPEEXIT));
}

static int
man_args(struct man *m, int line, int *pos, char *buf, char **v)
{
        char     *start;

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

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

        *v = mandoc_getarg(m->parse, v, line, pos);
        return(1);
}
```

```
**********************************************************
   21486 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man_term.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: man_term.c,v 1.127 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * 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 */
};

#define DECL_ARGS         struct termp *p, \
                          struct mtermp *mt, \
                          const struct man_node *n, \
                          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 *);

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_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_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  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 */
        { pre_ign, NULL, MAN_NOTEXT }, /* na */
        { pre_sp, NULL, MAN_NOTEXT }, /* sp */
        { pre_literal, NULL, 0 }, /* nf */
        { pre_literal, NULL, 0 }, /* fi */
        { NULL, NULL, 0 }, /* RE */
        { pre_RS, post_RS, 0 }, /* RS */
        { pre_ign, NULL, 0 }, /* DT */
        { pre_ign, NULL, 0 }, /* UC */
        { pre_ign, NULL, 0 }, /* PD */
        { pre_ign, NULL, 0 }, /* AT */
        { pre_in, NULL, MAN_NOTEXT }, /* in */
```

```
128         { pre_ft, NULL, MAN_NOTEXT }, /* ft */
129         { pre_OP, NULL, 0 }, /* OP */
130 };



134 void
135 terminal_man(void *arg, const struct man *man)
136 {
137         struct termp            *p;
138         const struct man_node   *n;
139         const struct man_meta   *m;
140         struct mtermp            mt;

142         p = (struct termp *)arg;

144         if (0 == p->defindent)
145                 p->defindent = 7;

147         p->overstep = 0;
148         p->maxrmargin = p->defrmargin;
149         p->tabwidth = term_len(p, 5);

151         if (NULL == p->symtab)
152                 p->symtab = mchars_alloc();

154         n = man_node(man);
155         m = man_meta(man);

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

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

162         mt.lmargin[mt.lmargincur] = term_len(p, p->defindent);
163         mt.offset = term_len(p, p->defindent);

165         if (n->child)
166                 print_man_nodelist(p, &mt, n->child, m);

168         term_end(p);
169 }


172 static size_t
173 a2height(const struct termp *p, const char *cp)
174 {
175         struct roffsu    su;

177         if ( ! a2roffsu(cp, &su, SCALE_VS))
178                 SCALE_VS_INIT(&su, atoi(cp));

180         return(term_vspan(p, &su));
181 }


184 static int
185 a2width(const struct termp *p, const char *cp)
186 {
187         struct roffsu    su;

189         if ( ! a2roffsu(cp, &su, SCALE_BU))
190                 return(-1);

192         return((int)term_hspan(p, &su));
193 }
```



```
195 /*
196  * Printing leading vertical space before a block.
197  * This is used for the paragraph macros.
198  * The rules are pretty simple, since there's very little nesting going
199  * on here.  Basically, if we're the first within another block (SS/SH),
200  * then don't emit vertical space.  If we are (RS), then do.  If not the
201  * first, print it.
202  */
203 static void
204 print_bvspace(struct termp *p, const struct man_node *n)
205 {

207         term_newln(p);

209         if (n->body && n->body->child)
210                 if (MAN_TBL == n->body->child->type)
211                         return;

213         if (MAN_ROOT == n->parent->type || MAN_RS != n->parent->tok)
214                 if (NULL == n->prev)
215                         return;

217         term_vspace(p);
218 }

220 /* ARGSUSED */
221 static int
222 pre_ign(DECL_ARGS)
223 {

225         return(0);
226 }


229 /* ARGSUSED */
230 static int
231 pre_I(DECL_ARGS)
232 {

234         term_fontrepl(p, TERMFONT_UNDER);
235         return(1);
236 }


239 /* ARGSUSED */
240 static int
241 pre_literal(DECL_ARGS)
242 {

244         term_newln(p);

246         if (MAN_nf == n->tok)
247                 mt->fl |= MANT_LITERAL;
248         else
249                 mt->fl &= ~MANT_LITERAL;

251         /*
252          * Unlike .IP and .TP, .HP does not have a HEAD.
253          * So in case a second call to term_flushln() is needed,
254          * indentation has to be set up explicitly.
255          */
256         if (MAN_HP == n->parent->tok && p->rmargin < p->maxrmargin) {
257                 p->offset = p->rmargin;
258                 p->rmargin = p->maxrmargin;
259                 p->flags &= ~(TERMP_NOBREAK | TERMP_TWOSPACE);
```

```
260                 p->flags |= TERMP_NOSPACE;
261         }

263         return(0);
264 }

266 /* ARGSUSED */
267 static int
268 pre_alternate(DECL_ARGS)
269 {
270         enum termfont            font[2];
271         const struct man_node   *nn;
272         int                      savelit, i;

274         switch (n->tok) {
275         case (MAN_RB):
276                 font[0] = TERMFONT_NONE;
277                 font[1] = TERMFONT_BOLD;
278                 break;
279         case (MAN_RI):
280                 font[0] = TERMFONT_NONE;
281                 font[1] = TERMFONT_UNDER;
282                 break;
283         case (MAN_BR):
284                 font[0] = TERMFONT_BOLD;
285                 font[1] = TERMFONT_NONE;
286                 break;
287         case (MAN_BI):
288                 font[0] = TERMFONT_BOLD;
289                 font[1] = TERMFONT_UNDER;
290                 break;
291         case (MAN_IR):
292                 font[0] = TERMFONT_UNDER;
293                 font[1] = TERMFONT_NONE;
294                 break;
295         case (MAN_IB):
296                 font[0] = TERMFONT_UNDER;
297                 font[1] = TERMFONT_BOLD;
298                 break;
299         default:
300                 abort();
301         }

303         savelit = MANT_LITERAL & mt->fl;
304         mt->fl &= ~MANT_LITERAL;

306         for (i = 0, nn = n->child; nn; nn = nn->next, i = 1 - i) {
307                 term_fontrepl(p, font[i]);
308                 if (savelit && NULL == nn->next)
309                         mt->fl |= MANT_LITERAL;
310                 print_man_node(p, mt, nn, m);
311                 if (nn->next)
312                         p->flags |= TERMP_NOSPACE;
313         }

315         return(0);
316 }

318 /* ARGSUSED */
319 static int
320 pre_B(DECL_ARGS)
321 {

323         term_fontrepl(p, TERMFONT_BOLD);
324         return(1);
325 }
```



```
327 /* ARGSUSED */
328 static int
329 pre_OP(DECL_ARGS)
330 {

332         term_word(p, "[");
333         p->flags |= TERMP_NOSPACE;

335         if (NULL != (n = n->child)) {
336                 term_fontrepl(p, TERMFONT_BOLD);
337                 term_word(p, n->string);
338         }
339         if (NULL != n && NULL != n->next) {
340                 term_fontrepl(p, TERMFONT_UNDER);
341                 term_word(p, n->next->string);
342         }

344         term_fontrepl(p, TERMFONT_NONE);
345         p->flags |= TERMP_NOSPACE;
346         term_word(p, "]");
347         return(0);
348 }

350 /* ARGSUSED */
351 static int
352 pre_ft(DECL_ARGS)
353 {
354         const char      *cp;

356         if (NULL == n->child) {
357                 term_fontlast(p);
358                 return(0);
359         }

361         cp = n->child->string;
362         switch (*cp) {
363         case ('4'):
364                 /* FALLTHROUGH */
365         case ('3'):
366                 /* FALLTHROUGH */
367         case ('B'):
368                 term_fontrepl(p, TERMFONT_BOLD);
369                 break;
370         case ('2'):
371                 /* FALLTHROUGH */
372         case ('I'):
373                 term_fontrepl(p, TERMFONT_UNDER);
374                 break;
375         case ('P'):
376                 term_fontlast(p);
377                 break;
378         case ('1'):
379                 /* FALLTHROUGH */
380         case ('C'):
381                 /* FALLTHROUGH */
382         case ('R'):
383                 term_fontrepl(p, TERMFONT_NONE);
384                 break;
385         default:
386                 break;
387         }
388         return(0);
389 }

391 /* ARGSUSED */
```

```
392 static int
393 pre_in(DECL_ARGS)
394 {
395         int              len, less;
396         size_t           v;
397         const char      *cp;

399         term_newln(p);

401         if (NULL == n->child) {
402                 p->offset = mt->offset;
403                 return(0);
404         }

406         cp = n->child->string;
407         less = 0;

409         if ('-' == *cp)
410                 less = -1;
411         else if ('+' == *cp)
412                 less = 1;
413         else
414                 cp--;

416         if ((len = a2width(p, ++cp)) < 0)
417                 return(0);

419         v = (size_t)len;

421         if (less < 0)
422                 p->offset -= p->offset > v ? v : p->offset;
423         else if (less > 0)
424                 p->offset += v;
425         else
426                 p->offset = v;

428         /* Don't let this creep beyond the right margin. */

430         if (p->offset > p->rmargin)
431                 p->offset = p->rmargin;

433         return(0);
434 }


437 /* ARGSUSED */
438 static int
439 pre_sp(DECL_ARGS)
440 {
441         size_t           i, len;

443         if ((NULL == n->prev && n->parent)) {
444                 if (MAN_SS == n->parent->tok)
445                         return(0);
446                 if (MAN_SH == n->parent->tok)
447                         return(0);
448         }

450         switch (n->tok) {
451         case (MAN_br):
452                 len = 0;
453                 break;
454         default:
455                 len = n->child ? a2height(p, n->child->string) : 1;
456                 break;
457         }
```



```
459         if (0 == len)
460                 term_newln(p);
461         for (i = 0; i < len; i++)
462                 term_vspace(p);

464         return(0);
465 }


468 /* ARGSUSED */
469 static int
470 pre_HP(DECL_ARGS)
471 {
472         size_t                   len, one;
473         int                      ival;
474         const struct man_node   *nn;

476         switch (n->type) {
477         case (MAN_BLOCK):
478                 print_bvspace(p, n);
479                 return(1);
480         case (MAN_BODY):
481                 p->flags |= TERMP_NOBREAK;
482                 p->flags |= TERMP_TWOSPACE;
483                 break;
484         default:
485                 return(0);
486         }

488         len = mt->lmargin[mt->lmargincur];
489         ival = -1;

491         /* Calculate offset. */

493         if (NULL != (nn = n->parent->head->child))
494                 if ((ival = a2width(p, nn->string)) >= 0)
495                         len = (size_t)ival;

497         one = term_len(p, 1);
498         if (len < one)
499                 len = one;

501         p->offset = mt->offset;
502         p->rmargin = mt->offset + len;

504         if (ival >= 0)
505                 mt->lmargin[mt->lmargincur] = (size_t)ival;

507         return(1);
508 }


511 /* ARGSUSED */
512 static void
513 post_HP(DECL_ARGS)
514 {

516         switch (n->type) {
517         case (MAN_BLOCK):
518                 term_flushln(p);
519                 break;
520         case (MAN_BODY):
521                 term_flushln(p);
522                 p->flags &= ~TERMP_NOBREAK;
523                 p->flags &= ~TERMP_TWOSPACE;
```

```
524                 p->offset = mt->offset;
525                 p->rmargin = p->maxrmargin;
526                 break;
527         default:
528                 break;
529         }
530 }


533 /* ARGSUSED */
534 static int
535 pre_PP(DECL_ARGS)
536 {

538         switch (n->type) {
539         case (MAN_BLOCK):
540                 mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
541                 print_bvspace(p, n);
542                 break;
543         default:
544                 p->offset = mt->offset;
545                 break;
546         }

548         return(MAN_HEAD != n->type);
549 }


552 /* ARGSUSED */
553 static int
554 pre_IP(DECL_ARGS)
555 {
556         const struct man_node   *nn;
557         size_t                   len;
558         int                      savelit, ival;

560         switch (n->type) {
561         case (MAN_BODY):
562                 p->flags |= TERMP_NOSPACE;
563                 break;
564         case (MAN_HEAD):
565                 p->flags |= TERMP_NOBREAK;
566                 break;
567         case (MAN_BLOCK):
568                 print_bvspace(p, n);
569                 /* FALLTHROUGH */
570         default:
571                 return(1);
572         }

574         len = mt->lmargin[mt->lmargincur];
575         ival = -1;

577         /* Calculate the offset from the optional second argument. */
578         if (NULL != (nn = n->parent->head->child))
579                 if (NULL != (nn = nn->next))
580                         if ((ival = a2width(p, nn->string)) >= 0)
581                                 len = (size_t)ival;

583         switch (n->type) {
584         case (MAN_HEAD):
585                 /* Handle zero-width lengths. */
586                 if (0 == len)
587                         len = term_len(p, 1);

589                 p->offset = mt->offset;
```



```
590                 p->rmargin = mt->offset + len;
591                 if (ival < 0)
592                         break;

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

597                 savelit = MANT_LITERAL & mt->fl;
598                 mt->fl &= ~MANT_LITERAL;

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

603                 if (savelit)
604                         mt->fl |= MANT_LITERAL;

606                 return(0);
607         case (MAN_BODY):
608                 p->offset = mt->offset + len;
609                 p->rmargin = p->maxrmargin;
610                 break;
611         default:
612                 break;
613         }

615         return(1);
616 }


619 /* ARGSUSED */
620 static void
621 post_IP(DECL_ARGS)
622 {

624         switch (n->type) {
625         case (MAN_HEAD):
626                 term_flushln(p);
627                 p->flags &= ~TERMP_NOBREAK;
628                 p->rmargin = p->maxrmargin;
629                 break;
630         case (MAN_BODY):
631                 term_newln(p);
632                 break;
633         default:
634                 break;
635         }
636 }


639 /* ARGSUSED */
640 static int
641 pre_TP(DECL_ARGS)
642 {
643         const struct man_node   *nn;
644         size_t                   len;
645         int                      savelit, ival;

647         switch (n->type) {
648         case (MAN_HEAD):
649                 p->flags |= TERMP_NOBREAK;
650                 break;
651         case (MAN_BODY):
652                 p->flags |= TERMP_NOSPACE;
653                 break;
654         case (MAN_BLOCK):
655                 print_bvspace(p, n);
```

```
656                 /* FALLTHROUGH */
657         default:
658                 return(1);
659         }

661         len = (size_t)mt->lmargin[mt->lmargincur];
662         ival = -1;

664         /* Calculate offset. */

666         if (NULL != (nn = n->parent->head->child))
667                 if (nn->string && nn->parent->line == nn->line)
668                         if ((ival = a2width(p, nn->string)) >= 0)
669                                 len = (size_t)ival;

671         switch (n->type) {
672         case (MAN_HEAD):
673                 /* Handle zero-length properly. */
674                 if (0 == len)
675                         len = term_len(p, 1);

677                 p->offset = mt->offset;
678                 p->rmargin = mt->offset + len;

680                 savelit = MANT_LITERAL & mt->fl;
681                 mt->fl &= ~MANT_LITERAL;

683                 /* Don't print same-line elements. */
684                 for (nn = n->child; nn; nn = nn->next)
685                         if (nn->line > n->line)
686                                 print_man_node(p, mt, nn, m);

688                 if (savelit)
689                         mt->fl |= MANT_LITERAL;
690                 if (ival >= 0)
691                         mt->lmargin[mt->lmargincur] = (size_t)ival;

693                 return(0);
694         case (MAN_BODY):
695                 p->offset = mt->offset + len;
696                 p->rmargin = p->maxrmargin;
697                 break;
698         default:
699                 break;
700         }

702         return(1);
703 }


706 /* ARGSUSED */
707 static void
708 post_TP(DECL_ARGS)
709 {

711         switch (n->type) {
712         case (MAN_HEAD):
713                 term_flushln(p);
714                 p->flags &= ~TERMP_NOBREAK;
715                 p->flags &= ~TERMP_TWOSPACE;
716                 p->rmargin = p->maxrmargin;
717                 break;
718         case (MAN_BODY):
719                 term_newln(p);
720                 break;
721         default:
```



```
722                 break;
723         }
724 }


727 /* ARGSUSED */
728 static int
729 pre_SS(DECL_ARGS)
730 {

732         switch (n->type) {
733         case (MAN_BLOCK):
734                 mt->fl &= ~MANT_LITERAL;
735                 mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
736                 mt->offset = term_len(p, p->defindent);
737                 /* If following a prior empty 'SS', no vspace. */
738                 if (n->prev && MAN_SS == n->prev->tok)
739                         if (NULL == n->prev->body->child)
740                                 break;
741                 if (NULL == n->prev)
742                         break;
743                 term_vspace(p);
744                 break;
745         case (MAN_HEAD):
746                 term_fontrepl(p, TERMFONT_BOLD);
747                 p->offset = term_len(p, p->defindent/2);
748                 break;
749         case (MAN_BODY):
750                 p->offset = mt->offset;
751                 break;
752         default:
753                 break;
754         }

756         return(1);
757 }


760 /* ARGSUSED */
761 static void
762 post_SS(DECL_ARGS)
763 {
764
765         switch (n->type) {
766         case (MAN_HEAD):
767                 term_newln(p);
768                 break;
769         case (MAN_BODY):
770                 term_newln(p);
771                 break;
772         default:
773                 break;
774         }
775 }


778 /* ARGSUSED */
779 static int
780 pre_SH(DECL_ARGS)
781 {

783         switch (n->type) {
784         case (MAN_BLOCK):
785                 mt->fl &= ~MANT_LITERAL;
786                 mt->lmargin[mt->lmargincur] = term_len(p, p->defindent);
787                 mt->offset = term_len(p, p->defindent);
```

```
788                 /* If following a prior empty ‘SH’, no vspace. */
789                 if (n->prev && MAN_SH == n->prev->tok)
790                         if (NULL == n->prev->body->child)
791                                 break;
792                 /* If the first macro, no vspae. */
793                 if (NULL == n->prev)
794                         break;
795                 term_vspace(p);
796                 break;
797         case (MAN_HEAD):
798                 term_fontrepl(p, TERMFONT_BOLD);
799                 p->offset = 0;
800                 break;
801         case (MAN_BODY):
802                 p->offset = mt->offset;
803                 break;
804         default:
805                 break;
806         }
807
808         return(1);
809 }
810
811
812 /* ARGSUSED */
813 static void
814 post_SH(DECL_ARGS)
815 {
816
817         switch (n->type) {
818         case (MAN_HEAD):
819                 term_newln(p);
820                 break;
821         case (MAN_BODY):
822                 term_newln(p);
823                 break;
824         default:
825                 break;
826         }
827 }
828
829 /* ARGSUSED */
830 static int
831 pre_RS(DECL_ARGS)
832 {
833         int              ival;
834         size_t           sz;
835
836         switch (n->type) {
837         case (MAN_BLOCK):
838                 term_newln(p);
839                 return(1);
840         case (MAN_HEAD):
841                 return(0);
842         default:
843                 break;
844         }
845
846         sz = term_len(p, p->defindent);
847
848         if (NULL != (n = n->parent->head->child))
849                 if ((ival = a2width(p, n->string)) >= 0)
850                         sz = (size_t)ival;
851
852         mt->offset += sz;
853         p->rmargin = p->maxrmargin;
```



```
854         p->offset = mt->offset < p->rmargin ? mt->offset : p->rmargin;
855
856         if (++mt->lmarginsz < MAXMARGINS)
857                 mt->lmargincur = mt->lmarginsz;
858
859         mt->lmargin[mt->lmargincur] = mt->lmargin[mt->lmargincur - 1];
860         return(1);
861 }
862
863 /* ARGSUSED */
864 static void
865 post_RS(DECL_ARGS)
866 {
867         int              ival;
868         size_t           sz;
869
870         switch (n->type) {
871         case (MAN_BLOCK):
872                 return;
873         case (MAN_HEAD):
874                 return;
875         default:
876                 term_newln(p);
877                 break;
878         }
879
880         sz = term_len(p, p->defindent);
881
882         if (NULL != (n = n->parent->head->child))
883                 if ((ival = a2width(p, n->string)) >= 0)
884                         sz = (size_t)ival;
885
886         mt->offset = mt->offset < sz ?  0 : mt->offset - sz;
887         p->offset = mt->offset;
888
889         if (--mt->lmarginsz < MAXMARGINS)
890                 mt->lmargincur = mt->lmarginsz;
891 }
892
893 static void
894 print_man_node(DECL_ARGS)
895 {
896         size_t           rm, rmax;
897         int              c;
898
899         switch (n->type) {
900         case(MAN_TEXT):
901                 /*
902                  * If we have a blank line, output a vertical space.
903                  * If we have a space as the first character, break
904                  * before printing the line’s data.
905                  */
906                 if (’\0’ == *n->string) {
907                         term_vspace(p);
908                         return;
909                 } else if (’ ’ == *n->string && MAN_LINE & n->flags)
910                         term_newln(p);
911
912                 term_word(p, n->string);
913
914                 /*
915                  * If we’re in a literal context, make sure that words
916                  * togehter on the same line stay together.  This is a
917                  * 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;
936         case (MAN_EQN):
937                 term_eqn(p, n->eqn);
938                 return;
939         case (MAN_TBL):
940                 /*
941                  * Tables are preceded by a newline.  Then process a
942                  * table line, which will cause line termination,
943                  */
944                 if (TBL_SPAN_FIRST & n->span->flags)
945                         term_newln(p);
946                 term_tbl(p, n->span);
947                 return;
948         default:
949                 break;
950         }

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

955         c = 1;
956         if (termacts[n->tok].pre)
957                 c = (*termacts[n->tok].pre)(p, mt, n, m);

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

962         if (termacts[n->tok].post)
963                 (*termacts[n->tok].post)(p, mt, n, m);
964         if ( ! (MAN_NOTEXT & termacts[n->tok].flags))
965                 term_fontrepl(p, TERMFONT_NONE);

967         if (MAN_EOS & n->flags)
968                 p->flags |= TERMP_SENTENCE;
969 }


972 static void
973 print_man_nodelist(DECL_ARGS)
974 {

976         print_man_node(p, mt, n, m);
977         if ( ! n->next)
978                 return;
979         print_man_nodelist(p, mt, n->next, m);
980 }


983 static void
984 print_man_foot(struct termp *p, const void *arg)
985 {
```



```
986         char            title[BUFSIZ];
987         size_t          datelen;
988         const struct man_meta *meta;

990         meta = (const struct man_meta *)arg;
991         assert(meta->title);
992         assert(meta->msec);
993         assert(meta->date);

995         term_fontrepl(p, TERMFONT_NONE);

997         term_vspace(p);

999         /*
1000         * Temporary, undocumented option to imitate mdoc(7) output.
1001         * In the bottom right corner, use the source instead of
1002         * the title.
1003         */

1005        if ( ! p->mdocstyle) {
1006                term_vspace(p);
1007                term_vspace(p);
1008                snprintf(title, BUFSIZ, "%s(%s)", meta->title, meta->msec);
1009        } else if (meta->source) {
1010                strlcpy(title, meta->source, BUFSIZ);
1011        } else {
1012                title[0] = '\0';
1013        }
1014        datelen = term_strlen(p, meta->date);

1016        /* Bottom left corner: manual source. */

1018        p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;
1019        p->offset = 0;
1020        p->rmargin = (p->maxrmargin - datelen + term_len(p, 1)) / 2;

1022        if (meta->source)
1023                term_word(p, meta->source);
1024        term_flushln(p);

1026        /* At the bottom in the middle: manual date. */

1028        p->flags |= TERMP_NOSPACE;
1029        p->offset = p->rmargin;
1030        p->rmargin = p->maxrmargin - term_strlen(p, title);
1031        if (p->offset + datelen >= p->rmargin)
1032                p->rmargin = p->offset + datelen;

1034        term_word(p, meta->date);
1035        term_flushln(p);

1037        /* Bottom right corner: manual title and section. */

1039        p->flags &= ~TERMP_NOBREAK;
1040        p->flags |= TERMP_NOSPACE;
1041        p->offset = p->rmargin;
1042        p->rmargin = p->maxrmargin;

1044        term_word(p, title);
1045        term_flushln(p);
1046 }


1049 static void
1050 print_man_head(struct termp *p, const void *arg)
1051 {
```

```
1052         char            buf[BUFSIZ], title[BUFSIZ];
1053         size_t          buflen, titlen;
1054         const struct man_meta *m;

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

1060         if (m->vol)
1061                 strlcpy(buf, m->vol, BUFSIZ);
1062         else
1063                 buf[0] = '\0';
1064         buflen = term_strlen(p, buf);

1066         /* Top left corner: manual title and section. */

1068         snprintf(title, BUFSIZ, "%s(%s)", m->title, m->msec);
1069         titlen = term_strlen(p, title);

1071         p->flags |= TERMP_NOBREAK | TERMP_NOSPACE;
1072         p->offset = 0;
1073         p->rmargin = 2 * (titlen+1) + buflen < p->maxrmargin ?
1074             (p->maxrmargin -
1075              term_strlen(p, buf) + term_len(p, 1)) / 2 :
1076             p->maxrmargin - buflen;

1078         term_word(p, title);
1079         term_flushln(p);

1081         /* At the top in the middle: manual volume. */

1083         p->flags |= TERMP_NOSPACE;
1084         p->offset = p->rmargin;
1085         p->rmargin = p->offset + buflen + titlen < p->maxrmargin ?
1086             p->maxrmargin - titlen : p->maxrmargin;

1088         term_word(p, buf);
1089         term_flushln(p);

1091         /* Top right corner: title and section, again. */

1093         p->flags &= ~TERMP_NOBREAK;
1094         if (p->rmargin + titlen <= p->maxrmargin) {
1095                 p->flags |= TERMP_NOSPACE;
1096                 p->offset = p->rmargin;
1097                 p->rmargin = p->maxrmargin;
1098                 term_word(p, title);
1099                 term_flushln(p);
1100         }

1102         p->flags &= ~TERMP_NOSPACE;
1103         p->offset = 0;
1104         p->rmargin = p->maxrmargin;

1106         /*
1107          * Groff prints three blank lines before the content.
1108          * Do the same, except in the temporary, undocumented
1109          * mode imitating mdoc(7) output.
1110          */

1112         term_vspace(p);
1113         if ( ! p->mdocstyle) {
1114                 term_vspace(p);
1115                 term_vspace(p);
1116         }
1117 }
```

```
**********************************************************
   11290 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/man_validate.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: man_validate.c,v 1.80 2012/01/03 15:16:24 kristaps Exp $ */
/*
 * 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 <sys/types.h>

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

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

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

typedef int     (*v_check)(CHKARGS);

struct  man_valid {
        v_check  *pres;
        v_check  *posts;
};

static  int       check_eq0(CHKARGS);
static  int       check_eq2(CHKARGS);
static  int       check_le1(CHKARGS);
static  int       check_ge2(CHKARGS);
static  int       check_le5(CHKARGS);
static  int       check_par(CHKARGS);
static  int       check_part(CHKARGS);
static  int       check_root(CHKARGS);
static  void      check_text(CHKARGS);

static  int       post_AT(CHKARGS);
static  int       post_vs(CHKARGS);
static  int       post_fi(CHKARGS);
static  int       post_ft(CHKARGS);
static  int       post_nf(CHKARGS);
```



```
static  int       post_sec(CHKARGS);
static  int       post_TH(CHKARGS);
static  int       post_UC(CHKARGS);
static  int       pre_sec(CHKARGS);

static  v_check   posts_at[] = { post_AT, NULL };
static  v_check   posts_br[] = { post_vs, check_eq0, NULL };
static  v_check   posts_eq0[] = { check_eq0, NULL };
static  v_check   posts_eq2[] = { check_eq2, NULL };
static  v_check   posts_fi[] = { check_eq0, post_fi, NULL };
static  v_check   posts_ft[] = { post_ft, NULL };
static  v_check   posts_nf[] = { check_eq0, post_nf, NULL };
static  v_check   posts_par[] = { check_par, NULL };
static  v_check   posts_part[] = { check_part, NULL };
static  v_check   posts_sec[] = { post_sec, NULL };
static  v_check   posts_sp[] = { post_vs, check_le1, NULL };
static  v_check   posts_th[] = { check_ge2, check_le5, post_TH, NULL };
static  v_check   posts_uc[] = { post_UC, NULL };
static  v_check   pres_sec[] = { pre_sec, NULL };

static  const struct man_valid man_valids[MAN_MAX] = {
        { NULL, posts_br }, /* br */
        { NULL, posts_th }, /* TH */
        { pres_sec, posts_sec }, /* SH */
        { pres_sec, posts_sec }, /* SS */
        { NULL, NULL }, /* TP */
        { NULL, posts_par }, /* LP */
        { NULL, posts_par }, /* PP */
        { NULL, posts_par }, /* P */
        { NULL, NULL }, /* IP */
        { NULL, NULL }, /* HP */
        { NULL, NULL }, /* SM */
        { NULL, NULL }, /* SB */
        { NULL, NULL }, /* BI */
        { NULL, NULL }, /* IB */
        { NULL, NULL }, /* BR */
        { NULL, NULL }, /* RB */
        { NULL, NULL }, /* R */
        { NULL, NULL }, /* B */
        { NULL, NULL }, /* I */
        { NULL, NULL }, /* IR */
        { NULL, NULL }, /* RI */
        { NULL, posts_eq0 }, /* na */
        { NULL, posts_sp }, /* sp */
        { NULL, posts_nf }, /* nf */
        { NULL, posts_fi }, /* fi */
        { NULL, NULL }, /* RE */
        { NULL, posts_part }, /* RS */
        { NULL, NULL }, /* DT */
        { NULL, posts_uc }, /* UC */
        { NULL, NULL }, /* PD */
        { NULL, posts_at }, /* AT */
        { NULL, NULL }, /* in */
        { NULL, posts_ft }, /* ft */
        { NULL, posts_eq2 }, /* OP */
};


int
man_valid_pre(struct man *m, struct man_node *n)
{
        v_check        *cp;

        switch (n->type) {
        case (MAN_TEXT):
                /* FALLTHROUGH */
```

```
128         case (MAN_ROOT):
129                 /* FALLTHROUGH */
130         case (MAN_EQN):
131                 /* FALLTHROUGH */
132         case (MAN_TBL):
133                 return(1);
134         default:
135                 break;
136         }

138         if (NULL == (cp = man_valids[n->tok].pres))
139                 return(1);
140         for ( ; *cp; cp++)
141                 if ( ! (*cp)(m, n))
142                         return(0);
143         return(1);
144 }


147 int
148 man_valid_post(struct man *m)
149 {
150         v_check         *cp;

152         if (MAN_VALID & m->last->flags)
153                 return(1);
154         m->last->flags |= MAN_VALID;

156         switch (m->last->type) {
157         case (MAN_TEXT):
158                 check_text(m, m->last);
159                 return(1);
160         case (MAN_ROOT):
161                 return(check_root(m, m->last));
162         case (MAN_EQN):
163                 /* FALLTHROUGH */
164         case (MAN_TBL):
165                 return(1);
166         default:
167                 break;
168         }

170         if (NULL == (cp = man_valids[m->last->tok].posts))
171                 return(1);
172         for ( ; *cp; cp++)
173                 if ( ! (*cp)(m, m->last))
174                         return(0);

176         return(1);
177 }


180 static int
181 check_root(CHKARGS)
182 {

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);

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

192         if (NULL == m->first->child) {
193                 man_nmsg(m, n, MANDOCERR_NODOCBODY);
```



```
194                 return(0);
195         } else if (NULL == m->meta.title) {
196                 man_nmsg(m, n, MANDOCERR_NOTITLE);

198                 /*
199                  * If a title hasn't been set, do so now (by
200                  * implication, date and section also aren't set).
201                  */

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);
207         }

209         return(1);
210 }

212 static void
213 check_text(CHKARGS)
214 {
215         char            *cp, *p;

217         if (MAN_LITERAL & m->flags)
218                 return;

220         cp = n->string;
221         for (p = cp; NULL != (p = strchr(p, '\t')); p++)
222                 man_pmsg(m, n->line, (int)(p - cp), MANDOCERR_BADTAB);
223 }

225 #define INEQ_DEFINE(x, ineq, name) \
226 static int \
227 check_##name(CHKARGS) \
228 { \
229         if (n->nchild ineq (x)) \
230                 return(1); \
231         mandoc_vmsg(MANDOCERR_ARGCOUNT, m->parse, n->line, n->pos, \
232                         "line arguments %s %d (have %d)", \
233                         #ineq, (x), n->nchild); \
234         return(1); \
235 }

237 INEQ_DEFINE(0, ==, eq0)
238 INEQ_DEFINE(2, ==, eq2)
239 INEQ_DEFINE(1, <=, le1)
240 INEQ_DEFINE(2, >=, ge2)
241 INEQ_DEFINE(5, <=, le5)

243 static int
244 post_ft(CHKARGS)
245 {
246         char    *cp;
247         int      ok;

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

252         ok = 0;
253         cp = n->child->string;
254         switch (*cp) {
255         case ('1'):
256                 /* FALLTHROUGH */
257         case ('2'):
258                 /* FALLTHROUGH */
259         case ('3'):
```

```
260                 /* FALLTHROUGH */
261         case ('4'):
262                 /* FALLTHROUGH */
263         case ('I'):
264                 /* FALLTHROUGH */
265         case ('P'):
266                 /* FALLTHROUGH */
267         case ('R'):
268                 if ('\0' == cp[1])
269                         ok = 1;
270                 break;
271         case ('B'):
272                 if ('\0' == cp[1] || ('I' == cp[1] && '\0' == cp[2]))
273                         ok = 1;
274                 break;
275         case ('C'):
276                 if ('W' == cp[1] && '\0' == cp[2])
277                         ok = 1;
278                 break;
279         default:
280                 break;
281         }

283         if (0 == ok) {
284                 mandoc_vmsg
285                         (MANDOCERR_BADFONT, m->parse,
286                          n->line, n->pos, "%s", cp);
287                 *cp = '\0';
288         }

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

296         return(1);
297 }

299 static int
300 pre_sec(CHKARGS)
301 {

303         if (MAN_BLOCK == n->type)
304                 m->flags &= ~MAN_LITERAL;
305         return(1);
306 }

308 static int
309 post_sec(CHKARGS)
310 {

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

315         man_nmsg(m, n, MANDOCERR_SYNTARGCOUNT);
316         return(0);
317 }

319 static int
320 check_part(CHKARGS)
321 {

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



```
327         return(1);
328 }


331 static int
332 check_par(CHKARGS)
333 {

335         switch (n->type) {
336         case (MAN_BLOCK):
337                 if (0 == n->body->nchild)
338                         man_node_delete(m, n);
339                 break;
340         case (MAN_BODY):
341                 if (0 == n->nchild)
342                         man_nmsg(m, n, MANDOCERR_IGNPAR);
343                 break;
344         case (MAN_HEAD):
345                 if (n->nchild)
346                         man_nmsg(m, n, MANDOCERR_ARGSLOST);
347                 break;
348         default:
349                 break;
350         }

352         return(1);
353 }


356 static int
357 post_TH(CHKARGS)
358 {
359         const char      *p;
360         int              line, pos;

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);

373         line = n->line;
374         pos = n->pos;
375         m->meta.title = m->meta.vol = m->meta.date =
376                 m->meta.msec = m->meta.source = NULL;

378         /* ->TITLE<- MSEC DATE SOURCE VOL */

380         n = n->child;
381         if (n && n->string) {
382                 for (p = n->string; '\0' != *p; p++) {
383                         /* Only warn about this once... */
384                         if (isalpha((unsigned char)*p) &&
385                                         ! isupper((unsigned char)*p)) {
386                                 man_nmsg(m, n, MANDOCERR_UPPERCASE);
387                                 break;
388                         }
389                 }
390                 m->meta.title = mandoc_strdup(n->string);
391         } else
```

```
392                 m->meta.title = mandoc_strdup("");

394         /* TITLE ->MSEC<- DATE SOURCE VOL */

396         if (n)
397                 n = n->next;
398         if (n && n->string)
399                 m->meta.msec = mandoc_strdup(n->string);
400         else
401                 m->meta.msec = mandoc_strdup("");

403         /* TITLE MSEC ->DATE<- SOURCE VOL */

405         if (n)
406                 n = n->next;
407         if (n && n->string && '\0' != n->string[0]) {
408                 pos = n->pos;
409                 m->meta.date = mandoc_normdate
410                     (m->parse, n->string, line, pos);
411         } else
412                 m->meta.date = mandoc_strdup("");

414         /* TITLE MSEC DATE ->SOURCE<- VOL */

416         if (n && (n = n->next))
417                 m->meta.source = mandoc_strdup(n->string);

419         /* TITLE MSEC DATE SOURCE ->VOL<- */
420         /* If missing, use the default VOL name for MSEC. */

422         if (n && (n = n->next))
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);

428         /*
429          * Remove the 'TH' node after we've processed it for our
430          * meta-data.
431          */
432         man_node_delete(m, m->last);
433         return(1);
434 }

436 static int
437 post_nf(CHKARGS)
438 {

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

443         m->flags |= MAN_LITERAL;
444         return(1);
445 }

447 static int
448 post_fi(CHKARGS)
449 {

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

454         m->flags &= ~MAN_LITERAL;
455         return(1);
456 }
```



```
458 static int
459 post_UC(CHKARGS)
460 {
461         static const char * const bsd_versions[] = {
462             "3rd Berkeley Distribution",
463             "4th Berkeley Distribution",
464             "4.2 Berkeley Distribution",
465             "4.3 Berkeley Distribution",
466             "4.4 Berkeley Distribution",
467         };

469         const char      *p, *s;

471         n = n->child;

473         if (NULL == n || MAN_TEXT != n->type)
474                 p = bsd_versions[0];
475         else {
476                 s = n->string;
477                 if (0 == strcmp(s, "3"))
478                         p = bsd_versions[0];
479                 else if (0 == strcmp(s, "4"))
480                         p = bsd_versions[1];
481                 else if (0 == strcmp(s, "5"))
482                         p = bsd_versions[2];
483                 else if (0 == strcmp(s, "6"))
484                         p = bsd_versions[3];
485                 else if (0 == strcmp(s, "7"))
486                         p = bsd_versions[4];
487                 else
488                         p = bsd_versions[0];
489         }

491         if (m->meta.source)
492                 free(m->meta.source);

494         m->meta.source = mandoc_strdup(p);
495         return(1);
496 }

498 static int
499 post_AT(CHKARGS)
500 {
501         static const char * const unix_versions[] = {
502             "7th Edition",
503             "System III",
504             "System V",
505             "System V Release 2",
506         };

508         const char      *p, *s;
509         struct man_node *nn;

511         n = n->child;

513         if (NULL == n || MAN_TEXT != n->type)
514                 p = unix_versions[0];
515         else {
516                 s = n->string;
517                 if (0 == strcmp(s, "3"))
518                         p = unix_versions[0];
519                 else if (0 == strcmp(s, "4"))
520                         p = unix_versions[1];
521                 else if (0 == strcmp(s, "5")) {
522                         nn = n->next;
523                         if (nn && MAN_TEXT == nn->type && nn->string[0])
```

```
524                                 p = unix_versions[3];
525                         else
526                                 p = unix_versions[2];
527                 } else
528                         p = unix_versions[0];
529         }

531         if (m->meta.source)
532                 free(m->meta.source);

534         m->meta.source = mandoc_strdup(p);
535         return(1);
536 }

538 static int
539 post_vs(CHKARGS)
540 {

542         /*
543          * Don't warn about this because it occurs in pod2man and would
544          * cause considerable (unfixable) warnage.
545          */
546         if (NULL == n->prev && MAN_ROOT == n->parent->type)
547                 man_node_delete(m, n);

549         return(1);
550 }
```

```
**********************************************************
   14130 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mandoc.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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 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 *);

/*
 * 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)
{
        int              i;
        size_t           sz;
        const char      *cp;

        i = 0;

        /* The expression consists of a subexpression. */

        if ('\\' == start[i]) {
                cp = &start[++i];
                /*
                 * 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 ('(' != start[i++])
                return(0);

        /*
         * 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;
        term = numeric = '\0';

        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;
                lim = 2;
                break;
        case ('['):
                gly = ESCAPE_SPECIAL;
```

```
128                 /*
129                  * Unicode escapes are defined in groff as \[uXXXX] to
130                  * \[u10FFFF], where the contained value must be a valid
131                  * Unicode codepoint.  Here, however, only check whether
132                  * it's not a zero-width escape.
133                  */
134                 if ('u' == cp[i] && ']' != cp[i + 1])
135                         gly = ESCAPE_UNICODE;
136                 term = ']';
137                 break;
138         case ('C'):
139                 if ('\'' != cp[i])
140                         return(ESCAPE_ERROR);
141                 gly = ESCAPE_SPECIAL;
142                 term = '\'';
143                 break;
144
145         /*
146          * Handle all triggers matching \X(xy, \Xx, and \X[xxxx], where
147          * 'X' is the trigger.  These have opaque sub-strings.
148          */
149         case ('F'):
150                 /* FALLTHROUGH */
151         case ('g'):
152                 /* FALLTHROUGH */
153         case ('k'):
154                 /* FALLTHROUGH */
155         case ('M'):
156                 /* FALLTHROUGH */
157         case ('m'):
158                 /* FALLTHROUGH */
159         case ('n'):
160                 /* FALLTHROUGH */
161         case ('V'):
162                 /* FALLTHROUGH */
163         case ('Y'):
164                 gly = ESCAPE_IGNORE;
165                 /* FALLTHROUGH */
166         case ('f'):
167                 if (ESCAPE_ERROR == gly)
168                         gly = ESCAPE_FONT;
169
170                 rstart= &cp[i];
171                 if (start)
172                         *start = rstart;
173
174                 switch (cp[i++]) {
175                 case ('('):
176                         lim = 2;
177                         break;
178                 case ('['):
179                         term = ']';
180                         break;
181                 default:
182                         lim = 1;
183                         i--;
184                         break;
185                 }
186                 break;
187
188         /*
189          * These escapes are of the form \X'Y', where 'X' is the trigger
190          * and 'Y' is any string.  These have opaque sub-strings.
191          */
192         case ('A'):
193                 /* FALLTHROUGH */
```



```
194         case ('b'):
195                 /* FALLTHROUGH */
196         case ('D'):
197                 /* FALLTHROUGH */
198         case ('o'):
199                 /* FALLTHROUGH */
200         case ('R'):
201                 /* FALLTHROUGH */
202         case ('X'):
203                 /* FALLTHROUGH */
204         case ('Z'):
205                 if ('\'' != cp[i++])
206                         return(ESCAPE_ERROR);
207                 gly = ESCAPE_IGNORE;
208                 term = '\'';
209                 break;
210
211         /*
212          * These escapes are of the form \X'N', where 'X' is the trigger
213          * and 'N' resolves to a numerical expression.
214          */
215         case ('B'):
216                 /* FALLTHROUGH */
217         case ('h'):
218                 /* FALLTHROUGH */
219         case ('H'):
220                 /* FALLTHROUGH */
221         case ('L'):
222                 /* FALLTHROUGH */
223         case ('l'):
224                 gly = ESCAPE_NUMBERED;
225                 /* FALLTHROUGH */
226         case ('S'):
227                 /* FALLTHROUGH */
228         case ('v'):
229                 /* FALLTHROUGH */
230         case ('w'):
231                 /* FALLTHROUGH */
232         case ('x'):
233                 if (ESCAPE_ERROR == gly)
234                         gly = ESCAPE_IGNORE;
235                 if ('\'' != cp[i++])
236                         return(ESCAPE_ERROR);
237                 term = numeric = '\'';
238                 break;
239
240         /*
241          * Special handling for the numbered character escape.
242          * XXX Do any other escapes need similar handling?
243          */
244         case ('N'):
245                 if ('\0' == cp[i])
246                         return(ESCAPE_ERROR);
247                 *end = &cp[++i];
248                 if (isdigit((unsigned char)cp[i-1]))
249                         return(ESCAPE_IGNORE);
250                 while (isdigit((unsigned char)**end))
251                         (*end)++;
252                 if (start)
253                         *start = &cp[i];
254                 if (sz)
255                         *sz = *end - &cp[i];
256                 if ('\0' != **end)
257                         (*end)++;
258                 return(ESCAPE_NUMBERED);
```

```
260         /*
261          * Sizes get a special category of their own.
262          */
263         case ('s'):
264                 gly = ESCAPE_IGNORE;

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

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

275                 switch (cp[i++]) {
276                 case ('('):
277                         lim = 2;
278                         break;
279                 case ('['):
280                         term = numeric = ']';
281                         break;
282                 case ('\''):
283                         term = numeric = '\'';
284                         break;
285                 default:
286                         lim = 1;
287                         i--;
288                         break;
289                 }

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

296                 break;

298         /*
299          * Anything else is assumed to be a glyph.
300          */
301         default:
302                 gly = ESCAPE_SPECIAL;
303                 lim = 1;
304                 i--;
305                 break;
306         }

308         assert(ESCAPE_ERROR != gly);

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

314         /*
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.
320          */

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          */

336         if ('\0' != term) {
337                 *end = strchr(&cp[i], term);
338                 if ('\0' == *end)
339                         return(ESCAPE_ERROR);

341                 rlim = *end - &cp[i];
342                 if (sz)
343                         *sz = rlim;
344                 (*end)++;
345                 goto out;
346         }

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]))
357                 return(ESCAPE_ERROR);

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

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

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

368         /* Run post-processors. */

370         switch (gly) {
371         case (ESCAPE_FONT):
372                 /*
373                  * Pretend that the constant-width font modes are the
374                  * same as the regular font modes.
375                  */
376                 if (2 == rlim && 'C' == *rstart)
377                         rstart++;
378                 else if (1 != rlim)
379                         break;

381                 switch (*rstart) {
382                 case ('3'):
383                         /* FALLTHROUGH */
384                 case ('B'):
385                         gly = ESCAPE_FONTBOLD;
386                         break;
387                 case ('2'):
388                         /* FALLTHROUGH */
389                 case ('I'):
390                         gly = ESCAPE_FONTITALIC;
391                         break;
```

```
392                 case ('P'):
393                         gly = ESCAPE_FONTPREV;
394                         break;
395                 case ('1'):
396                         /* FALLTHROUGH */
397                 case ('R'):
398                         gly = ESCAPE_FONTROMAN;
399                         break;
400                 }
401                 break;
402         case (ESCAPE_SPECIAL):
403                 if (1 != rlim)
404                         break;
405                 if ('c' == *rstart)
406                         gly = ESCAPE_NOSPACE;
407                 break;
408         default:
409                 break;
410         }

412         return(gly);
413 }

415 void *
416 mandoc_calloc(size_t num, size_t size)
417 {
418         void            *ptr;

420         ptr = calloc(num, size);
421         if (NULL == ptr) {
422                 perror(NULL);
423                 exit((int)MANDOCLEVEL_SYSERR);
424         }

426         return(ptr);
427 }


430 void *
431 mandoc_malloc(size_t size)
432 {
433         void            *ptr;

435         ptr = malloc(size);
436         if (NULL == ptr) {
437                 perror(NULL);
438                 exit((int)MANDOCLEVEL_SYSERR);
439         }

441         return(ptr);
442 }


445 void *
446 mandoc_realloc(void *ptr, size_t size)
447 {

449         ptr = realloc(ptr, size);
450         if (NULL == ptr) {
451                 perror(NULL);
452                 exit((int)MANDOCLEVEL_SYSERR);
453         }

455         return(ptr);
456 }
```



```
458 char *
459 mandoc_strndup(const char *ptr, size_t sz)
460 {
461         char            *p;

463         p = mandoc_malloc(sz + 1);
464         memcpy(p, ptr, sz);
465         p[(int)sz] = '\0';
466         return(p);
467 }

469 char *
470 mandoc_strdup(const char *ptr)
471 {
472         char            *p;

474         p = strdup(ptr);
475         if (NULL == p) {
476                 perror(NULL);
477                 exit((int)MANDOCLEVEL_SYSERR);
478         }

480         return(p);
481 }

483 /*
484  * Parse a quoted or unquoted roff-style request or macro argument.
485  * Return a pointer to the parsed argument, which is either the original
486  * pointer or advanced by one byte in case the argument is quoted.
487  * Null-terminate the argument in place.
488  * Collapse pairs of quotes inside quoted arguments.
489  * Advance the argument pointer to the next argument,
490  * or to the null byte terminating the argument line.
491  */
492 char *
493 mandoc_getarg(struct mparse *parse, char **cpp, int ln, int *pos)
494 {
495         char     *start, *cp;
496         int       quoted, pairs, white;

498         /* Quoting can only start with a new word. */
499         start = *cpp;
500         quoted = 0;
501         if ('"' == *start) {
502                 quoted = 1;
503                 start++;
504         }

506         pairs = 0;
507         white = 0;
508         for (cp = start; '\0' != *cp; cp++) {
509                 /* Move left after quoted quotes and escaped backslashes. */
510                 if (pairs)
511                         cp[-pairs] = cp[0];
512                 if ('\\' == cp[0]) {
513                         if ('\\' == cp[1]) {
514                                 /* Poor man's copy mode. */
515                                 pairs++;
516                                 cp++;
517                         } else if (0 == quoted && ' ' == cp[1])
518                                 /* Skip escaped blanks. */
519                                 cp++;
520                 } else if (0 == quoted) {
521                         if (' ' == cp[0]) {
522                                 /* Unescaped blanks end unquoted args. */
523                                 white = 1;
```

```
524                                 break;
525                         }
526                 } else if ('"' == cp[0]) {
527                         if ('"' == cp[1]) {
528                                 /* Quoted quotes collapse. */
529                                 pairs++;
530                                 cp++;
531                         } else {
532                                 /* Unquoted quotes end quoted args. */
533                                 quoted = 2;
534                                 break;
535                         }
536                 }
537         }

539         /* Quoted argument without a closing quote. */
540         if (1 == quoted)
541                 mandoc_msg(MANDOCERR_BADQUOTE, parse, ln, *pos, NULL);

543         /* Null-terminate this argument and move to the next one. */
544         if (pairs)
545                 cp[-pairs] = '\0';
546         if ('\0' != *cp) {
547                 *cp++ = '\0';
548                 while (' ' == *cp)
549                         cp++;
550         }
551         *pos += (int)(cp - start) + (quoted ? 1 : 0);
552         *cpp = cp;

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

557         return(start);
558 }

560 static int
561 a2time(time_t *t, const char *fmt, const char *p)
562 {
563         struct tm        tm;
564         char            *pp;

566         memset(&tm, 0, sizeof(struct tm));

568         pp = NULL;
569 #ifdef  HAVE_STRPTIME
570         pp = strptime(p, fmt, &tm);
571 #endif
572         if (NULL != pp && '\0' == *pp) {
573                 *t = mktime(&tm);
574                 return(1);
575         }

577         return(0);
578 }

580 static char *
581 time2a(time_t t)
582 {
583         struct tm       *tm;
584         char            *buf, *p;
585         size_t           ssz;
586         int              isz;

588         tm = localtime(&t);
```



```
590         /*
591          * Reserve space:
592          * up to 9 characters for the month (September) + blank
593          * up to 2 characters for the day + comma + blank
594          * 4 characters for the year and a terminating '\0'
595          */
596         p = buf = mandoc_malloc(10 + 4 + 4 + 1);

598         if (0 == (ssz = strftime(p, 10 + 1, "%B ", tm)))
599                 goto fail;
600         p += (int)ssz;

602         if (-1 == (isz = snprintf(p, 4 + 1, "%d, ", tm->tm_mday)))
603                 goto fail;
604         p += isz;

606         if (0 == strftime(p, 4 + 1, "%Y", tm))
607                 goto fail;
608         return(buf);

610 fail:
611         free(buf);
612         return(NULL);
613 }

615 char *
616 mandoc_normdate(struct mparse *parse, char *in, int ln, int pos)
617 {
618         char            *out;
619         time_t           t;

621         if (NULL == in || '\0' == *in ||
622             0 == strcmp(in, "$" "Mdocdate$")) {
623                 mandoc_msg(MANDOCERR_NODATE, parse, ln, pos, NULL);
624                 time(&t);
625         }
626         else if (a2time(&t, "%Y-%m-%d", in))
627                 t = 0;
628         else if (!a2time(&t, "$" "Mdocdate: %b %d %Y $", in) &&
629             !a2time(&t, "%b %d, %Y", in)) {
630                 mandoc_msg(MANDOCERR_BADDATE, parse, ln, pos, NULL);
631                 t = 0;
632         }
633         out = t ? time2a(t) : NULL;
634         return(out ? out : mandoc_strdup(in));
635 }

637 int
638 mandoc_eos(const char *p, size_t sz, int enclosed)
639 {
640         const char *q;
641         int found;

643         if (0 == sz)
644                 return(0);

646         /*
647          * End-of-sentence recognition must include situations where
648          * some symbols, such as ')', allow prior EOS punctuation to
649          * propagate outward.
650          */

652         found = 0;
653         for (q = p + (int)sz - 1; q >= p; q--) {
654                 switch (*q) {
655                 case ('\"'):
```

```
656                         /* FALLTHROUGH */
657                 case ('\''):
658                         /* FALLTHROUGH */
659                 case (']'):
660                         /* FALLTHROUGH */
661                 case (')'):
662                         if (0 == found)
663                                 enclosed = 1;
664                         break;
665                 case ('.'):
666                         /* FALLTHROUGH */
667                 case ('!'):
668                         /* FALLTHROUGH */
669                 case ('?'):
670                         found = 1;
671                         break;
672                 default:
673                         return(found && (!enclosed || isalnum((unsigned char)*q)
674                 }
675         }

677         return(found && !enclosed);
678 }

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 }

706 /*
707  * Convert a string to a long that may not be <0.
708  * If the string is invalid, or is less than 0, return -1.
709  */
710 int
711 mandoc_strntoi(const char *p, size_t sz, int base)
712 {
713         char             buf[32];
714         char            *ep;
715         long             v;

717         if (sz > 31)
718                 return(-1);

720         memcpy(buf, p, sz);
721         buf[(int)sz] = '\0';
```



```
723         errno = 0;
724         v = strtol(buf, &ep, base);

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

729         if (v > INT_MAX)
730                 v = INT_MAX;
731         if (v < INT_MIN)
732                 v = INT_MIN;

734         return((int)v);
735 }
```

```
**********************************************************
   13893 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mandoc.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mandoc.h,v 1.99 2012/02/16 20:51:31 joerg 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.
 */
#ifndef MANDOC_H
#define MANDOC_H

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

/*
 * Status level.  This refers to both internal status (i.e., whilst
 * running, when warnings/errors are reported) and an indicator of a
 * threshold of when to halt (when said internal state exceeds the
 * threshold).
 */
enum    mandoclevel {
        MANDOCLEVEL_OK = 0,
        MANDOCLEVEL_RESERVED,
        MANDOCLEVEL_WARNING, /* warnings: syntax, whitespace, etc. */
        MANDOCLEVEL_ERROR, /* input has been thrown away */
        MANDOCLEVEL_FATAL, /* input is borked */
        MANDOCLEVEL_BADARG, /* bad argument in invocation */
        MANDOCLEVEL_SYSERR, /* system error */
        MANDOCLEVEL_MAX
};

/*
 * All possible things that can go wrong within a parse, be it libroff,
 * libmdoc, or libman.
 */
enum    mandocerr {
        MANDOCERR_OK,

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

        /* related to the prologue */
        MANDOCERR_NOTITLE, /* no title in document */
        MANDOCERR_UPPERCASE, /* document title should be all caps */
        MANDOCERR_BADMSEC, /* unknown manual section */
        MANDOCERR_NODATE, /* date missing, using today's date */
        MANDOCERR_BADDATE, /* cannot parse date, using it verbatim */
        MANDOCERR_PROLOGOOO, /* prologue macros out of order */
        MANDOCERR_PROLOGREP, /* duplicate prologue macro */
        MANDOCERR_BADPROLOG, /* macro not allowed in prologue */
        MANDOCERR_BADBODY, /* macro not allowed in body */

        /* related to document structure */
        MANDOCERR_SO, /* .so is fragile, better use ln(1) */
```



```
        MANDOCERR_NAMESECFIRST, /* NAME section must come first */
        MANDOCERR_BADNAMESEC, /* bad NAME section contents */
        MANDOCERR_NONAME, /* manual name not yet set */
        MANDOCERR_SECOOO, /* sections out of conventional order */
        MANDOCERR_SECREP, /* duplicate section name */
        MANDOCERR_SECMSEC, /* section not in conventional manual section */

        /* related to macros and nesting */
        MANDOCERR_MACROOBS, /* skipping obsolete macro */
        MANDOCERR_IGNPAR, /* skipping paragraph macro */
        MANDOCERR_IGNNS, /* skipping no-space macro */
        MANDOCERR_SCOPENEST, /* blocks badly nested */
        MANDOCERR_CHILD, /* child violates parent syntax */
        MANDOCERR_NESTEDDISP, /* nested displays are not portable */
        MANDOCERR_SCOPEREP, /* already in literal mode */
        MANDOCERR_LINESCOPE, /* line scope broken */

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

        /* related to bad macro arguments */
        MANDOCERR_IGNARGV, /* skipping argument */
        MANDOCERR_ARGVREP, /* duplicate argument */
        MANDOCERR_DISPREP, /* duplicate display type */
        MANDOCERR_LISTREP, /* duplicate list type */
        MANDOCERR_BADATT, /* unknown AT&T UNIX version */
        MANDOCERR_BADBOOL, /* bad Boolean value */
        MANDOCERR_BADFONT, /* unknown font */
        MANDOCERR_BADSTANDARD, /* unknown standard specifier */
        MANDOCERR_BADWIDTH, /* bad width argument */

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

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

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

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

        /* related to tables */
        MANDOCERR_TBL, /* bad table syntax */
        MANDOCERR_TBLOPT, /* bad table option */
        MANDOCERR_TBLLAYOUT, /* bad table layout */
        MANDOCERR_TBLNOLAYOUT, /* no table layout cells specified */
        MANDOCERR_TBLNODATA, /* no table data cells specified */
        MANDOCERR_TBLIGNDATA, /* ignore data in cell */
        MANDOCERR_TBLBLOCK, /* data block still open */
        MANDOCERR_TBLEXTRADAT, /* ignoring extra data cells */
```

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

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

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

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

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 };

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



```
194 };

196 enum    tbl_cellt {
197         TBL_CELL_CENTRE, /* c, C */
198         TBL_CELL_RIGHT, /* r, R */
199         TBL_CELL_LEFT, /* l, L */
200         TBL_CELL_NUMBER, /* n, N */
201         TBL_CELL_SPAN, /* s, S */
202         TBL_CELL_LONG, /* a, A */
203         TBL_CELL_DOWN, /* ^ */
204         TBL_CELL_HORIZ, /* _, - */
205         TBL_CELL_DHORIZ, /* = */
206         TBL_CELL_VERT, /* | */
207         TBL_CELL_DVERT, /* || */
208         TBL_CELL_MAX
209 };

211 /*
212  * A cell in a layout row.
213  */
214 struct  tbl_cell {
215         struct tbl_cell  *next;
216         enum tbl_cellt    pos;
217         size_t            spacing;
218         int               flags;
219 #define TBL_CELL_TALIGN  (1 << 0) /* t, T */
220 #define TBL_CELL_BALIGN  (1 << 1) /* d, D */
221 #define TBL_CELL_BOLD    (1 << 2) /* fB, B, b */
222 #define TBL_CELL_ITALIC  (1 << 3) /* fI, I, i */
223 #define TBL_CELL_EQUAL   (1 << 4) /* e, E */
224 #define TBL_CELL_UP      (1 << 5) /* u, U */
225 #define TBL_CELL_WIGN    (1 << 6) /* z, Z */
226         struct tbl_head  *head;
227 };

229 /*
230  * A layout row.
231  */
232 struct  tbl_row {
233         struct tbl_row   *next;
234         struct tbl_cell  *first;
235         struct tbl_cell  *last;
236 };

238 enum    tbl_datt {
239         TBL_DATA_NONE, /* has no data */
240         TBL_DATA_DATA, /* consists of data/string */
241         TBL_DATA_HORIZ, /* horizontal line */
242         TBL_DATA_DHORIZ, /* double-horizontal line */
243         TBL_DATA_NHORIZ, /* squeezed horizontal line */
244         TBL_DATA_NDHORIZ /* squeezed double-horizontal line */
245 };

247 /*
248  * A cell within a row of data.  The "string" field contains the actual
249  * string value that's in the cell.  The rest is layout.
250  */
251 struct  tbl_dat {
252         struct tbl_cell  *layout; /* layout cell */
253         int               spans; /* how many spans follow */
254         struct tbl_dat   *next;
255         char             *string; /* data (NULL if not TBL_DATA_DATA) */
256         enum tbl_datt     pos;
257 };

259 enum    tbl_spant {
```

```
260         TBL_SPAN_DATA, /* span consists of data */
261         TBL_SPAN_HORIZ, /* span is horizontal line */
262         TBL_SPAN_DHORIZ /* span is double horizontal line */
263 };

265 /*
266  * A row of data in a table.
267  */
268 struct  tbl_span {
269         struct tbl       *tbl;
270         struct tbl_head  *head;
271         struct tbl_row   *layout; /* layout row */
272         struct tbl_dat   *first;
273         struct tbl_dat   *last;
274         int               line; /* parse line */
275         int               flags;
276 #define TBL_SPAN_FIRST   (1 << 0)
277 #define TBL_SPAN_LAST    (1 << 1)
278         enum tbl_spant    pos;
279         struct tbl_span  *next;
280 };

282 enum    eqn_boxt {
283         EQN_ROOT, /* root of parse tree */
284         EQN_TEXT, /* text (number, variable, whatever) */
285         EQN_SUBEXPR, /* nested ‘eqn’ subexpression */
286         EQN_LIST, /* subexpressions list */
287         EQN_MATRIX /* matrix subexpression */
288 };

290 enum    eqn_markt {
291         EQNMARK_NONE = 0,
292         EQNMARK_DOT,
293         EQNMARK_DOTDOT,
294         EQNMARK_HAT,
295         EQNMARK_TILDE,
296         EQNMARK_VEC,
297         EQNMARK_DYAD,
298         EQNMARK_BAR,
299         EQNMARK_UNDER,
300         EQNMARK__MAX
301 };

303 enum    eqn_fontt {
304         EQNFONT_NONE = 0,
305         EQNFONT_ROMAN,
306         EQNFONT_BOLD,
307         EQNFONT_FAT,
308         EQNFONT_ITALIC,
309         EQNFONT__MAX
310 };

312 enum    eqn_post {
313         EQNPOS_NONE = 0,
314         EQNPOS_OVER,
315         EQNPOS_SUP,
316         EQNPOS_SUB,
317         EQNPOS_TO,
318         EQNPOS_FROM,
319         EQNPOS__MAX
320 };

322 enum    eqn_pilet {
323         EQNPILE_NONE = 0,
324         EQNPILE_PILE,
325         EQNPILE_CPILE,
```



```
326         EQNPILE_RPILE,
327         EQNPILE_LPILE,
328         EQNPILE_COL,
329         EQNPILE_CCOL,
330         EQNPILE_RCOL,
331         EQNPILE_LCOL,
332         EQNPILE__MAX
333 };

335  /*
336  * A "box" is a parsed mathematical expression as defined by the eqn.7
337  * grammar.
338  */
339 struct  eqn_box {
340         int               size; /* font size of expression */
341 #define EQN_DEFSIZE       INT_MIN
342         enum eqn_boxt     type; /* type of node */
343         struct eqn_box   *first; /* first child node */
344         struct eqn_box   *last; /* last child node */
345         struct eqn_box   *next; /* node sibling */
346         struct eqn_box   *parent; /* node sibling */
347         char             *text; /* text (or NULL) */
348         char             *left;
349         char             *right;
350         enum eqn_post     pos; /* position of next box */
351         enum eqn_markt    mark; /* a mark about the box */
352         enum eqn_fontt    font; /* font of box */
353         enum eqn_pilet    pile; /* equation piling */
354 };

356 /*
357  * An equation consists of a tree of expressions starting at a given
358  * line and position.
359  */
360 struct  eqn {
361         char             *name; /* identifier (or NULL) */
362         struct eqn_box   *root; /* root mathematical expression */
363         int               ln; /* invocation line */
364         int               pos; /* invocation position */
365 };

367 /*
368  * The type of parse sequence.  This value is usually passed via the
369  * mandoc(1) command line of -man and -mdoc.  It’s almost exclusively
370  * -mandoc but the others have been retained for compatibility.
371  */
372 enum    mparset {
373         MPARSE_AUTO, /* magically determine the document type */
374         MPARSE_MDOC, /* assume -mdoc */
375         MPARSE_MAN /* assume -man */
376 };

378 enum    mandoc_esc {
379         ESCAPE_ERROR = 0, /* bail! unparsable escape */
380         ESCAPE_IGNORE, /* escape to be ignored */
381         ESCAPE_SPECIAL, /* a regular special character */
382         ESCAPE_FONT, /* a generic font mode */
383         ESCAPE_FONTBOLD, /* bold font mode */
384         ESCAPE_FONTITALIC, /* italic font mode */
385         ESCAPE_FONTROMAN, /* roman font mode */
386         ESCAPE_FONTPREV, /* previous font mode */
387         ESCAPE_NUMBERED, /* a numbered glyph */
388         ESCAPE_UNICODE, /* a unicode codepoint */
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,
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*/
```

```
**********************************************************
   20580 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mdoc.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc.c,v 1.196 2011/09/30 00:13:28 schwarze Exp $ */
/*
 * 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 <sys/types.h>

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

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

const   char *const __mdoc_macronames[MDOC_MAX] = {
        "Ap",           "Dd",           "Dt",           "Os",
        "Sh",           "Ss",           "Pp",           "D1",
        "Dl",           "Bd",           "Ed",           "Bl",
        "El",           "It",           "Ad",           "An",
        "Ar",           "Cd",           "Cm",           "Dv",
        "Er",           "Ev",           "Ex",           "Fa",
        "Fd",           "Fl",           "Fn",           "Ft",
        "Ic",           "In",           "Li",           "Nd",
        "Nm",           "Op",           "Ot",           "Pa",
        "Rv",           "St",           "Va",           "Vt",
        /* LINTED */
        "Xr",           "%A",           "%B",           "%D",
        /* LINTED */
        "%I",           "%J",           "%N",           "%O",
        /* LINTED */
        "%P",           "%R",           "%T",           "%V",
        "Ac",           "Ao",           "Aq",           "At",
        "Bc",           "Bf",           "Bo",           "Bq",
        "Bsx",          "Bx",           "Db",           "Dc",
        "Do",           "Dq",           "Ec",           "Ef",
        "Em",           "Eo",           "Fx",           "Ms",
        "No",           "Ns",           "Nx",           "Ox",
        "Pc",           "Pf",           "Po",           "Pq",
        "Qc",           "Ql",           "Qo",           "Qq",
        "Re",           "Rs",           "Sc",           "So",
```



```
        "Sq",           "Sm",           "Sx",           "Sy",
        "Tn",           "Ux",           "Xc",           "Xo",
        "Fo",           "Fc",           "Oo",           "Oc",
        "Bk",           "Ek",           "Bt",           "Hf",
        "Fr",           "Ud",           "Lb",           "Lp",
        "Lk",           "Mt",           "Brq",          "Bro",
        /* LINTED */
        "Brc",          "%C",           "Es",           "En",
        /* LINTED */
        "Dx",           "%Q",           "br",           "sp",
        /* LINTED */
        "%U",           "Ta"
        };

const   char *const __mdoc_argnames[MDOC_ARG_MAX] = {
        "split",                "nosplit",              "ragged",
        "unfilled",             "literal",              "file",
        "offset",               "bullet",               "dash",
        "hyphen",               "item",                 "enum",
        "tag",                  "diag",                 "hang",
        "ohang",                "inset",                "column",
        "width",                "compact",              "std",
        "filled",               "words",                "emphasis",
        "symbolic",             "nested",               "centered"
        };

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

static  void              mdoc_node_free(struct mdoc_node *);
static  void              mdoc_node_unlink(struct mdoc *,
                                  struct mdoc_node *);
static  void              mdoc_free1(struct mdoc *);
static  void              mdoc_alloc1(struct mdoc *);
static  struct mdoc_node *node_alloc(struct mdoc *, int, int,
                                  enum mdoct, enum mdoc_type);
static  int               node_append(struct mdoc *,
                                  struct mdoc_node *);
#if 0
static  int               mdoc_preptext(struct mdoc *, int, char *, int);
#endif
static  int               mdoc_ptext(struct mdoc *, int, char *, int);
static  int               mdoc_pmacro(struct mdoc *, int, char *, int);

const struct mdoc_node *
mdoc_node(const struct mdoc *m)
{

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


const struct mdoc_meta *
mdoc_meta(const struct mdoc *m)
{

        assert( ! (MDOC_HALT & m->flags));
        return(&m->meta);
}


/*
 * Frees volatile resources (parse tree, meta-data, fields).
 */
static void
```

```
mdoc_free1(struct mdoc *mdoc)
{

        if (mdoc->first)
                mdoc_node_delete(mdoc, mdoc->first);
        if (mdoc->meta.title)
                free(mdoc->meta.title);
        if (mdoc->meta.os)
                free(mdoc->meta.os);
        if (mdoc->meta.name)
                free(mdoc->meta.name);
        if (mdoc->meta.arch)
                free(mdoc->meta.arch);
        if (mdoc->meta.vol)
                free(mdoc->meta.vol);
        if (mdoc->meta.msec)
                free(mdoc->meta.msec);
        if (mdoc->meta.date)
                free(mdoc->meta.date);
}


/*
 * Allocate all volatile resources (parse tree, meta-data, fields).
 */
static void
mdoc_alloc1(struct mdoc *mdoc)
{

        memset(&mdoc->meta, 0, sizeof(struct mdoc_meta));
        mdoc->flags = 0;
        mdoc->lastnamed = mdoc->lastsec = SEC_NONE;
        mdoc->last = mandoc_calloc(1, sizeof(struct mdoc_node));
        mdoc->first = mdoc->last;
        mdoc->last->type = MDOC_ROOT;
        mdoc->last->tok = MDOC_MAX;
        mdoc->next = MDOC_NEXT_CHILD;
}


/*
 * Free up volatile resources (see mdoc_free1()) then re-initialises the
 * data with mdoc_alloc1().  After invocation, parse data has been reset
 * and the parser is ready for re-invocation on a new tree; however,
 * cross-parse non-volatile data is kept intact.
 */
void
mdoc_reset(struct mdoc *mdoc)
{

        mdoc_free1(mdoc);
        mdoc_alloc1(mdoc);
}


/*
 * Completely free up all volatile and non-volatile parse resources.
 * After invocation, the pointer is no longer usable.
 */
void
mdoc_free(struct mdoc *mdoc)
{

        mdoc_free1(mdoc);
        free(mdoc);
}
```



```


/*
 * Allocate volatile and non-volatile parse resources.
 */
struct mdoc *
mdoc_alloc(struct roff *roff, struct mparse *parse)
{
        struct mdoc     *p;

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

        p->parse = parse;
        p->roff = roff;

        mdoc_hash_init();
        mdoc_alloc1(p);
        return(p);
}


/*
 * Climb back up the parse tree, validating open scopes.  Mostly calls
 * through to macro_end() in macro.c.
 */
int
mdoc_endparse(struct mdoc *m)
{

        assert( ! (MDOC_HALT & m->flags));
        if (mdoc_macroend(m))
                return(1);
        m->flags |= MDOC_HALT;
        return(0);
}

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

        assert( ! (MDOC_HALT & m->flags));

        /* No text before an initial macro. */

        if (SEC_NONE == m->lastnamed) {
                mdoc_pmsg(m, ep->ln, ep->pos, MANDOCERR_NOTEXT);
                return(1);
        }

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

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

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

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

        assert( ! (MDOC_HALT & m->flags));
```

```
261         /* No text before an initial macro. */

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

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

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

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


279 /*
280  * Main parse routine.  Parses a single line -- really just hands off to
281  * the macro (mdoc_pmacro()) or text parser (mdoc_ptext()).
282  */
283 int
284 mdoc_parseln(struct mdoc *m, int ln, char *buf, int offs)
285 {

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

289         m->flags |= MDOC_NEWLINE;

291         /*
292          * Let the roff nS register switch SYNOPSIS mode early,
293          * such that the parser knows at all times
294          * whether this mode is on or off.
295          * Note that this mode is also switched by the Sh macro.
296          */
297         if (roff_regisset(m->roff, REG_nS)) {
298                 if (roff_regget(m->roff, REG_nS))
299                         m->flags |= MDOC_SYNOPSIS;
300                 else
301                         m->flags &= ~MDOC_SYNOPSIS;
302         }

304         return(mandoc_getcontrol(buf, &offs) ?
305                         mdoc_pmacro(m, ln, buf, offs) :
306                         mdoc_ptext(m, ln, buf, offs));
307 }

309 int
310 mdoc_macro(MACRO_PROT_ARGS)
311 {
312         assert(tok < MDOC_MAX);

314         /* If we're in the body, deny prologue calls. */

316         if (MDOC_PROLOGUE & mdoc_macros[tok].flags &&
317                         MDOC_PBODY & m->flags) {
318                 mdoc_pmsg(m, line, ppos, MANDOCERR_BADBODY);
319                 return(1);
320         }

322         /* If we're in the prologue, deny "body" macros.  */

324         if ( ! (MDOC_PROLOGUE & mdoc_macros[tok].flags) &&
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;
339         }

341         return((*mdoc_macros[tok].fp)(m, tok, line, ppos, pos, buf));
342 }


345 static int
346 node_append(struct mdoc *mdoc, struct mdoc_node *p)
347 {

349         assert(mdoc->last);
350         assert(mdoc->first);
351         assert(MDOC_ROOT != p->type);

353         switch (mdoc->next) {
354         case (MDOC_NEXT_SIBLING):
355                 mdoc->last->next = p;
356                 p->prev = mdoc->last;
357                 p->parent = mdoc->last->parent;
358                 break;
359         case (MDOC_NEXT_CHILD):
360                 mdoc->last->child = p;
361                 p->parent = mdoc->last;
362                 break;
363         default:
364                 abort();
365                 /* NOTREACHED */
366         }

368         p->parent->nchild++;

370         /*
371          * Copy over the normalised-data pointer of our parent.  Not
372          * everybody has one, but copying a null pointer is fine.
373          */

375         switch (p->type) {
376         case (MDOC_BODY):
377                 /* FALLTHROUGH */
378         case (MDOC_TAIL):
379                 /* FALLTHROUGH */
380         case (MDOC_HEAD):
381                 p->norm = p->parent->norm;
382                 break;
383         default:
384                 break;
385         }

387         if ( ! mdoc_valid_pre(mdoc, p))
388                 return(0);

390         switch (p->type) {
391         case (MDOC_HEAD):
```

```
392                 assert(MDOC_BLOCK == p->parent->type);
393                 p->parent->head = p;
394                 break;
395         case (MDOC_TAIL):
396                 assert(MDOC_BLOCK == p->parent->type);
397                 p->parent->tail = p;
398                 break;
399         case (MDOC_BODY):
400                 if (p->end)
401                         break;
402                 assert(MDOC_BLOCK == p->parent->type);
403                 p->parent->body = p;
404                 break;
405         default:
406                 break;
407         }

409         mdoc->last = p;

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

422         return(1);
423 }


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

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

439         /* Flag analysis. */

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

449         return(p);
450 }


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



```
458         p = node_alloc(m, line, pos, tok, MDOC_TAIL);
459         if ( ! node_append(m, p))
460                 return(0);
461         m->next = MDOC_NEXT_CHILD;
462         return(1);
463 }


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

471         assert(m->first);
472         assert(m->last);

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


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

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


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

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


511 int
512 mdoc_block_alloc(struct mdoc *m, int line, int pos,
513                 enum mdoct tok, struct mdoc_arg *args)
514 {
515         struct mdoc_node *p;

517         p = node_alloc(m, line, pos, tok, MDOC_BLOCK);
518         p->args = args;
519         if (p->args)
520                 (args->refcnt)++;

522         switch (tok) {
523         case (MDOC_Bd):
```

```
524                 /* FALLTHROUGH */
525         case (MDOC_Bf):
526                 /* FALLTHROUGH */
527         case (MDOC_Bl):
528                 /* FALLTHROUGH */
529         case (MDOC_Rs):
530                 p->norm = mandoc_calloc(1, sizeof(union mdoc_data));
531                 break;
532         default:
533                 break;
534         }

536         if ( ! node_append(m, p))
537                 return(0);
538         m->next = MDOC_NEXT_CHILD;
539         return(1);
540 }


543 int
544 mdoc_elem_alloc(struct mdoc *m, int line, int pos,
545                 enum mdoct tok, struct mdoc_arg *args)
546 {
547         struct mdoc_node *p;

549         p = node_alloc(m, line, pos, tok, MDOC_ELEM);
550         p->args = args;
551         if (p->args)
552                 (args->refcnt)++;

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

562         if ( ! node_append(m, p))
563                 return(0);
564         m->next = MDOC_NEXT_CHILD;
565         return(1);
566 }

568 int
569 mdoc_word_alloc(struct mdoc *m, int line, int pos, const char *p)
570 {
571         struct mdoc_node *n;

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

576         if ( ! node_append(m, n))
577                 return(0);

579         m->next = MDOC_NEXT_SIBLING;
580         return(1);
581 }


584 static void
585 mdoc_node_free(struct mdoc_node *p)
586 {

588         if (MDOC_BLOCK == p->type || MDOC_ELEM == p->type)
589                 free(p->norm);
```



```
590         if (p->string)
591                 free(p->string);
592         if (p->args)
593                 mdoc_argv_free(p->args);
594         free(p);
595 }


598 static void
599 mdoc_node_unlink(struct mdoc *m, struct mdoc_node *n)
600 {

602         /* Adjust siblings. */

604         if (n->prev)
605                 n->prev->next = n->next;
606         if (n->next)
607                 n->next->prev = n->prev;

609         /* Adjust parent. */

611         if (n->parent) {
612                 n->parent->nchild--;
613                 if (n->parent->child == n)
614                         n->parent->child = n->prev ? n->prev : n->next;
615                 if (n->parent->last == n)
616                         n->parent->last = n->prev ? n->prev : NULL;
617         }

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

621         if (m && m->last == n) {
622                 if (n->prev) {
623                         m->last = n->prev;
624                         m->next = MDOC_NEXT_SIBLING;
625                 } else {
626                         m->last = n->parent;
627                         m->next = MDOC_NEXT_CHILD;
628                 }
629         }

631         if (m && m->first == n)
632                 m->first = NULL;
633 }


636 void
637 mdoc_node_delete(struct mdoc *m, struct mdoc_node *p)
638 {

640         while (p->child) {
641                 assert(p->nchild);
642                 mdoc_node_delete(m, p->child);
643         }
644         assert(0 == p->nchild);

646         mdoc_node_unlink(m, p);
647         mdoc_node_free(p);
648 }

650 #if 0
651 /*
652  * Pre-treat a text line.
653  * Text lines can consist of equations, which must be handled apart from
654  * the regular text.
655  * Thus, use this function to step through a line checking if it has any
```

```
656  * equations embedded in it.
657  * This must handle multiple equations AND equations that do not end at
658  * the end-of-line, i.e., will re-enter in the next roff parse.
659  */
660 static int
661 mdoc_preptext(struct mdoc *m, int line, char *buf, int offs)
662 {
663         char            *start, *end;
664         char             delim;

666         while ('\0' != buf[offs]) {
667                 /* Mark starting position if eqn is set. */
668                 start = NULL;
669                 if ('\0' != (delim = roff_eqndelim(m->roff)))
670                         if (NULL != (start = strchr(buf + offs, delim)))
671                                 *start++ = '\0';

673                 /* Parse text as normal. */
674                 if ( ! mdoc_ptext(m, line, buf, offs))
675                         return(0);

677                 /* Continue only if an equation exists. */
678                 if (NULL == start)
679                         break;

681                 /* Read past the end of the equation. */
682                 offs += start - (buf + offs);
683                 assert(start == &buf[offs]);
684                 if (NULL != (end = strchr(buf + offs, delim))) {
685                         *end++ = '\0';
686                         while (' ' == *end)
687                                 end++;
688                 }

690                 /* Parse the equation itself. */
691                 roff_openeqn(m->roff, NULL, line, offs, buf);

693                 /* Process a finished equation? */
694                 if (roff_closeeqn(m->roff))
695                         if ( ! mdoc_addeqn(m, roff_eqn(m->roff)))
696                                 return(0);
697                 offs += (end - (buf + offs));
698         }

700         return(1);
701 }
702 #endif

704 /*
705  * Parse free-form text, that is, a line that does not begin with the
706  * control character.
707  */
708 static int
709 mdoc_ptext(struct mdoc *m, int line, char *buf, int offs)
710 {
711         char             *c, *ws, *end;
712         struct mdoc_node *n;

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

716         if (SEC_NONE == m->lastnamed) {
717                 mdoc_pmsg(m, line, offs, MANDOCERR_NOTEXT);
718                 return(1);
719         }

721         assert(m->last);
```



```
722         n = m->last;

724         /*
725          * Divert directly to list processing if we're encountering a
726          * columnar MDOC_BLOCK with or without a prior MDOC_BLOCK entry
727          * (a MDOC_BODY means it's already open, in which case we should
728          * process within its context in the normal way).
729          */

731         if (MDOC_Bl == n->tok && MDOC_BODY == n->type &&
732                         LIST_column == n->norm->Bl.type) {
733                 /* 'Bl' is open without any children. */
734                 m->flags |= MDOC_FREECOL;
735                 return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
736         }

738         if (MDOC_It == n->tok && MDOC_BLOCK == n->type &&
739                         NULL != n->parent &&
740                         MDOC_Bl == n->parent->tok &&
741                         LIST_column == n->parent->norm->Bl.type) {
742                 /* 'Bl' has block-level 'It' children. */
743                 m->flags |= MDOC_FREECOL;
744                 return(mdoc_macro(m, MDOC_It, line, offs, &offs, buf));
745         }

747         /*
748          * Search for the beginning of unescaped trailing whitespace (ws)
749          * and for the first character not to be output (end).
750          */

752         /* FIXME: replace with strcspn(). */
753         ws = NULL;
754         for (c = end = buf + offs; *c; c++) {
755                 switch (*c) {
756                 case ' ':
757                         if (NULL == ws)
758                                 ws = c;
759                         continue;
760                 case '\t':
761                         /*
762                          * Always warn about trailing tabs,
763                          * even outside literal context,
764                          * where they should be put on the next line.
765                          */
766                         if (NULL == ws)
767                                 ws = c;
768                         /*
769                          * Strip trailing tabs in literal context only;
770                          * outside, they affect the next line.
771                          */
772                         if (MDOC_LITERAL & m->flags)
773                                 continue;
774                         break;
775                 case '\\':
776                         /* Skip the escaped character, too, if any. */
777                         if (c[1])
778                                 c++;
779                         /* FALLTHROUGH */
780                 default:
781                         ws = NULL;
782                         break;
783                 }
784                 end = c + 1;
785         }
786         *end = '\0';
```

```
788         if (ws)
789                 mdoc_pmsg(m, line, (int)(ws-buf), MANDOCERR_EOLNSPACE);
790
791         if ('\0' == buf[offs] && ! (MDOC_LITERAL & m->flags)) {
792                 mdoc_pmsg(m, line, (int)(c-buf), MANDOCERR_NOBLANKLN);
793
794                 /*
795                  * Insert a 'sp' in the case of a blank line.  Technically,
796                  * blank lines aren't allowed, but enough manuals assume this
797                  * behaviour that we want to work around it.
798                  */
799                 if ( ! mdoc_elem_alloc(m, line, offs, MDOC_sp, NULL))
800                         return(0);
801
802                 m->next = MDOC_NEXT_SIBLING;
803                 return(1);
804         }
805
806         if ( ! mdoc_word_alloc(m, line, offs, buf+offs))
807                 return(0);
808
809         if (MDOC_LITERAL & m->flags)
810                 return(1);
811
812         /*
813          * End-of-sentence check.  If the last character is an unescaped
814          * EOS character, then flag the node as being the end of a
815          * sentence.  The front-end will know how to interpret this.
816          */
817
818         assert(buf < end);
819
820         if (mandoc_eos(buf+offs, (size_t)(end-buf-offs), 0))
821                 m->last->flags |= MDOC_EOS;
822
823         return(1);
824 }
825
826
827 /*
828  * Parse a macro line, that is, a line beginning with the control
829  * character.
830  */
831 static int
832 mdoc_pmacro(struct mdoc *m, int ln, char *buf, int offs)
833 {
834         enum mdoct        tok;
835         int               i, sv;
836         char              mac[5];
837         struct mdoc_node *n;
838
839         /* Empty post-control lines are ignored. */
840
841         if ('"' == buf[offs]) {
842                 mdoc_pmsg(m, ln, offs, MANDOCERR_BADCOMMENT);
843                 return(1);
844         } else if ('\0' == buf[offs])
845                 return(1);
846
847         sv = offs;
848
849         /*
850          * Copy the first word into a nil-terminated buffer.
851          * Stop copying when a tab, space, or eoln is encountered.
852          */
```



```
853
854         i = 0;
855         while (i < 4 && '\0' != buf[offs] &&
856                         ' ' != buf[offs] && '\t' != buf[offs])
857                 mac[i++] = buf[offs++];
858
859         mac[i] = '\0';
860
861         tok = (i > 1 || i < 4) ? mdoc_hash_find(mac) : MDOC_MAX;
862
863         if (MDOC_MAX == tok) {
864                 mandoc_vmsg(MANDOCERR_MACRO, m->parse,
865                                 ln, sv, "%s", buf + sv - 1);
866                 return(1);
867         }
868
869         /* Disregard the first trailing tab, if applicable. */
870
871         if ('\t' == buf[offs])
872                 offs++;
873
874         /* Jump to the next non-whitespace word. */
875
876         while (buf[offs] && ' ' == buf[offs])
877                 offs++;
878
879         /*
880          * Trailing whitespace.  Note that tabs are allowed to be passed
881          * into the parser as "text", so we only warn about spaces here.
882          */
883
884         if ('\0' == buf[offs] && ' ' == buf[offs - 1])
885                 mdoc_pmsg(m, ln, offs - 1, MANDOCERR_EOLNSPACE);
886
887         /*
888          * If an initial macro or a list invocation, divert directly
889          * into macro processing.
890          */
891
892         if (NULL == m->last || MDOC_It == tok || MDOC_El == tok) {
893                 if ( ! mdoc_macro(m, tok, ln, sv, &offs, buf))
894                         goto err;
895                 return(1);
896         }
897
898         n = m->last;
899         assert(m->last);
900
901         /*
902          * If the first macro of a 'Bl -column', open an 'It' block
903          * context around the parsed macro.
904          */
905
906         if (MDOC_Bl == n->tok && MDOC_BODY == n->type &&
907                         LIST_column == n->norm->Bl.type) {
908                 m->flags |= MDOC_FREECOL;
909                 if ( ! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
910                         goto err;
911                 return(1);
912         }
913
914         /*
915          * If we're following a block-level 'It' within a 'Bl -column'
916          * context (perhaps opened in the above block or in ptext()),
917          * then open an 'It' block context around the parsed macro.
918          */
```

```
920         if (MDOC_It == n->tok && MDOC_BLOCK == n->type &&
921                         NULL != n->parent &&
922                         MDOC_Bl == n->parent->tok &&
923                         LIST_column == n->parent->norm->Bl.type) {
924                 m->flags |= MDOC_FREECOL;
925                 if ( ! mdoc_macro(m, MDOC_It, ln, sv, &sv, buf))
926                         goto err;
927                 return(1);
928         }

930         /* Normal processing of a macro. */

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

935         return(1);

937 err:    /* Error out. */

939         m->flags |= MDOC_HALT;
940         return(0);
941 }

943 enum mdelim
944 mdoc_isdelim(const char *p)
945 {

947         if ('\0' == p[0])
948                 return(DELIM_NONE);

950         if ('\0' == p[1])
951                 switch (p[0]) {
952                 case('('):
953                         /* FALLTHROUGH */
954                 case('['):
955                         return(DELIM_OPEN);
956                 case('|'):
957                         return(DELIM_MIDDLE);
958                 case('.'):
959                         /* FALLTHROUGH */
960                 case(','):
961                         /* FALLTHROUGH */
962                 case(';'):
963                         /* FALLTHROUGH */
964                 case(':'):
965                         /* FALLTHROUGH */
966                 case('?'):
967                         /* FALLTHROUGH */
968                 case('!'):
969                         /* FALLTHROUGH */
970                 case(')'):
971                         /* FALLTHROUGH */
972                 case(']'):
973                         return(DELIM_CLOSE);
974                 default:
975                         return(DELIM_NONE);
976                 }

978         if ('\\' != p[0])
979                 return(DELIM_NONE);

981         if (0 == strcmp(p + 1, "."))
982                 return(DELIM_CLOSE);
983         if (0 == strcmp(p + 1, "*(Ba"))
984                 return(DELIM_MIDDLE);
```



```
986         return(DELIM_NONE);
987 }
```

```
**********************************************************
    8425 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mdoc.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc.h,v 1.122 2011/03/22 14:05:45 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.
 */
#ifndef MDOC_H
#define MDOC_H

enum    mdoct {
        MDOC_Ap = 0,
        MDOC_Dd,
        MDOC_Dt,
        MDOC_Os,
        MDOC_Sh,
        MDOC_Ss,
        MDOC_Pp,
        MDOC_D1,
        MDOC_Dl,
        MDOC_Bd,
        MDOC_Ed,
        MDOC_Bl,
        MDOC_El,
        MDOC_It,
        MDOC_Ad,
        MDOC_An,
        MDOC_Ar,
        MDOC_Cd,
        MDOC_Cm,
        MDOC_Dv,
        MDOC_Er,
        MDOC_Ev,
        MDOC_Ex,
        MDOC_Fa,
        MDOC_Fd,
        MDOC_Fl,
        MDOC_Fn,
        MDOC_Ft,
        MDOC_Ic,
        MDOC_In,
        MDOC_Li,
        MDOC_Nd,
        MDOC_Nm,
        MDOC_Op,
        MDOC_Ot,
        MDOC_Pa,
        MDOC_Rv,
        MDOC_St,
        MDOC_Va,
        MDOC_Vt,
        MDOC_Xr,
```



```
        MDOC__A,
        MDOC__B,
        MDOC__D,
        MDOC__I,
        MDOC__J,
        MDOC__N,
        MDOC__O,
        MDOC__P,
        MDOC__R,
        MDOC__T,
        MDOC__V,
        MDOC_Ac,
        MDOC_Ao,
        MDOC_Aq,
        MDOC_At,
        MDOC_Bc,
        MDOC_Bf,
        MDOC_Bo,
        MDOC_Bq,
        MDOC_Bsx,
        MDOC_Bx,
        MDOC_Db,
        MDOC_Dc,
        MDOC_Do,
        MDOC_Dq,
        MDOC_Ec,
        MDOC_Ef,
        MDOC_Em,
        MDOC_Eo,
        MDOC_Fx,
        MDOC_Ms,
        MDOC_No,
        MDOC_Ns,
        MDOC_Nx,
        MDOC_Ox,
        MDOC_Pc,
        MDOC_Pf,
        MDOC_Po,
        MDOC_Pq,
        MDOC_Qc,
        MDOC_Ql,
        MDOC_Qo,
        MDOC_Qq,
        MDOC_Re,
        MDOC_Rs,
        MDOC_Sc,
        MDOC_So,
        MDOC_Sq,
        MDOC_Sm,
        MDOC_Sx,
        MDOC_Sy,
        MDOC_Tn,
        MDOC_Ux,
        MDOC_Xc,
        MDOC_Xo,
        MDOC_Fo,
        MDOC_Fc,
        MDOC_Oo,
        MDOC_Oc,
        MDOC_Bk,
        MDOC_Ek,
        MDOC_Bt,
        MDOC_Hf,
        MDOC_Fr,
        MDOC_Ud,
        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 };

146 enum    mdocargt {
147         MDOC_Split, /* -split */
148         MDOC_Nosplit, /* -nospli */
149         MDOC_Ragged, /* -ragged */
150         MDOC_Unfilled, /* -unfilled */
151         MDOC_Literal, /* -literal */
152         MDOC_File, /* -file */
153         MDOC_Offset, /* -offset */
154         MDOC_Bullet, /* -bullet */
155         MDOC_Dash, /* -dash */
156         MDOC_Hyphen, /* -hyphen */
157         MDOC_Item, /* -item */
158         MDOC_Enum, /* -enum */
159         MDOC_Tag, /* -tag */
160         MDOC_Diag, /* -diag */
161         MDOC_Hang, /* -hang */
162         MDOC_Ohang, /* -ohang */
163         MDOC_Inset, /* -inset */
164         MDOC_Column, /* -column */
165         MDOC_Width, /* -width */
166         MDOC_Compact, /* -compact */
167         MDOC_Std, /* -std */
168         MDOC_Filled, /* -filled */
169         MDOC_Words, /* -words */
170         MDOC_Emphasis, /* -emphasis */
171         MDOC_Symbolic, /* -symbolic */
172         MDOC_Nested, /* -nested */
173         MDOC_Centred, /* -centered */
174         MDOC_ARG_MAX
175 };

177 enum    mdoc_type {
178         MDOC_TEXT,
179         MDOC_ELEM,
180         MDOC_HEAD,
181         MDOC_TAIL,
182         MDOC_BODY,
183         MDOC_BLOCK,
184         MDOC_TBL,
185         MDOC_EQN,
186         MDOC_ROOT
187 };

189 /*
190  * Section (named/unnamed) of 'Sh'.   Note that these appear in the
191  * conventional order imposed by mdoc.7.  In the case of SEC_NONE, no
192  * section has been invoked (this shouldn't happen).  SEC_CUSTOM refers
193  * to other sections.
```



```
194  */
195 enum    mdoc_sec {
196         SEC_NONE = 0,
197         SEC_NAME, /* NAME */
198         SEC_LIBRARY, /* LIBRARY */
199         SEC_SYNOPSIS, /* SYNOPSIS */
200         SEC_DESCRIPTION, /* DESCRIPTION */
201         SEC_IMPLEMENTATION, /* IMPLEMENTATION NOTES */
202         SEC_RETURN_VALUES, /* RETURN VALUES */
203         SEC_ENVIRONMENT,  /* ENVIRONMENT */
204         SEC_FILES, /* FILES */
205         SEC_EXIT_STATUS, /* EXIT STATUS */
206         SEC_EXAMPLES, /* EXAMPLES */
207         SEC_DIAGNOSTICS, /* DIAGNOSTICS */
208         SEC_COMPATIBILITY, /* COMPATIBILITY */
209         SEC_ERRORS, /* ERRORS */
210         SEC_SEE_ALSO, /* SEE ALSO */
211         SEC_STANDARDS, /* STANDARDS */
212         SEC_HISTORY, /* HISTORY */
213         SEC_AUTHORS, /* AUTHORS */
214         SEC_CAVEATS, /* CAVEATS */
215         SEC_BUGS, /* BUGS */
216         SEC_SECURITY, /* SECURITY */
217         SEC_CUSTOM,
218         SEC__MAX
219 };

221 struct  mdoc_meta {
222         char             *msec; /* 'Dt' section (1, 3p, etc.) */
223         char             *vol; /* 'Dt' volume (implied) */
224         char             *arch; /* 'Dt' arch (i386, etc.) */
225         char             *date; /* 'Dd' normalised date */
226         char             *title; /* 'Dt' title (FOO, etc.) */
227         char             *os; /* 'Os' system (OpenBSD, etc.) */
228         char             *name; /* leading 'Nm' name */
229 };

231 /*
232  * An argument to a macro (multiple values = '-column xxx yyy').
233  */
234 struct  mdoc_argv {
235         enum mdocargt     arg; /* type of argument */
236         int               line;
237         int               pos;
238         size_t            sz; /* elements in "value" */
239         char            **value; /* argument strings */
240 };

242 /*
243  * Reference-counted macro arguments.  These are refcounted because
244  * blocks have multiple instances of the same arguments spread across
245  * the HEAD, BODY, TAIL, and BLOCK node types.
246  */
247 struct  mdoc_arg {
248         size_t            argc;
249         struct mdoc_argv *argv;
250         unsigned int      refcnt;
251 };

253 /*
254  * Indicates that a BODY's formatting has ended, but the scope is still
255  * open.  Used for syntax-broken blocks.
256  */
257 enum    mdoc_endbody {
258         ENDBODY_NOT = 0,
259         ENDBODY_SPACE, /* is broken: append a space */
```

```
260         ENDBODY_NOSPACE /* is broken: don't append a space */
261 };

263 enum    mdoc_list {
264         LIST__NONE = 0,
265         LIST_bullet, /* -bullet */
266         LIST_column, /* -column */
267         LIST_dash, /* -dash */
268         LIST_diag, /* -diag */
269         LIST_enum, /* -enum */
270         LIST_hang, /* -hang */
271         LIST_hyphen, /* -hyphen */
272         LIST_inset, /* -inset */
273         LIST_item, /* -item */
274         LIST_ohang, /* -ohang */
275         LIST_tag, /* -tag */
276         LIST_MAX
277 };

279 enum    mdoc_disp {
280         DISP__NONE = 0,
281         DISP_centred, /* -centered */
282         DISP_ragged, /* -ragged */
283         DISP_unfilled, /* -unfilled */
284         DISP_filled, /* -filled */
285         DISP_literal /* -literal */
286 };

288 enum    mdoc_auth {
289         AUTH__NONE = 0,
290         AUTH_split, /* -split */
291         AUTH_nosplit /* -nosplit */
292 };

294 enum    mdoc_font {
295         FONT__NONE = 0,
296         FONT_Em, /* Em, -emphasis */
297         FONT_Li, /* Li, -literal */
298         FONT_Sy /* Sy, -symbolic */
299 };

301 struct  mdoc_bd {
302         const char       *offs; /* -offset */
303         enum mdoc_disp    type; /* -ragged, etc. */
304         int               comp; /* -compact */
305 };

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 };

316 struct  mdoc_bf {
317         enum mdoc_font    font; /* font */
318 };

320 struct  mdoc_an {
321         enum mdoc_auth    auth; /* -split, etc. */
322 };

324 struct  mdoc_rs {
325         int               quote_T; /* whether to quote %T */
```



```
326 };

328 /*
329  * Consists of normalised node arguments.  These should be used instead
330  * of iterating through the mdoc_arg pointers of a node: defaults are
331  * provided, etc.
332  */
333 union   mdoc_data {
334         struct mdoc_an    An;
335         struct mdoc_bd    Bd;
336         struct mdoc_bf    Bf;
337         struct mdoc_bl    Bl;
338         struct mdoc_rs    Rs;
339 };

341 /*
342  * Single node in tree-linked AST.
343  */
344 struct  mdoc_node {
345         struct mdoc_node *parent; /* parent AST node */
346         struct mdoc_node *child; /* first child AST node */
347         struct mdoc_node *last; /* last child AST node */
348         struct mdoc_node *next; /* sibling AST node */
349         struct mdoc_node *prev; /* prior sibling AST node */
350         int               nchild; /* number children */
351         int               line; /* parse line */
352         int               pos; /* parse column */
353         enum mdoct        tok; /* tok or MDOC__MAX if none */
354         int               flags;
355 #define MDOC_VALID       (1 << 0) /* has been validated */
356 #define MDOC_EOS         (1 << 2) /* at sentence boundary */
357 #define MDOC_LINE        (1 << 3) /* first macro/text on line */
358 #define MDOC_SYNPRETTY   (1 << 4) /* SYNOPSIS-style formatting */
359 #define MDOC_ENDED       (1 << 5) /* rendering has been ended */
360 #define MDOC_DELIMO      (1 << 6)
361 #define MDOC_DELIMC      (1 << 7)
362         enum mdoc_type    type; /* AST node type */
363         enum mdoc_sec     sec; /* current named section */
364         union mdoc_data  *norm; /* normalised args */
365         /* FIXME: these can be union'd to shave a few bytes. */
366         struct mdoc_arg  *args; /* BLOCK/ELEM */
367         struct mdoc_node *pending; /* BLOCK */
368         struct mdoc_node *head; /* BLOCK */
369         struct mdoc_node *body; /* BLOCK */
370         struct mdoc_node *tail; /* BLOCK */
371         char             *string; /* TEXT */
372         const struct tbl_span *span; /* TBL */
373         const struct eqn *eqn; /* EQN */
374         enum mdoc_endbody end; /* BODY */
375 };

377 /* Names of macros.  Index is enum mdoct. */
378 extern  const char *const *mdoc_macronames;

380 /* Names of macro args.  Index is enum mdocargt. */
381 extern  const char *const *mdoc_argnames;

383 __BEGIN_DECLS

385 struct  mdoc;

387 const struct mdoc_node *mdoc_node(const struct mdoc *);
388 const struct mdoc_meta *mdoc_meta(const struct mdoc *);

390 __END_DECLS
```

```
392 #endif /*!MDOC_H*/
```

```
**********************************************************
   17101 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mdoc_argv.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_argv.c,v 1.82 2012/03/23 05:50:24 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 <stdlib.h>
#include <stdio.h>
#include <string.h>

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

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

enum    argsflag {
        ARGSFL_NONE = 0,
        ARGSFL_DELIM, /* handle delimiters of [[::delim::][ ]+]+ */
        ARGSFL_TABSEP /* handle tab/‘Ta’ separated phrases */
};

enum    argvflag {
        ARGV_NONE, /* no args to flag (e.g., -split) */
        ARGV_SINGLE, /* one arg to flag (e.g., -file xxx)  */
        ARGV_MULTI, /* multiple args (e.g., -column xxx yyy) */
        ARGV_OPT_SINGLE /* optional arg (e.g., -offset [xxx]) */
};

struct  mdocarg {
        enum argsflag    flags;
        const enum mdocargt *argvs;
};

static  void             argn_free(struct mdoc_arg *, int);
static  enum margserr    args(struct mdoc *, int, int *,
                              char *, enum argsflag, char **);
static  int              args_checkpunct(const char *, int);
static  int              argv_multi(struct mdoc *, int,
                              struct mdoc_argv *, int *, char *);
static  int              argv_opt_single(struct mdoc *, int,
                              struct mdoc_argv *, int *, char *);
```



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

static  const enum argvflag argvflags[MDOC_ARG_MAX] = {
        ARGV_NONE,      /* MDOC_Split */
        ARGV_NONE,      /* MDOC_Nosplit */
        ARGV_NONE,      /* MDOC_Ragged */
        ARGV_NONE,      /* MDOC_Unfilled */
        ARGV_NONE,      /* MDOC_Literal */
        ARGV_SINGLE,    /* MDOC_File */
        ARGV_OPT_SINGLE, /* MDOC_Offset */
        ARGV_NONE,      /* MDOC_Bullet */
        ARGV_NONE,      /* MDOC_Dash */
        ARGV_NONE,      /* MDOC_Hyphen */
        ARGV_NONE,      /* MDOC_Item */
        ARGV_NONE,      /* MDOC_Enum */
        ARGV_NONE,      /* MDOC_Tag */
        ARGV_NONE,      /* MDOC_Diag */
        ARGV_NONE,      /* MDOC_Hang */
        ARGV_NONE,      /* MDOC_Ohang */
        ARGV_NONE,      /* MDOC_Inset */
        ARGV_MULTI,     /* MDOC_Column */
        ARGV_OPT_SINGLE, /* MDOC_Width */
        ARGV_NONE,      /* MDOC_Compact */
        ARGV_NONE,      /* MDOC_Std */
        ARGV_NONE,      /* MDOC_Filled */
        ARGV_NONE,      /* MDOC_Words */
        ARGV_NONE,      /* MDOC_Emphasis */
        ARGV_NONE,      /* MDOC_Symbolic */
        ARGV_NONE       /* MDOC_Symbolic */
};

static  const enum mdocargt args_Ex[] = {
        MDOC_Std,
        MDOC_ARG_MAX
};

static  const enum mdocargt args_An[] = {
        MDOC_Split,
        MDOC_Nosplit,
        MDOC_ARG_MAX
};

static  const enum mdocargt args_Bd[] = {
        MDOC_Ragged,
        MDOC_Unfilled,
        MDOC_Filled,
        MDOC_Literal,
        MDOC_File,
        MDOC_Offset,
        MDOC_Compact,
        MDOC_Centred,
        MDOC_ARG_MAX
};

static  const enum mdocargt args_Bf[] = {
        MDOC_Emphasis,
        MDOC_Literal,
        MDOC_Symbolic,
        MDOC_ARG_MAX
};

static  const enum mdocargt args_Bk[] = {
        MDOC_Words,
        MDOC_ARG_MAX
};
```

```
129 static  const enum mdocargt args_Bl[] = {
130         MDOC_Bullet,
131         MDOC_Dash,
132         MDOC_Hyphen,
133         MDOC_Item,
134         MDOC_Enum,
135         MDOC_Tag,
136         MDOC_Diag,
137         MDOC_Hang,
138         MDOC_Ohang,
139         MDOC_Inset,
140         MDOC_Column,
141         MDOC_Width,
142         MDOC_Offset,
143         MDOC_Compact,
144         MDOC_Nested,
145         MDOC_ARG_MAX
146 };

148 static  const struct mdocarg mdocargs[MDOC_MAX] = {
149         { ARGSFL_NONE, NULL }, /* Ap */
150         { ARGSFL_NONE, NULL }, /* Dd */
151         { ARGSFL_NONE, NULL }, /* Dt */
152         { ARGSFL_NONE, NULL }, /* Os */
153         { ARGSFL_NONE, NULL }, /* Sh */
154         { ARGSFL_NONE, NULL }, /* Ss */
155         { ARGSFL_NONE, NULL }, /* Pp */
156         { ARGSFL_DELIM, NULL }, /* D1 */
157         { ARGSFL_DELIM, NULL }, /* Dl */
158         { ARGSFL_NONE, args_Bd }, /* Bd */
159         { ARGSFL_NONE, NULL }, /* Ed */
160         { ARGSFL_NONE, args_Bl }, /* Bl */
161         { ARGSFL_NONE, NULL }, /* El */
162         { ARGSFL_NONE, NULL }, /* It */
163         { ARGSFL_DELIM, NULL }, /* Ad */
164         { ARGSFL_DELIM, args_An }, /* An */
165         { ARGSFL_DELIM, NULL }, /* Ar */
166         { ARGSFL_NONE, NULL }, /* Cd */
167         { ARGSFL_DELIM, NULL }, /* Cm */
168         { ARGSFL_DELIM, NULL }, /* Dv */
169         { ARGSFL_DELIM, NULL }, /* Er */
170         { ARGSFL_DELIM, NULL }, /* Ev */
171         { ARGSFL_NONE, args_Ex }, /* Ex */
172         { ARGSFL_DELIM, NULL }, /* Fa */
173         { ARGSFL_NONE, NULL }, /* Fd */
174         { ARGSFL_DELIM, NULL }, /* Fl */
175         { ARGSFL_DELIM, NULL }, /* Fn */
176         { ARGSFL_DELIM, NULL }, /* Ft */
177         { ARGSFL_DELIM, NULL }, /* Ic */
178         { ARGSFL_NONE, NULL }, /* In */
179         { ARGSFL_DELIM, NULL }, /* Li */
180         { ARGSFL_NONE, NULL }, /* Nd */
181         { ARGSFL_DELIM, NULL }, /* Nm */
182         { ARGSFL_DELIM, NULL }, /* Op */
183         { ARGSFL_NONE, NULL }, /* Ot */
184         { ARGSFL_DELIM, NULL }, /* Pa */
185         { ARGSFL_NONE, args_Ex }, /* Rv */
186         { ARGSFL_DELIM, NULL }, /* St */
187         { ARGSFL_DELIM, NULL }, /* Va */
188         { ARGSFL_DELIM, NULL }, /* Vt */
189         { ARGSFL_DELIM, NULL }, /* Xr */
190         { ARGSFL_NONE, NULL }, /* %A */
191         { ARGSFL_NONE, NULL }, /* %B */
192         { ARGSFL_NONE, NULL }, /* %D */
193         { ARGSFL_NONE, NULL }, /* %I */
```



```
194         { ARGSFL_NONE, NULL }, /* %J */
195         { ARGSFL_NONE, NULL }, /* %N */
196         { ARGSFL_NONE, NULL }, /* %O */
197         { ARGSFL_NONE, NULL }, /* %P */
198         { ARGSFL_NONE, NULL }, /* %R */
199         { ARGSFL_NONE, NULL }, /* %T */
200         { ARGSFL_NONE, NULL }, /* %V */
201         { ARGSFL_DELIM, NULL }, /* Ac */
202         { ARGSFL_NONE, NULL }, /* Ao */
203         { ARGSFL_DELIM, NULL }, /* Aq */
204         { ARGSFL_DELIM, NULL }, /* At */
205         { ARGSFL_DELIM, NULL }, /* Bc */
206         { ARGSFL_NONE, args_Bf }, /* Bf */
207         { ARGSFL_NONE, NULL }, /* Bo */
208         { ARGSFL_DELIM, NULL }, /* Bq */
209         { ARGSFL_DELIM, NULL }, /* Bsx */
210         { ARGSFL_DELIM, NULL }, /* Bx */
211         { ARGSFL_NONE, NULL }, /* Db */
212         { ARGSFL_DELIM, NULL }, /* Dc */
213         { ARGSFL_NONE, NULL }, /* Do */
214         { ARGSFL_DELIM, NULL }, /* Dq */
215         { ARGSFL_DELIM, NULL }, /* Ec */
216         { ARGSFL_NONE, NULL }, /* Ef */
217         { ARGSFL_DELIM, NULL }, /* Em */
218         { ARGSFL_NONE, NULL }, /* Eo */
219         { ARGSFL_DELIM, NULL }, /* Fx */
220         { ARGSFL_DELIM, NULL }, /* Ms */
221         { ARGSFL_DELIM, NULL }, /* No */
222         { ARGSFL_DELIM, NULL }, /* Ns */
223         { ARGSFL_DELIM, NULL }, /* Nx */
224         { ARGSFL_DELIM, NULL }, /* Ox */
225         { ARGSFL_DELIM, NULL }, /* Pc */
226         { ARGSFL_DELIM, NULL }, /* Pf */
227         { ARGSFL_NONE, NULL }, /* Po */
228         { ARGSFL_DELIM, NULL }, /* Pq */
229         { ARGSFL_DELIM, NULL }, /* Qc */
230         { ARGSFL_DELIM, NULL }, /* Ql */
231         { ARGSFL_NONE, NULL }, /* Qo */
232         { ARGSFL_DELIM, NULL }, /* Qq */
233         { ARGSFL_NONE, NULL }, /* Re */
234         { ARGSFL_NONE, NULL }, /* Rs */
235         { ARGSFL_DELIM, NULL }, /* Sc */
236         { ARGSFL_NONE, NULL }, /* So */
237         { ARGSFL_DELIM, NULL }, /* Sq */
238         { ARGSFL_NONE, NULL }, /* Sm */
239         { ARGSFL_DELIM, NULL }, /* Sx */
240         { ARGSFL_DELIM, NULL }, /* Sy */
241         { ARGSFL_DELIM, NULL }, /* Tn */
242         { ARGSFL_DELIM, NULL }, /* Ux */
243         { ARGSFL_DELIM, NULL }, /* Xc */
244         { ARGSFL_NONE, NULL }, /* Xo */
245         { ARGSFL_NONE, NULL }, /* Fo */
246         { ARGSFL_NONE, NULL }, /* Fc */
247         { ARGSFL_NONE, NULL }, /* Oo */
248         { ARGSFL_DELIM, NULL }, /* Oc */
249         { ARGSFL_NONE, args_Bk }, /* Bk */
250         { ARGSFL_NONE, NULL }, /* Ek */
251         { ARGSFL_NONE, NULL }, /* Bt */
252         { ARGSFL_NONE, NULL }, /* Hf */
253         { ARGSFL_NONE, NULL }, /* Fr */
254         { ARGSFL_NONE, NULL }, /* Ud */
255         { ARGSFL_NONE, NULL }, /* Lb */
256         { ARGSFL_NONE, NULL }, /* Lp */
257         { ARGSFL_DELIM, NULL }, /* Lk */
258         { ARGSFL_DELIM, NULL }, /* Mt */
259         { ARGSFL_DELIM, NULL }, /* Brq */
```

```
260         { ARGSFL_NONE, NULL }, /* Bro */
261         { ARGSFL_DELIM, NULL }, /* Brc */
262         { ARGSFL_NONE, NULL }, /* %C */
263         { ARGSFL_NONE, NULL }, /* Es */
264         { ARGSFL_NONE, NULL }, /* En */
265         { ARGSFL_NONE, NULL }, /* Dx */
266         { ARGSFL_NONE, NULL }, /* %Q */
267         { ARGSFL_NONE, NULL }, /* br */
268         { ARGSFL_NONE, NULL }, /* sp */
269         { ARGSFL_NONE, NULL }, /* %U */
270         { ARGSFL_NONE, NULL }, /* Ta */
271 };


274 /*
275  * Parse an argument from line text.  This comes in the form of -key
276  * [value0...], which may either have a single mandatory value, at least
277  * one mandatory value, an optional single value, or no value.
278  */
279 enum margverr
280 mdoc_argv(struct mdoc *m, int line, enum mdoct tok,
281                 struct mdoc_arg **v, int *pos, char *buf)
282 {
283         char             *p, sv;
284         struct mdoc_argv tmp;
285         struct mdoc_arg  *arg;
286         const enum mdocargt *ap;

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

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

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

299         assert(*pos > 0);

301         for ( ; buf[*pos] ; (*pos)++)
302                 if (' ' == buf[*pos] && '\\' != buf[*pos - 1])
303                         break;

305         /*
306          * We want to nil-terminate the word to look it up (it's easier
307          * that way).  But we may not have a flag, in which case we need
308          * to restore the line as-is.  So keep around the stray byte,
309          * which we'll reset upon exiting (if necessary).
310          */

312         if ('\0' != (sv = buf[*pos]))
313                 buf[(*pos)++] = '\0';

315         /*
316          * Now look up the word as a flag.  Use temporary storage that
317          * we'll copy into the node's flags, if necessary.
318          */

320         memset(&tmp, 0, sizeof(struct mdoc_argv));

322         tmp.line = line;
323         tmp.pos = *pos;
324         tmp.arg = MDOC_ARG_MAX;
```



```
326         while (MDOC_ARG_MAX != (tmp.arg = *ap++))
327                 if (0 == strcmp(p, mdoc_argnames[tmp.arg]))
328                         break;

330         if (MDOC_ARG_MAX == tmp.arg) {
331                 /*
332                  * The flag was not found.
333                  * Restore saved zeroed byte and return as a word.
334                  */
335                 if (sv)
336                         buf[*pos - 1] = sv;
337                 return(ARGV_WORD);
338         }

340         /* Read to the next word (the argument). */

342         while (buf[*pos] && ' ' == buf[*pos])
343                 (*pos)++;

345         switch (argvflags[tmp.arg]) {
346         case (ARGV_SINGLE):
347                 if ( ! argv_single(m, line, &tmp, pos, buf))
348                         return(ARGV_ERROR);
349                 break;
350         case (ARGV_MULTI):
351                 if ( ! argv_multi(m, line, &tmp, pos, buf))
352                         return(ARGV_ERROR);
353                 break;
354         case (ARGV_OPT_SINGLE):
355                 if ( ! argv_opt_single(m, line, &tmp, pos, buf))
356                         return(ARGV_ERROR);
357                 break;
358         case (ARGV_NONE):
359                 break;
360         }

362         if (NULL == (arg = *v))
363                 arg = *v = mandoc_calloc(1, sizeof(struct mdoc_arg));

365         arg->argc++;
366         arg->argv = mandoc_realloc
367                 (arg->argv, arg->argc * sizeof(struct mdoc_argv));

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

372         return(ARGV_ARG);
373 }

375 void
376 mdoc_argv_free(struct mdoc_arg *p)
377 {
378         int              i;

380         if (NULL == p)
381                 return;

383         if (p->refcnt) {
384                 --(p->refcnt);
385                 if (p->refcnt)
386                         return;
387         }
388         assert(p->argc);

390         for (i = (int)p->argc - 1; i >= 0; i--)
391                 argn_free(p, i);
```

```
393         free(p->argv);
394         free(p);
395 }

397 static void
398 argn_free(struct mdoc_arg *p, int iarg)
399 {
400         struct mdoc_argv *arg;
401         int              j;

403         arg = &p->argv[iarg];

405         if (arg->sz && arg->value) {
406                 for (j = (int)arg->sz - 1; j >= 0; j--)
407                         free(arg->value[j]);
408                 free(arg->value);
409         }

411         for (--p->argc; iarg < (int)p->argc; iarg++)
412                 p->argv[iarg] = p->argv[iarg+1];
413 }

415 enum margserr
416 mdoc_zargs(struct mdoc *m, int line, int *pos, char *buf, char **v)
417 {

419         return(args(m, line, pos, buf, ARGSFL_NONE, v));
420 }

422 enum margserr
423 mdoc_args(struct mdoc *m, int line, int *pos,
424                 char *buf, enum mdoct tok, char **v)
425 {
426         enum argsflag     fl;
427         struct mdoc_node *n;

429         fl = mdocargs[tok].flags;

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

434         /*
435          * We know that we're in an 'It', so it's reasonable to expect
436          * us to be sitting in a 'Bl'.  Someday this may not be the case
437          * (if we allow random 'It's sitting out there), so provide a
438          * safe fall-back into the default behaviour.
439          */

441         for (n = m->last; n; n = n->parent)
442                 if (MDOC_Bl == n->tok)
443                         if (LIST_column == n->norm->Bl.type) {
444                                 fl = ARGSFL_TABSEP;
445                                 break;
446                         }

448         return(args(m, line, pos, buf, fl, v));
449 }

451 static enum margserr
452 args(struct mdoc *m, int line, int *pos,
453                 char *buf, enum argsflag fl, char **v)
454 {
455         char            *p, *pp;
456         enum margserr    rc;
```



```
458         if ('\0' == buf[*pos]) {
459                 if (MDOC_PPHRASE & m->flags)
460                         return(ARGS_EOLN);
461                 /*
462                  * If we're not in a partial phrase and the flag for
463                  * being a phrase literal is still set, the punctuation
464                  * is unterminated.
465                  */
466                 if (MDOC_PHRASELIT & m->flags)
467                         mdoc_pmsg(m, line, *pos, MANDOCERR_BADQUOTE);

469                 m->flags &= ~MDOC_PHRASELIT;
470                 return(ARGS_EOLN);
471         }

473         *v = &buf[*pos];

475         if (ARGSFL_DELIM == fl)
476                 if (args_checkpunct(buf, *pos))
477                         return(ARGS_PUNCT);

479         /*
480          * First handle TABSEP items, restricted to 'Bl -column'.  This
481          * ignores conventional token parsing and instead uses tabs or
482          * 'Ta' macros to separate phrases.  Phrases are parsed again
483          * for arguments at a later phase.
484          */

486         if (ARGSFL_TABSEP == fl) {
487                 /* Scan ahead to tab (can't be escaped). */
488                 p = strchr(*v, '\t');
489                 pp = NULL;

491                 /* Scan ahead to unescaped 'Ta'. */
492                 if ( ! (MDOC_PHRASELIT & m->flags))
493                         for (pp = *v; ; pp++) {
494                                 if (NULL == (pp = strstr(pp, "Ta")))
495                                         break;
496                                 if (pp > *v && ' ' != *(pp - 1))
497                                         continue;
498                                 if (' ' == *(pp + 2) || '\0' == *(pp + 2))
499                                         break;
500                         }

502                 /* By default, assume a phrase. */
503                 rc = ARGS_PHRASE;

505                 /*
506                  * Adjust new-buffer position to be beyond delimiter
507                  * mark (e.g., Ta -> end + 2).
508                  */
509                 if (p && pp) {
510                         *pos += pp < p ? 2 : 1;
511                         rc = pp < p ? ARGS_PHRASE : ARGS_PPHRASE;
512                         p = pp < p ? pp : p;
513                 } else if (p && ! pp) {
514                         rc = ARGS_PPHRASE;
515                         *pos += 1;
516                 } else if (pp && ! p) {
517                         p = pp;
518                         *pos += 2;
519                 } else {
520                         rc = ARGS_PEND;
521                         p = strchr(*v, 0);
522                 }
```

```
524                 /* Whitespace check for eoln case... */
525                 if ('\0' == *p && ' ' == *(p - 1))
526                         mdoc_pmsg(m, line, *pos, MANDOCERR_EOLNSPACE);
527
528                 *pos += (int)(p - *v);
529
530                 /* Strip delimiter's preceding whitespace. */
531                 pp = p - 1;
532                 while (pp > *v && ' ' == *pp) {
533                         if (pp > *v && '\\' == *(pp - 1))
534                                 break;
535                         pp--;
536                 }
537                 *(pp + 1) = 0;
538
539                 /* Strip delimiter's proceeding whitespace. */
540                 for (pp = &buf[*pos]; ' ' == *pp; pp++, (*pos)++)
541                         /* Skip ahead. */ ;
542
543                 return(rc);
544         }
545
546         /*
547          * Process a quoted literal.  A quote begins with a double-quote
548          * and ends with a double-quote NOT preceded by a double-quote.
549          * Whitespace is NOT involved in literal termination.
550          */
551
552         if (MDOC_PHRASELIT & m->flags || '\"' == buf[*pos]) {
553                 if ( ! (MDOC_PHRASELIT & m->flags))
554                         *v = &buf[++(*pos)];
555
556                 if (MDOC_PPHRASE & m->flags)
557                         m->flags |= MDOC_PHRASELIT;
558
559                 for ( ; buf[*pos]; (*pos)++) {
560                         if ('\"' != buf[*pos])
561                                 continue;
562                         if ('\"' != buf[*pos + 1])
563                                 break;
564                         (*pos)++;
565                 }
566
567                 if ('\0' == buf[*pos]) {
568                         if (MDOC_PPHRASE & m->flags)
569                                 return(ARGS_QWORD);
570                         mdoc_pmsg(m, line, *pos, MANDOCERR_BADQUOTE);
571                         return(ARGS_QWORD);
572                 }
573
574                 m->flags &= ~MDOC_PHRASELIT;
575                 buf[(*pos)++] = '\0';
576
577                 if ('\0' == buf[*pos])
578                         return(ARGS_QWORD);
579
580                 while (' ' == buf[*pos])
581                         (*pos)++;
582
583                 if ('\0' == buf[*pos])
584                         mdoc_pmsg(m, line, *pos, MANDOCERR_EOLNSPACE);
585
586                 return(ARGS_QWORD);
587         }
588
589         p = &buf[*pos];
```



```
590         *v = mandoc_getarg(m->parse, &p, line, pos);
591
592         return(ARGS_WORD);
593 }
594
595 /*
596  * Check if the string consists only of space-separated closing
597  * delimiters.  This is a bit of a dance: the first must be a close
598  * delimiter, but it may be followed by middle delimiters.  Arbitrary
599  * whitespace may separate these tokens.
600  */
601 static int
602 args_checkpunct(const char *buf, int i)
603 {
604         int              j;
605         char             dbuf[DELIMSZ];
606         enum mdelim      d;
607
608         /* First token must be a close-delimiter. */
609
610         for (j = 0; buf[i] && ' ' != buf[i] && j < DELIMSZ; j++, i++)
611                 dbuf[j] = buf[i];
612
613         if (DELIMSZ == j)
614                 return(0);
615
616         dbuf[j] = '\0';
617         if (DELIM_CLOSE != mdoc_isdelim(dbuf))
618                 return(0);
619
620         while (' ' == buf[i])
621                 i++;
622
623         /* Remaining must NOT be open/none. */
624
625         while (buf[i]) {
626                 j = 0;
627                 while (buf[i] && ' ' != buf[i] && j < DELIMSZ)
628                         dbuf[j++] = buf[i++];
629
630                 if (DELIMSZ == j)
631                         return(0);
632
633                 dbuf[j] = '\0';
634                 d = mdoc_isdelim(dbuf);
635                 if (DELIM_NONE == d || DELIM_OPEN == d)
636                         return(0);
637
638                 while (' ' == buf[i])
639                         i++;
640         }
641
642         return('\0' == buf[i]);
643 }
644
645 static int
646 argv_multi(struct mdoc *m, int line,
647                 struct mdoc_argv *v, int *pos, char *buf)
648 {
649         enum margserr    ac;
650         char            *p;
651
652         for (v->sz = 0; ; v->sz++) {
653                 if ('-' == buf[*pos])
654                         break;
655                 ac = args(m, line, pos, buf, ARGSFL_NONE, &p);
```

```
656                 if (ARGS_ERROR == ac)
657                         return(0);
658                 else if (ARGS_EOLN == ac)
659                         break;
660
661                 if (0 == v->sz % MULTI_STEP)
662                         v->value = mandoc_realloc(v->value,
663                                 (v->sz + MULTI_STEP) * sizeof(char *));
664
665                 v->value[(int)v->sz] = mandoc_strdup(p);
666         }
667
668         return(1);
669 }
670
671 static int
672 argv_opt_single(struct mdoc *m, int line,
673                 struct mdoc_argv *v, int *pos, char *buf)
674 {
675         enum margserr    ac;
676         char            *p;
677
678         if ('-' == buf[*pos])
679                 return(1);
680
681         ac = args(m, line, pos, buf, ARGSFL_NONE, &p);
682         if (ARGS_ERROR == ac)
683                 return(0);
684         if (ARGS_EOLN == ac)
685                 return(1);
686
687         v->sz = 1;
688         v->value = mandoc_malloc(sizeof(char *));
689         v->value[0] = mandoc_strdup(p);
690
691         return(1);
692 }
693
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;
701
702         ppos = *pos;
703
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);
710
711         v->sz = 1;
712         v->value = mandoc_malloc(sizeof(char *));
713         v->value[0] = mandoc_strdup(p);
714
715         return(1);
716 }
```

```
**********************************************************
    2287 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mdoc_hash.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_hash.c,v 1.18 2011/07/24 18:15:14 kristaps Exp $ */
/*
 * Copyright (c) 2008, 2009 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 <ctype.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

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

static  unsigned char    table[27 * 12];

/*
 * XXX - this hash has global scope, so if intended for use as a library
 * with multiple callers, it will need re-invocation protection.
 */
void
mdoc_hash_init(void)
{
        int              i, j, major;
        const char      *p;

        memset(table, UCHAR_MAX, sizeof(table));

        for (i = 0; i < (int)MDOC_MAX; i++) {
                p = mdoc_macronames[i];

                if (isalpha((unsigned char)p[1]))
                        major = 12 * (tolower((unsigned char)p[1]) - 97);
                else
                        major = 12 * 26;

                for (j = 0; j < 12; j++)
                        if (UCHAR_MAX == table[major + j]) {
                                table[major + j] = (unsigned char)i;
                                break;
                        }
```



```
                assert(j < 12);
        }
}

enum mdoct
mdoc_hash_find(const char *p)
{
        int               major, i, j;

        if (0 == p[0])
                return(MDOC_MAX);
        if ( ! isalpha((unsigned char)p[0]) && '%' != p[0])
                return(MDOC_MAX);

        if (isalpha((unsigned char)p[1]))
                major = 12 * (tolower((unsigned char)p[1]) - 97);
        else if ('1' == p[1])
                major = 12 * 26;
        else
                return(MDOC_MAX);

        if (p[2] && p[3])
                return(MDOC_MAX);

        for (j = 0; j < 12; j++) {
                if (UCHAR_MAX == (i = table[major + j]))
                        break;
                if (0 == strcmp(p, mdoc_macronames[i]))
                        return((enum mdoct)i);
        }

        return(MDOC_MAX);
}
```

```
**********************************************************
   43911 Tue Jul 15 13:48:06 2014
new/usr/src/cmd/mandoc/mdoc_html.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_html.c,v 1.182 2011/11/03 20:37:00 schwarze 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 <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

#define INDENT           5

#define MDOC_ARGS         const struct mdoc_meta *m, \
                          const struct mdoc_node *n, \
                          struct html *h

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

struct  htmlmdoc {
        int             (*pre)(MDOC_ARGS);
        void            (*post)(MDOC_ARGS);
};

static  void              print_mdoc(MDOC_ARGS);
static  void              print_mdoc_head(MDOC_ARGS);
static  void              print_mdoc_node(MDOC_ARGS);
static  void              print_mdoc_nodelist(MDOC_ARGS);
static  void              synopsis_pre(struct html *,
                                const struct mdoc_node *);

static  void              a2width(const char *, struct roffsu *);
static  void              a2offs(const char *, struct roffsu *);

static  void              mdoc_root_post(MDOC_ARGS);
```



```
static  int               mdoc_root_pre(MDOC_ARGS);

static  void              mdoc__x_post(MDOC_ARGS);
static  int               mdoc__x_pre(MDOC_ARGS);
static  int               mdoc_ad_pre(MDOC_ARGS);
static  int               mdoc_an_pre(MDOC_ARGS);
static  int               mdoc_ap_pre(MDOC_ARGS);
static  int               mdoc_ar_pre(MDOC_ARGS);
static  int               mdoc_bd_pre(MDOC_ARGS);
static  int               mdoc_bf_pre(MDOC_ARGS);
static  void              mdoc_bk_post(MDOC_ARGS);
static  int               mdoc_bk_pre(MDOC_ARGS);
static  int               mdoc_bl_pre(MDOC_ARGS);
static  int               mdoc_bt_pre(MDOC_ARGS);
static  int               mdoc_bx_pre(MDOC_ARGS);
static  int               mdoc_cd_pre(MDOC_ARGS);
static  int               mdoc_d1_pre(MDOC_ARGS);
static  int               mdoc_dv_pre(MDOC_ARGS);
static  int               mdoc_fa_pre(MDOC_ARGS);
static  int               mdoc_fd_pre(MDOC_ARGS);
static  int               mdoc_fl_pre(MDOC_ARGS);
static  int               mdoc_fn_pre(MDOC_ARGS);
static  int               mdoc_ft_pre(MDOC_ARGS);
static  int               mdoc_em_pre(MDOC_ARGS);
static  int               mdoc_er_pre(MDOC_ARGS);
static  int               mdoc_ev_pre(MDOC_ARGS);
static  int               mdoc_ex_pre(MDOC_ARGS);
static  void              mdoc_fo_post(MDOC_ARGS);
static  int               mdoc_fo_pre(MDOC_ARGS);
static  int               mdoc_ic_pre(MDOC_ARGS);
static  int               mdoc_igndelim_pre(MDOC_ARGS);
static  int               mdoc_in_pre(MDOC_ARGS);
static  int               mdoc_it_pre(MDOC_ARGS);
static  int               mdoc_lb_pre(MDOC_ARGS);
static  int               mdoc_li_pre(MDOC_ARGS);
static  int               mdoc_lk_pre(MDOC_ARGS);
static  int               mdoc_mt_pre(MDOC_ARGS);
static  int               mdoc_ms_pre(MDOC_ARGS);
static  int               mdoc_nd_pre(MDOC_ARGS);
static  int               mdoc_nm_pre(MDOC_ARGS);
static  int               mdoc_ns_pre(MDOC_ARGS);
static  int               mdoc_pa_pre(MDOC_ARGS);
static  void              mdoc_pf_post(MDOC_ARGS);
static  int               mdoc_pp_pre(MDOC_ARGS);
static  void              mdoc_quote_post(MDOC_ARGS);
static  int               mdoc_quote_pre(MDOC_ARGS);
static  int               mdoc_rs_pre(MDOC_ARGS);
static  int               mdoc_rv_pre(MDOC_ARGS);
static  int               mdoc_sh_pre(MDOC_ARGS);
static  int               mdoc_sm_pre(MDOC_ARGS);
static  int               mdoc_sp_pre(MDOC_ARGS);
static  int               mdoc_ss_pre(MDOC_ARGS);
static  int               mdoc_sx_pre(MDOC_ARGS);
static  int               mdoc_sy_pre(MDOC_ARGS);
static  int               mdoc_ud_pre(MDOC_ARGS);
static  int               mdoc_va_pre(MDOC_ARGS);
static  int               mdoc_vt_pre(MDOC_ARGS);
static  int               mdoc_xr_pre(MDOC_ARGS);
static  int               mdoc_xx_pre(MDOC_ARGS);

static  const struct htmlmdoc mdocs[MDOC_MAX] = {
        {mdoc_ap_pre, NULL}, /* Ap */
        {NULL, NULL}, /* Dd */
        {NULL, NULL}, /* Dt */
        {NULL, NULL}, /* Os */
        {mdoc_sh_pre, NULL }, /* Sh */
```

```
128         {mdoc_ss_pre, NULL }, /* Ss */
129         {mdoc_pp_pre, NULL}, /* Pp */
130         {mdoc_d1_pre, NULL}, /* D1 */
131         {mdoc_d1_pre, NULL}, /* Dl */
132         {mdoc_bd_pre, NULL}, /* Bd */
133         {NULL, NULL}, /* Ed */
134         {mdoc_bl_pre, NULL}, /* Bl */
135         {NULL, NULL}, /* El */
136         {mdoc_it_pre, NULL}, /* It */
137         {mdoc_ad_pre, NULL}, /* Ad */
138         {mdoc_an_pre, NULL}, /* An */
139         {mdoc_ar_pre, NULL}, /* Ar */
140         {mdoc_cd_pre, NULL}, /* Cd */
141         {mdoc_fl_pre, NULL}, /* Cm */
142         {mdoc_dv_pre, NULL}, /* Dv */
143         {mdoc_er_pre, NULL}, /* Er */
144         {mdoc_ev_pre, NULL}, /* Ev */
145         {mdoc_ex_pre, NULL}, /* Ex */
146         {mdoc_fa_pre, NULL}, /* Fa */
147         {mdoc_fd_pre, NULL}, /* Fd */
148         {mdoc_fl_pre, NULL}, /* Fl */
149         {mdoc_fn_pre, NULL}, /* Fn */
150         {mdoc_ft_pre, NULL}, /* Ft */
151         {mdoc_ic_pre, NULL}, /* Ic */
152         {mdoc_in_pre, NULL}, /* In */
153         {mdoc_li_pre, NULL}, /* Li */
154         {mdoc_nd_pre, NULL}, /* Nd */
155         {mdoc_nm_pre, NULL}, /* Nm */
156         {mdoc_quote_pre, mdoc_quote_post}, /* Op */
157         {NULL, NULL}, /* Ot */
158         {mdoc_pa_pre, NULL}, /* Pa */
159         {mdoc_rv_pre, NULL}, /* Rv */
160         {NULL, NULL}, /* St */
161         {mdoc_va_pre, NULL}, /* Va */
162         {mdoc_vt_pre, NULL}, /* Vt */
163         {mdoc_xr_pre, NULL}, /* Xr */
164         {mdoc__x_pre, mdoc__x_post}, /* %A */
165         {mdoc__x_pre, mdoc__x_post}, /* %B */
166         {mdoc__x_pre, mdoc__x_post}, /* %D */
167         {mdoc__x_pre, mdoc__x_post}, /* %I */
168         {mdoc__x_pre, mdoc__x_post}, /* %J */
169         {mdoc__x_pre, mdoc__x_post}, /* %N */
170         {mdoc__x_pre, mdoc__x_post}, /* %O */
171         {mdoc__x_pre, mdoc__x_post}, /* %P */
172         {mdoc__x_pre, mdoc__x_post}, /* %R */
173         {mdoc__x_pre, mdoc__x_post}, /* %T */
174         {mdoc__x_pre, mdoc__x_post}, /* %V */
175         {NULL, NULL}, /* Ac */
176         {mdoc_quote_pre, mdoc_quote_post}, /* Ao */
177         {mdoc_quote_pre, mdoc_quote_post}, /* Aq */
178         {NULL, NULL}, /* At */
179         {NULL, NULL}, /* Bc */
180         {mdoc_bf_pre, NULL}, /* Bf */
181         {mdoc_quote_pre, mdoc_quote_post}, /* Bo */
182         {mdoc_quote_pre, mdoc_quote_post}, /* Bq */
183         {mdoc_xx_pre, NULL}, /* Bsx */
184         {mdoc_bx_pre, NULL}, /* Bx */
185         {NULL, NULL}, /* Db */
186         {NULL, NULL}, /* Dc */
187         {mdoc_quote_pre, mdoc_quote_post}, /* Do */
188         {mdoc_quote_pre, mdoc_quote_post}, /* Dq */
189         {NULL, NULL}, /* Ec */ /* FIXME: no space */
190         {NULL, NULL}, /* Ef */
191         {mdoc_em_pre, NULL}, /* Em */
192         {mdoc_quote_pre, mdoc_quote_post}, /* Eo */
193         {mdoc_xx_pre, NULL}, /* Fx */
```



```
194         {mdoc_ms_pre, NULL}, /* Ms */
195         {mdoc_igndelim_pre, NULL}, /* No */
196         {mdoc_ns_pre, NULL}, /* Ns */
197         {mdoc_xx_pre, NULL}, /* Nx */
198         {mdoc_xx_pre, NULL}, /* Ox */
199         {NULL, NULL}, /* Pc */
200         {mdoc_igndelim_pre, mdoc_pf_post}, /* Pf */
201         {mdoc_quote_pre, mdoc_quote_post}, /* Po */
202         {mdoc_quote_pre, mdoc_quote_post}, /* Pq */
203         {NULL, NULL}, /* Qc */
204         {mdoc_quote_pre, mdoc_quote_post}, /* Ql */
205         {mdoc_quote_pre, mdoc_quote_post}, /* Qo */
206         {mdoc_quote_pre, mdoc_quote_post}, /* Qq */
207         {NULL, NULL}, /* Re */
208         {mdoc_rs_pre, NULL}, /* Rs */
209         {NULL, NULL}, /* Sc */
210         {mdoc_quote_pre, mdoc_quote_post}, /* So */
211         {mdoc_quote_pre, mdoc_quote_post}, /* Sq */
212         {mdoc_sm_pre, NULL}, /* Sm */
213         {mdoc_sx_pre, NULL}, /* Sx */
214         {mdoc_sy_pre, NULL}, /* Sy */
215         {NULL, NULL}, /* Tn */
216         {mdoc_xx_pre, NULL}, /* Ux */
217         {NULL, NULL}, /* Xc */
218         {NULL, NULL}, /* Xo */
219         {mdoc_fo_pre, mdoc_fo_post}, /* Fo */
220         {NULL, NULL}, /* Fc */
221         {mdoc_quote_pre, mdoc_quote_post}, /* Oo */
222         {NULL, NULL}, /* Oc */
223         {mdoc_bk_pre, mdoc_bk_post}, /* Bk */
224         {NULL, NULL}, /* Ek */
225         {mdoc_bt_pre, NULL}, /* Bt */
226         {NULL, NULL}, /* Hf */
227         {NULL, NULL}, /* Fr */
228         {mdoc_ud_pre, NULL}, /* Ud */
229         {mdoc_lb_pre, NULL}, /* Lb */
230         {mdoc_pp_pre, NULL}, /* Lp */
231         {mdoc_lk_pre, NULL}, /* Lk */
232         {mdoc_mt_pre, NULL}, /* Mt */
233         {mdoc_quote_pre, mdoc_quote_post}, /* Brq */
234         {mdoc_quote_pre, mdoc_quote_post}, /* Bro */
235         {NULL, NULL}, /* Brc */
236         {mdoc__x_pre, mdoc__x_post}, /* %C */
237         {NULL, NULL}, /* Es */  /* TODO */
238         {NULL, NULL}, /* En */  /* TODO */
239         {mdoc_xx_pre, NULL}, /* Dx */
240         {mdoc__x_pre, mdoc__x_post}, /* %Q */
241         {mdoc_sp_pre, NULL}, /* br */
242         {mdoc_sp_pre, NULL}, /* sp */
243         {mdoc__x_pre, mdoc__x_post}, /* %U */
244         {NULL, NULL}, /* Ta */
245 };

247 static  const char * const lists[LIST_MAX] = {
248         NULL,
249         "list-bul",
250         "list-col",
251         "list-dash",
252         "list-diag",
253         "list-enum",
254         "list-hang",
255         "list-hyph",
256         "list-inset",
257         "list-item",
258         "list-ohang",
259         "list-tag"
```

```
};

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

        print_mdoc(mdoc_meta(m), mdoc_node(m), (struct html *)arg);
        putchar('\n');
}


/*
 * Calculate the scaling unit passed in a '-width' argument.  This uses
 * either a native scaling unit (e.g., 1i, 2m) or the string length of
 * the value.
 */
static void
a2width(const char *p, struct roffsu *su)
{

        if ( ! a2roffsu(p, su, SCALE_MAX)) {
                su->unit = SCALE_BU;
                su->scale = html_strlen(p);
        }
}


/*
 * See the same function in mdoc_term.c for documentation.
 */
static void
synopsis_pre(struct html *h, const struct mdoc_node *n)
{

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

        if (n->prev->tok == n->tok &&
                        MDOC_Fo != n->tok &&
                        MDOC_Ft != n->tok &&
                        MDOC_Fn != n->tok) {
                print_otag(h, TAG_BR, 0, NULL);
                return;
        }

        switch (n->prev->tok) {
        case (MDOC_Fd):
                /* FALLTHROUGH */
        case (MDOC_Fn):
                /* FALLTHROUGH */
        case (MDOC_Fo):
                /* FALLTHROUGH */
        case (MDOC_In):
                /* FALLTHROUGH */
        case (MDOC_Vt):
                print_otag(h, TAG_P, 0, NULL);
                break;
        case (MDOC_Ft):
                if (MDOC_Fn != n->tok && MDOC_Fo != n->tok) {
                        print_otag(h, TAG_P, 0, NULL);
                        break;
                }
                /* FALLTHROUGH */
        default:
                print_otag(h, TAG_BR, 0, NULL);
                break;
```



```
        }
}


/*
 * Calculate the scaling unit passed in an '-offset' argument.  This
 * uses either a native scaling unit (e.g., 1i, 2m), one of a set of
 * predefined strings (indent, etc.), or the string length of the value.
 */
static void
a2offs(const char *p, struct roffsu *su)
{

        /* FIXME: "right"? */

        if (0 == strcmp(p, "left"))
                SCALE_HS_INIT(su, 0);
        else if (0 == strcmp(p, "indent"))
                SCALE_HS_INIT(su, INDENT);
        else if (0 == strcmp(p, "indent-two"))
                SCALE_HS_INIT(su, INDENT * 2);
        else if ( ! a2roffsu(p, su, SCALE_MAX))
                SCALE_HS_INIT(su, html_strlen(p));
}


static void
print_mdoc(MDOC_ARGS)
{
        struct tag      *t, *tt;
        struct htmlpair  tag;

        PAIR_CLASS_INIT(&tag, "mandoc");

        if ( ! (HTML_FRAGMENT & h->oflags)) {
                print_gen_decls(h);
                t = print_otag(h, TAG_HTML, 0, NULL);
                tt = print_otag(h, TAG_HEAD, 0, NULL);
                print_mdoc_head(m, n, h);
                print_tagq(h, tt);
                print_otag(h, TAG_BODY, 0, NULL);
                print_otag(h, TAG_DIV, 1, &tag);
        } else
                t = print_otag(h, TAG_DIV, 1, &tag);

        print_mdoc_nodelist(m, n, h);
        print_tagq(h, t);
}


/* ARGSUSED */
static void
print_mdoc_head(MDOC_ARGS)
{

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

        if (m->arch)
                bufcat_fmt(h, " (%s)", m->arch);

        print_otag(h, TAG_TITLE, 0, NULL);
        print_text(h, h->buf);
}
```

```
393 static void
394 print_mdoc_nodelist(MDOC_ARGS)
395 {

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


403 static void
404 print_mdoc_node(MDOC_ARGS)
405 {
406         int              child;
407         struct tag      *t;

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

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

420                 /*
421                  * Make sure that if we're in a literal mode already
422                  * (i.e., within a <PRE>) don't print the newline.
423                  */
424                 if (' ' == *n->string && MDOC_LINE & n->flags)
425                         if ( ! (HTML_LITERAL & h->flags))
426                                 print_otag(h, TAG_BR, 0, NULL);
427                 if (MDOC_DELIMC & n->flags)
428                         h->flags |= HTML_NOSPACE;
429                 print_text(h, n->string);
430                 if (MDOC_DELIMO & n->flags)
431                         h->flags |= HTML_NOSPACE;
432                 return;
433         case (MDOC_EQN):
434                 print_eqn(h, n->eqn);
435                 break;
436         case (MDOC_TBL):
437                 /*
438                  * This will take care of initialising all of the table
439                  * state data for the first table, then tearing it down
440                  * for the last one.
441                  */
442                 print_tbl(h, n->span);
443                 return;
444         default:
445                 /*
446                  * Close out the current table, if it's open, and unset
447                  * the "meta" table state.  This will be reopened on the
448                  * next table element.
449                  */
450                 if (h->tblt) {
451                         print_tblclose(h);
452                         t = h->tags.head;
453                 }

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



```
458                 break;
459         }

461         if (HTML_KEEP & h->flags) {
462                 if (n->prev && n->prev->line != n->line) {
463                         h->flags &= ~HTML_KEEP;
464                         h->flags |= HTML_PREKEEP;
465                 } else if (NULL == n->prev) {
466                         if (n->parent && n->parent->line != n->line) {
467                                 h->flags &= ~HTML_KEEP;
468                                 h->flags |= HTML_PREKEEP;
469                         }
470                 }
471         }

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

476         print_stagq(h, t);

478         switch (n->type) {
479         case (MDOC_ROOT):
480                 mdoc_root_post(m, n, h);
481                 break;
482         case (MDOC_EQN):
483                 break;
484         default:
485                 if (mdocs[n->tok].post && ENDBODY_NOT == n->end)
486                         (*mdocs[n->tok].post)(m, n, h);
487                 break;
488         }
489 }

491 /* ARGSUSED */
492 static void
493 mdoc_root_post(MDOC_ARGS)
494 {
495         struct htmlpair  tag[3];
496         struct tag      *t, *tt;

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

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

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

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

515         PAIR_CLASS_INIT(&tag[0], "foot-os");
516         PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
517         print_otag(h, TAG_TD, 2, tag);
518         print_text(h, m->os);
519         print_tagq(h, t);
520 }


523 /* ARGSUSED */
```

```
524 static int
525 mdoc_root_pre(MDOC_ARGS)
526 {
527         struct htmlpair  tag[3];
528         struct tag      *t, *tt;
529         char             b[BUFSIZ], title[BUFSIZ];

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

533         if (m->arch) {
534                 strlcat(b, " (", BUFSIZ);
535                 strlcat(b, m->arch, BUFSIZ);
536                 strlcat(b, ")", BUFSIZ);
537         }

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

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

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

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

554         PAIR_CLASS_INIT(&tag[0], "head-ltitle");
555         print_otag(h, TAG_TD, 1, tag);
556         print_text(h, title);
557         print_stagq(h, tt);

559         PAIR_CLASS_INIT(&tag[0], "head-vol");
560         PAIR_INIT(&tag[1], ATTR_ALIGN, "center");
561         print_otag(h, TAG_TD, 2, tag);
562         print_text(h, b);
563         print_stagq(h, tt);

565         PAIR_CLASS_INIT(&tag[0], "head-rtitle");
566         PAIR_INIT(&tag[1], ATTR_ALIGN, "right");
567         print_otag(h, TAG_TD, 2, tag);
568         print_text(h, title);
569         print_tagq(h, t);
570         return(1);
571 }


574 /* ARGSUSED */
575 static int
576 mdoc_sh_pre(MDOC_ARGS)
577 {
578         struct htmlpair  tag;

580         if (MDOC_BLOCK == n->type) {
581                 PAIR_CLASS_INIT(&tag, "section");
582                 print_otag(h, TAG_DIV, 1, &tag);
583                 return(1);
584         } else if (MDOC_BODY == n->type)
585                 return(1);

587         bufinit(h);
588         bufcat(h, "x");
```



```
590         for (n = n->child; n && MDOC_TEXT == n->type; ) {
591                 bufcat_id(h, n->string);
592                 if (NULL != (n = n->next))
593                         bufcat_id(h, " ");
594         }

596         if (NULL == n) {
597                 PAIR_ID_INIT(&tag, h->buf);
598                 print_otag(h, TAG_H1, 1, &tag);
599         } else
600                 print_otag(h, TAG_H1, 0, NULL);

602         return(1);
603 }

605 /* ARGSUSED */
606 static int
607 mdoc_ss_pre(MDOC_ARGS)
608 {
609         struct htmlpair  tag;

611         if (MDOC_BLOCK == n->type) {
612                 PAIR_CLASS_INIT(&tag, "subsection");
613                 print_otag(h, TAG_DIV, 1, &tag);
614                 return(1);
615         } else if (MDOC_BODY == n->type)
616                 return(1);

618         bufinit(h);
619         bufcat(h, "x");

621         for (n = n->child; n && MDOC_TEXT == n->type; ) {
622                 bufcat_id(h, n->string);
623                 if (NULL != (n = n->next))
624                         bufcat_id(h, " ");
625         }

627         if (NULL == n) {
628                 PAIR_ID_INIT(&tag, h->buf);
629                 print_otag(h, TAG_H2, 1, &tag);
630         } else
631                 print_otag(h, TAG_H2, 0, NULL);

633         return(1);
634 }


637 /* ARGSUSED */
638 static int
639 mdoc_fl_pre(MDOC_ARGS)
640 {
641         struct htmlpair  tag;

643         PAIR_CLASS_INIT(&tag, "flag");
644         print_otag(h, TAG_B, 1, &tag);

646         /* ‘Cm’ has no leading hyphen. */

648         if (MDOC_Cm == n->tok)
649                 return(1);

651         print_text(h, "\\-");

653         if (n->child)
654                 h->flags |= HTML_NOSPACE;
655         else if (n->next && n->next->line == n->line)
```

```
656                 h->flags |= HTML_NOSPACE;

658         return(1);
659 }


662 /* ARGSUSED */
663 static int
664 mdoc_nd_pre(MDOC_ARGS)
665 {
666         struct htmlpair  tag;

668         if (MDOC_BODY != n->type)
669                 return(1);

671         /* XXX: this tag in theory can contain block elements. */

673         print_text(h, "\\(em");
674         PAIR_CLASS_INIT(&tag, "desc");
675         print_otag(h, TAG_SPAN, 1, &tag);
676         return(1);
677 }


680 static int
681 mdoc_nm_pre(MDOC_ARGS)
682 {
683         struct htmlpair  tag;
684         struct roffsu    su;
685         int              len;

687         switch (n->type) {
688         case (MDOC_ELEM):
689                 synopsis_pre(h, n);
690                 PAIR_CLASS_INIT(&tag, "name");
691                 print_otag(h, TAG_B, 1, &tag);
692                 if (NULL == n->child && m->name)
693                         print_text(h, m->name);
694                 return(1);
695         case (MDOC_HEAD):
696                 print_otag(h, TAG_TD, 0, NULL);
697                 if (NULL == n->child && m->name)
698                         print_text(h, m->name);
699                 return(1);
700         case (MDOC_BODY):
701                 print_otag(h, TAG_TD, 0, NULL);
702                 return(1);
703         default:
704                 break;
705         }

707         synopsis_pre(h, n);
708         PAIR_CLASS_INIT(&tag, "synopsis");
709         print_otag(h, TAG_TABLE, 1, &tag);

711         for (len = 0, n = n->child; n; n = n->next)
712                 if (MDOC_TEXT == n->type)
713                         len += html_strlen(n->string);

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

718         SCALE_HS_INIT(&su, (double)len);
719         bufinit(h);
720         bufcat_su(h, "width", &su);
721         PAIR_STYLE_INIT(&tag, h);
```



```
722         print_otag(h, TAG_COL, 1, &tag);
723         print_otag(h, TAG_COL, 0, NULL);
724         print_otag(h, TAG_TBODY, 0, NULL);
725         print_otag(h, TAG_TR, 0, NULL);
726         return(1);
727 }


730 /* ARGSUSED */
731 static int
732 mdoc_xr_pre(MDOC_ARGS)
733 {
734         struct htmlpair  tag[2];

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

739         PAIR_CLASS_INIT(&tag[0], "link-man");

741         if (h->base_man) {
742                 buffmt_man(h, n->child->string,
743                                 n->child->next ?
744                                 n->child->next->string : NULL);
745                 PAIR_HREF_INIT(&tag[1], h->buf);
746                 print_otag(h, TAG_A, 2, tag);
747         } else
748                 print_otag(h, TAG_A, 1, tag);

750         n = n->child;
751         print_text(h, n->string);

753         if (NULL == (n = n->next))
754                 return(0);

756         h->flags |= HTML_NOSPACE;
757         print_text(h, "(");
758         h->flags |= HTML_NOSPACE;
759         print_text(h, n->string);
760         h->flags |= HTML_NOSPACE;
761         print_text(h, ")");
762         return(0);
763 }


766 /* ARGSUSED */
767 static int
768 mdoc_ns_pre(MDOC_ARGS)
769 {

771         if ( ! (MDOC_LINE & n->flags))
772                 h->flags |= HTML_NOSPACE;
773         return(1);
774 }


777 /* ARGSUSED */
778 static int
779 mdoc_ar_pre(MDOC_ARGS)
780 {
781         struct htmlpair tag;

783         PAIR_CLASS_INIT(&tag, "arg");
784         print_otag(h, TAG_I, 1, &tag);
785         return(1);
786 }
```

```
789 /* ARGSUSED */
790 static int
791 mdoc_xx_pre(MDOC_ARGS)
792 {
793         const char      *pp;
794         struct htmlpair  tag;
795         int              flags;

797         switch (n->tok) {
798         case (MDOC_Bsx):
799                 pp = "BSD/OS";
800                 break;
801         case (MDOC_Dx):
802                 pp = "DragonFly";
803                 break;
804         case (MDOC_Fx):
805                 pp = "FreeBSD";
806                 break;
807         case (MDOC_Nx):
808                 pp = "NetBSD";
809                 break;
810         case (MDOC_Ox):
811                 pp = "OpenBSD";
812                 break;
813         case (MDOC_Ux):
814                 pp = "UNIX";
815                 break;
816         default:
817                 return(1);
818         }

820         PAIR_CLASS_INIT(&tag, "unix");
821         print_otag(h, TAG_SPAN, 1, &tag);

823         print_text(h, pp);
824         if (n->child) {
825                 flags = h->flags;
826                 h->flags |= HTML_KEEP;
827                 print_text(h, n->child->string);
828                 h->flags = flags;
829         }
830         return(0);
831 }


834 /* ARGSUSED */
835 static int
836 mdoc_bx_pre(MDOC_ARGS)
837 {
838         struct htmlpair  tag;

840         PAIR_CLASS_INIT(&tag, "unix");
841         print_otag(h, TAG_SPAN, 1, &tag);

843         if (NULL != (n = n->child)) {
844                 print_text(h, n->string);
845                 h->flags |= HTML_NOSPACE;
846                 print_text(h, "BSD");
847         } else {
848                 print_text(h, "BSD");
849                 return(0);
850         }

852         if (NULL != (n = n->next)) {
853                 h->flags |= HTML_NOSPACE;
```



```
854                 print_text(h, "-");
855                 h->flags |= HTML_NOSPACE;
856                 print_text(h, n->string);
857         }

859         return(0);
860 }

862 /* ARGSUSED */
863 static int
864 mdoc_it_pre(MDOC_ARGS)
865 {
866         struct roffsu    su;
867         enum mdoc_list   type;
868         struct htmlpair  tag[2];
869         const struct mdoc_node *bl;

871         bl = n->parent;
872         while (bl && MDOC_Bl != bl->tok)
873                 bl = bl->parent;

875         assert(bl);

877         type = bl->norm->Bl.type;

879         assert(lists[type]);
880         PAIR_CLASS_INIT(&tag[0], lists[type]);

882         bufinit(h);

884         if (MDOC_HEAD == n->type) {
885                 switch (type) {
886                 case(LIST_bullet):
887                         /* FALLTHROUGH */
888                 case(LIST_dash):
889                         /* FALLTHROUGH */
890                 case(LIST_item):
891                         /* FALLTHROUGH */
892                 case(LIST_hyphen):
893                         /* FALLTHROUGH */
894                 case(LIST_enum):
895                         return(0);
896                 case(LIST_diag):
897                         /* FALLTHROUGH */
898                 case(LIST_hang):
899                         /* FALLTHROUGH */
900                 case(LIST_inset):
901                         /* FALLTHROUGH */
902                 case(LIST_ohang):
903                         /* FALLTHROUGH */
904                 case(LIST_tag):
905                         SCALE_VS_INIT(&su, ! bl->norm->Bl.comp);
906                         bufcat_su(h, "margin-top", &su);
907                         PAIR_STYLE_INIT(&tag[1], h);
908                         print_otag(h, TAG_DT, 2, tag);
909                         if (LIST_diag != type)
910                                 break;
911                         PAIR_CLASS_INIT(&tag[0], "diag");
912                         print_otag(h, TAG_B, 1, tag);
913                         break;
914                 case(LIST_column):
915                         break;
916                 default:
917                         break;
918                 }
919         } else if (MDOC_BODY == n->type) {
```

```
920                 switch (type) {
921                 case(LIST_bullet):
922                         /* FALLTHROUGH */
923                 case(LIST_hyphen):
924                         /* FALLTHROUGH */
925                 case(LIST_dash):
926                         /* FALLTHROUGH */
927                 case(LIST_enum):
928                         /* FALLTHROUGH */
929                 case(LIST_item):
930                         SCALE_VS_INIT(&su, ! bl->norm->Bl.comp);
931                         bufcat_su(h, "margin-top", &su);
932                         PAIR_STYLE_INIT(&tag[1], h);
933                         print_otag(h, TAG_LI, 2, tag);
934                         break;
935                 case(LIST_diag):
936                         /* FALLTHROUGH */
937                 case(LIST_hang):
938                         /* FALLTHROUGH */
939                 case(LIST_inset):
940                         /* FALLTHROUGH */
941                 case(LIST_ohang):
942                         /* FALLTHROUGH */
943                 case(LIST_tag):
944                         if (NULL == bl->norm->Bl.width) {
945                                 print_otag(h, TAG_DD, 1, tag);
946                                 break;
947                         }
948                         a2width(bl->norm->Bl.width, &su);
949                         bufcat_su(h, "margin-left", &su);
950                         PAIR_STYLE_INIT(&tag[1], h);
951                         print_otag(h, TAG_DD, 2, tag);
952                         break;
953                 case(LIST_column):
954                         SCALE_VS_INIT(&su, ! bl->norm->Bl.comp);
955                         bufcat_su(h, "margin-top", &su);
956                         PAIR_STYLE_INIT(&tag[1], h);
957                         print_otag(h, TAG_TD, 2, tag);
958                         break;
959                 default:
960                         break;
961                 }
962         } else {
963                 switch (type) {
964                 case (LIST_column):
965                         print_otag(h, TAG_TR, 1, tag);
966                         break;
967                 default:
968                         break;
969                 }
970         }

972         return(1);
973 }

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

984         bufinit(h);
```



```
986         if (MDOC_BODY == n->type) {
987                 if (LIST_column == n->norm->Bl.type)
988                         print_otag(h, TAG_TBODY, 0, NULL);
989                 return(1);
990         }

992         if (MDOC_HEAD == n->type) {
993                 if (LIST_column != n->norm->Bl.type)
994                         return(0);

996                 /*
997                  * For each column, print out the <COL> tag with our
998                  * suggested width.  The last column gets min-width, as
999                  * in terminal mode it auto-sizes to the width of the
1000                 * screen and we want to preserve that behaviour.
1001                 */

1003                for (i = 0; i < (int)n->norm->Bl.ncols; i++) {
1004                        a2width(n->norm->Bl.cols[i], &su);
1005                        if (i < (int)n->norm->Bl.ncols - 1)
1006                                bufcat_su(h, "width", &su);
1007                        else
1008                                bufcat_su(h, "min-width", &su);
1009                        PAIR_STYLE_INIT(&tag[0], h);
1010                        print_otag(h, TAG_COL, 1, tag);
1011                }

1013                return(0);
1014        }

1016        SCALE_VS_INIT(&su, 0);
1017        bufcat_su(h, "margin-top", &su);
1018        bufcat_su(h, "margin-bottom", &su);
1019        PAIR_STYLE_INIT(&tag[0], h);

1021        assert(lists[n->norm->Bl.type]);
1022        strlcpy(buf, "list ", BUFSIZ);
1023        strlcat(buf, lists[n->norm->Bl.type], BUFSIZ);
1024        PAIR_INIT(&tag[1], ATTR_CLASS, buf);

1026        /* Set the block's left-hand margin. */

1028        if (n->norm->Bl.offs) {
1029                a2offs(n->norm->Bl.offs, &su);
1030                bufcat_su(h, "margin-left", &su);
1031        }

1033        switch (n->norm->Bl.type) {
1034        case(LIST_bullet):
1035                /* FALLTHROUGH */
1036        case(LIST_dash):
1037                /* FALLTHROUGH */
1038        case(LIST_hyphen):
1039                /* FALLTHROUGH */
1040        case(LIST_item):
1041                print_otag(h, TAG_UL, 2, tag);
1042                break;
1043        case(LIST_enum):
1044                print_otag(h, TAG_OL, 2, tag);
1045                break;
1046        case(LIST_diag):
1047                /* FALLTHROUGH */
1048        case(LIST_hang):
1049                /* FALLTHROUGH */
1050        case(LIST_inset):
1051                /* FALLTHROUGH */
```

```
1052         case(LIST_ohang):
1053                 /* FALLTHROUGH */
1054         case(LIST_tag):
1055                 print_otag(h, TAG_DL, 2, tag);
1056                 break;
1057         case(LIST_column):
1058                 print_otag(h, TAG_TABLE, 2, tag);
1059                 break;
1060         default:
1061                 abort();
1062                 /* NOTREACHED */
1063         }

1065         return(1);
1066 }

1068 /* ARGSUSED */
1069 static int
1070 mdoc_ex_pre(MDOC_ARGS)
1071 {
1072         struct tag      *t;
1073         struct htmlpair  tag;
1074         int              nchild;

1076         if (n->prev)
1077                 print_otag(h, TAG_BR, 0, NULL);

1079         PAIR_CLASS_INIT(&tag, "utility");

1081         print_text(h, "The");

1083         nchild = n->nchild;
1084         for (n = n->child; n; n = n->next) {
1085                 assert(MDOC_TEXT == n->type);

1087                 t = print_otag(h, TAG_B, 1, &tag);
1088                 print_text(h, n->string);
1089                 print_tagq(h, t);

1091                 if (nchild > 2 && n->next) {
1092                         h->flags |= HTML_NOSPACE;
1093                         print_text(h, ",");
1094                 }

1096                 if (n->next && NULL == n->next->next)
1097                         print_text(h, "and");
1098         }

1100         if (nchild > 1)
1101                 print_text(h, "utilities exit");
1102         else
1103                 print_text(h, "utility exits");

1105         print_text(h, "0 on success, and >0 if an error occurs.");
1106         return(0);
1107 }


1110 /* ARGSUSED */
1111 static int
1112 mdoc_em_pre(MDOC_ARGS)
1113 {
1114         struct htmlpair tag;

1116         PAIR_CLASS_INIT(&tag, "emph");
1117         print_otag(h, TAG_SPAN, 1, &tag);
```



```
1118         return(1);
1119 }


1122 /* ARGSUSED */
1123 static int
1124 mdoc_d1_pre(MDOC_ARGS)
1125 {
1126         struct htmlpair  tag[2];
1127         struct roffsu    su;

1129         if (MDOC_BLOCK != n->type)
1130                 return(1);

1132         SCALE_VS_INIT(&su, 0);
1133         bufinit(h);
1134         bufcat_su(h, "margin-top", &su);
1135         bufcat_su(h, "margin-bottom", &su);
1136         PAIR_STYLE_INIT(&tag[0], h);
1137         print_otag(h, TAG_BLOCKQUOTE, 1, tag);

1139         /* BLOCKQUOTE needs a block body. */

1141         PAIR_CLASS_INIT(&tag[0], "display");
1142         print_otag(h, TAG_DIV, 1, tag);

1144         if (MDOC_Dl == n->tok) {
1145                 PAIR_CLASS_INIT(&tag[0], "lit");
1146                 print_otag(h, TAG_CODE, 1, tag);
1147         }

1149         return(1);
1150 }


1153 /* ARGSUSED */
1154 static int
1155 mdoc_sx_pre(MDOC_ARGS)
1156 {
1157         struct htmlpair  tag[2];

1159         bufinit(h);
1160         bufcat(h, "#x");

1162         for (n = n->child; n; ) {
1163                 bufcat_id(h, n->string);
1164                 if (NULL != (n = n->next))
1165                         bufcat_id(h, " ");
1166         }

1168         PAIR_CLASS_INIT(&tag[0], "link-sec");
1169         PAIR_HREF_INIT(&tag[1], h->buf);

1171         print_otag(h, TAG_I, 1, tag);
1172         print_otag(h, TAG_A, 2, tag);
1173         return(1);
1174 }


1177 /* ARGSUSED */
1178 static int
1179 mdoc_bd_pre(MDOC_ARGS)
1180 {
1181         struct htmlpair          tag[2];
1182         int                      comp, sv;
1183         const struct mdoc_node  *nn;
```

```
1184         struct roffsu            su;
1185
1186         if (MDOC_HEAD == n->type)
1187                 return(0);
1188
1189         if (MDOC_BLOCK == n->type) {
1190                 comp = n->norm->Bd.comp;
1191                 for (nn = n; nn && ! comp; nn = nn->parent) {
1192                         if (MDOC_BLOCK != nn->type)
1193                                 continue;
1194                         if (MDOC_Ss == nn->tok || MDOC_Sh == nn->tok)
1195                                 comp = 1;
1196                         if (nn->prev)
1197                                 break;
1198                 }
1199                 if ( ! comp)
1200                         print_otag(h, TAG_P, 0, NULL);
1201                 return(1);
1202         }
1203
1204         SCALE_HS_INIT(&su, 0);
1205         if (n->norm->Bd.offs)
1206                 a2offs(n->norm->Bd.offs, &su);
1207
1208         bufinit(h);
1209         bufcat_su(h, "margin-left", &su);
1210         PAIR_STYLE_INIT(&tag[0], h);
1211
1212         if (DISP_unfilled != n->norm->Bd.type &&
1213                         DISP_literal != n->norm->Bd.type) {
1214                 PAIR_CLASS_INIT(&tag[1], "display");
1215                 print_otag(h, TAG_DIV, 2, tag);
1216                 return(1);
1217         }
1218
1219         PAIR_CLASS_INIT(&tag[1], "lit display");
1220         print_otag(h, TAG_PRE, 2, tag);
1221
1222         /* This can be recursive: save & set our literal state. */
1223
1224         sv = h->flags & HTML_LITERAL;
1225         h->flags |= HTML_LITERAL;
1226
1227         for (nn = n->child; nn; nn = nn->next) {
1228                 print_mdoc_node(m, nn, h);
1229                 /*
1230                  * If the printed node flushes its own line, then we
1231                  * needn't do it here as well.  This is hacky, but the
1232                  * notion of selective eoln whitespace is pretty dumb
1233                  * anyway, so don't sweat it.
1234                  */
1235                 switch (nn->tok) {
1236                 case (MDOC_Sm):
1237                         /* FALLTHROUGH */
1238                 case (MDOC_br):
1239                         /* FALLTHROUGH */
1240                 case (MDOC_sp):
1241                         /* FALLTHROUGH */
1242                 case (MDOC_Bl):
1243                         /* FALLTHROUGH */
1244                 case (MDOC_D1):
1245                         /* FALLTHROUGH */
1246                 case (MDOC_Dl):
1247                         /* FALLTHROUGH */
1248                 case (MDOC_Lp):
1249                         /* FALLTHROUGH */
```



```
1250                 case (MDOC_Pp):
1251                         continue;
1252                 default:
1253                         break;
1254                 }
1255                 if (nn->next && nn->next->line == nn->line)
1256                         continue;
1257                 else if (nn->next)
1258                         print_text(h, "\n");
1259
1260                 h->flags |= HTML_NOSPACE;
1261         }
1262
1263         if (0 == sv)
1264                 h->flags &= ~HTML_LITERAL;
1265
1266         return(0);
1267 }
1268
1269
1270 /* ARGSUSED */
1271 static int
1272 mdoc_pa_pre(MDOC_ARGS)
1273 {
1274         struct htmlpair tag;
1275
1276         PAIR_CLASS_INIT(&tag, "file");
1277         print_otag(h, TAG_I, 1, &tag);
1278         return(1);
1279 }
1280
1281
1282 /* ARGSUSED */
1283 static int
1284 mdoc_ad_pre(MDOC_ARGS)
1285 {
1286         struct htmlpair tag;
1287
1288         PAIR_CLASS_INIT(&tag, "addr");
1289         print_otag(h, TAG_I, 1, &tag);
1290         return(1);
1291 }
1292
1293
1294 /* ARGSUSED */
1295 static int
1296 mdoc_an_pre(MDOC_ARGS)
1297 {
1298         struct htmlpair tag;
1299
1300         /* TODO: -split and -nosplit (see termp_an_pre()). */
1301
1302         PAIR_CLASS_INIT(&tag, "author");
1303         print_otag(h, TAG_SPAN, 1, &tag);
1304         return(1);
1305 }
1306
1307
1308 /* ARGSUSED */
1309 static int
1310 mdoc_cd_pre(MDOC_ARGS)
1311 {
1312         struct htmlpair tag;
1313
1314         synopsis_pre(h, n);
1315         PAIR_CLASS_INIT(&tag, "config");
```

```
1316         print_otag(h, TAG_B, 1, &tag);
1317         return(1);
1318 }


1321 /* ARGSUSED */
1322 static int
1323 mdoc_dv_pre(MDOC_ARGS)
1324 {
1325         struct htmlpair tag;

1327         PAIR_CLASS_INIT(&tag, "define");
1328         print_otag(h, TAG_SPAN, 1, &tag);
1329         return(1);
1330 }


1333 /* ARGSUSED */
1334 static int
1335 mdoc_ev_pre(MDOC_ARGS)
1336 {
1337         struct htmlpair tag;

1339         PAIR_CLASS_INIT(&tag, "env");
1340         print_otag(h, TAG_SPAN, 1, &tag);
1341         return(1);
1342 }


1345 /* ARGSUSED */
1346 static int
1347 mdoc_er_pre(MDOC_ARGS)
1348 {
1349         struct htmlpair tag;

1351         PAIR_CLASS_INIT(&tag, "errno");
1352         print_otag(h, TAG_SPAN, 1, &tag);
1353         return(1);
1354 }


1357 /* ARGSUSED */
1358 static int
1359 mdoc_fa_pre(MDOC_ARGS)
1360 {
1361         const struct mdoc_node  *nn;
1362         struct htmlpair          tag;
1363         struct tag              *t;

1365         PAIR_CLASS_INIT(&tag, "farg");
1366         if (n->parent->tok != MDOC_Fo) {
1367                 print_otag(h, TAG_I, 1, &tag);
1368                 return(1);
1369         }

1371         for (nn = n->child; nn; nn = nn->next) {
1372                 t = print_otag(h, TAG_I, 1, &tag);
1373                 print_text(h, nn->string);
1374                 print_tagq(h, t);
1375                 if (nn->next) {
1376                         h->flags |= HTML_NOSPACE;
1377                         print_text(h, ",");
1378                 }
1379         }

1381         if (n->child && n->next && n->next->tok == MDOC_Fa) {
```



```
1382                 h->flags |= HTML_NOSPACE;
1383                 print_text(h, ",");
1384         }

1386         return(0);
1387 }


1390 /* ARGSUSED */
1391 static int
1392 mdoc_fd_pre(MDOC_ARGS)
1393 {
1394         struct htmlpair  tag[2];
1395         char             buf[BUFSIZ];
1396         size_t           sz;
1397         int              i;
1398         struct tag      *t;

1400         synopsis_pre(h, n);

1402         if (NULL == (n = n->child))
1403                 return(0);

1405         assert(MDOC_TEXT == n->type);

1407         if (strcmp(n->string, "#include")) {
1408                 PAIR_CLASS_INIT(&tag[0], "macro");
1409                 print_otag(h, TAG_B, 1, tag);
1410                 return(1);
1411         }

1413         PAIR_CLASS_INIT(&tag[0], "includes");
1414         print_otag(h, TAG_B, 1, tag);
1415         print_text(h, n->string);

1417         if (NULL != (n = n->next)) {
1418                 assert(MDOC_TEXT == n->type);
1419                 strlcpy(buf, '<' == *n->string || '"' == *n->string ?
1420                                 n->string + 1 : n->string, BUFSIZ);

1422                 sz = strlen(buf);
1423                 if (sz && ('>' == buf[sz - 1] || '"' == buf[sz - 1]))
1424                         buf[sz - 1] = '\0';

1426                 PAIR_CLASS_INIT(&tag[0], "link-includes");
1427
1428                 i = 1;
1429                 if (h->base_includes) {
1430                         buffmt_includes(h, buf);
1431                         PAIR_HREF_INIT(&tag[i], h->buf);
1432                         i++;
1433                 }

1435                 t = print_otag(h, TAG_A, i, tag);
1436                 print_text(h, n->string);
1437                 print_tagq(h, t);

1439                 n = n->next;
1440         }

1442         for ( ; n; n = n->next) {
1443                 assert(MDOC_TEXT == n->type);
1444                 print_text(h, n->string);
1445         }

1447         return(0);
```

```
1448 }


1451 /* ARGSUSED */
1452 static int
1453 mdoc_vt_pre(MDOC_ARGS)
1454 {
1455         struct htmlpair  tag;

1457         if (MDOC_BLOCK == n->type) {
1458                 synopsis_pre(h, n);
1459                 return(1);
1460         } else if (MDOC_ELEM == n->type) {
1461                 synopsis_pre(h, n);
1462         } else if (MDOC_HEAD == n->type)
1463                 return(0);

1465         PAIR_CLASS_INIT(&tag, "type");
1466         print_otag(h, TAG_SPAN, 1, &tag);
1467         return(1);
1468 }


1471 /* ARGSUSED */
1472 static int
1473 mdoc_ft_pre(MDOC_ARGS)
1474 {
1475         struct htmlpair  tag;

1477         synopsis_pre(h, n);
1478         PAIR_CLASS_INIT(&tag, "ftype");
1479         print_otag(h, TAG_I, 1, &tag);
1480         return(1);
1481 }


1484 /* ARGSUSED */
1485 static int
1486 mdoc_fn_pre(MDOC_ARGS)
1487 {
1488         struct tag      *t;
1489         struct htmlpair  tag[2];
1490         char             nbuf[BUFSIZ];
1491         const char      *sp, *ep;
1492         int              sz, i, pretty;

1494         pretty = MDOC_SYNPRETTY & n->flags;
1495         synopsis_pre(h, n);

1497         /* Split apart into type and name. */
1498         assert(n->child->string);
1499         sp = n->child->string;

1501         ep = strchr(sp, ' ');
1502         if (NULL != ep) {
1503                 PAIR_CLASS_INIT(&tag[0], "ftype");
1504                 t = print_otag(h, TAG_I, 1, tag);
1505
1506                 while (ep) {
1507                         sz = MIN((int)(ep - sp), BUFSIZ - 1);
1508                         (void)memcpy(nbuf, sp, (size_t)sz);
1509                         nbuf[sz] = '\0';
1510                         print_text(h, nbuf);
1511                         sp = ++ep;
1512                         ep = strchr(sp, ' ');
1513                 }
```



```
1514                 print_tagq(h, t);
1515         }

1517         PAIR_CLASS_INIT(&tag[0], "fname");

1519         /*
1520          * FIXME: only refer to IDs that we know exist.
1521          */

1523 #if 0
1524         if (MDOC_SYNPRETTY & n->flags) {
1525                 nbuf[0] = '\0';
1526                 html_idcat(nbuf, sp, BUFSIZ);
1527                 PAIR_ID_INIT(&tag[1], nbuf);
1528         } else {
1529                 strlcpy(nbuf, "#", BUFSIZ);
1530                 html_idcat(nbuf, sp, BUFSIZ);
1531                 PAIR_HREF_INIT(&tag[1], nbuf);
1532         }
1533 #endif

1535         t = print_otag(h, TAG_B, 1, tag);

1537         if (sp) {
1538                 strlcpy(nbuf, sp, BUFSIZ);
1539                 print_text(h, nbuf);
1540         }

1542         print_tagq(h, t);

1544         h->flags |= HTML_NOSPACE;
1545         print_text(h, "(");
1546         h->flags |= HTML_NOSPACE;

1548         PAIR_CLASS_INIT(&tag[0], "farg");
1549         bufinit(h);
1550         bufcat_style(h, "white-space", "nowrap");
1551         PAIR_STYLE_INIT(&tag[1], h);

1553         for (n = n->child->next; n; n = n->next) {
1554                 i = 1;
1555                 if (MDOC_SYNPRETTY & n->flags)
1556                         i = 2;
1557                 t = print_otag(h, TAG_I, i, tag);
1558                 print_text(h, n->string);
1559                 print_tagq(h, t);
1560                 if (n->next) {
1561                         h->flags |= HTML_NOSPACE;
1562                         print_text(h, ",");
1563                 }
1564         }

1566         h->flags |= HTML_NOSPACE;
1567         print_text(h, ")");

1569         if (pretty) {
1570                 h->flags |= HTML_NOSPACE;
1571                 print_text(h, ";");
1572         }

1574         return(0);
1575 }


1578 /* ARGSUSED */
1579 static int
```

```
1580 mdoc_sm_pre(MDOC_ARGS)
1581 {
1583         assert(n->child && MDOC_TEXT == n->child->type);
1584         if (0 == strcmp("on", n->child->string)) {
1585                 /*
1586                  * FIXME: no p->col to check.  Thus, if we have
1587                  *   .Bd -literal
1588                  *   .Sm off
1589                  *   1 2
1590                  *   .Sm on
1591                  *   3
1592                  *   .Ed
1593                  * the "3" is preceded by a space.
1594                  */
1595                 h->flags &= ~HTML_NOSPACE;
1596                 h->flags &= ~HTML_NONOSPACE;
1597         } else
1598                 h->flags |= HTML_NONOSPACE;
1600         return(0);
1601 }
1603 /* ARGSUSED */
1604 static int
1605 mdoc_pp_pre(MDOC_ARGS)
1606 {
1608         print_otag(h, TAG_P, 0, NULL);
1609         return(0);
1611 }
1613 /* ARGSUSED */
1614 static int
1615 mdoc_sp_pre(MDOC_ARGS)
1616 {
1617         struct roffsu    su;
1618         struct htmlpair  tag;
1620         SCALE_VS_INIT(&su, 1);
1622         if (MDOC_sp == n->tok) {
1623                 if (NULL != (n = n->child))
1624                         if ( ! a2roffsu(n->string, &su, SCALE_VS))
1625                                 SCALE_VS_INIT(&su, atoi(n->string));
1626         } else
1627                 su.scale = 0;
1629         bufinit(h);
1630         bufcat_su(h, "height", &su);
1631         PAIR_STYLE_INIT(&tag, h);
1632         print_otag(h, TAG_DIV, 1, &tag);
1634         /* So the div isn't empty: */
1635         print_text(h, "\\~");
1637         return(0);
1639 }
1641 /* ARGSUSED */
1642 static int
1643 mdoc_lk_pre(MDOC_ARGS)
1644 {
1645         struct htmlpair  tag[2];
```



```
1647         if (NULL == (n = n->child))
1648                 return(0);
1650         assert(MDOC_TEXT == n->type);
1652         PAIR_CLASS_INIT(&tag[0], "link-ext");
1653         PAIR_HREF_INIT(&tag[1], n->string);
1655         print_otag(h, TAG_A, 2, tag);
1657         if (NULL == n->next)
1658                 print_text(h, n->string);
1660         for (n = n->next; n; n = n->next)
1661                 print_text(h, n->string);
1663         return(0);
1664 }
1667 /* ARGSUSED */
1668 static int
1669 mdoc_mt_pre(MDOC_ARGS)
1670 {
1671         struct htmlpair  tag[2];
1672         struct tag      *t;
1674         PAIR_CLASS_INIT(&tag[0], "link-mail");
1676         for (n = n->child; n; n = n->next) {
1677                 assert(MDOC_TEXT == n->type);
1679                 bufinit(h);
1680                 bufcat(h, "mailto:");
1681                 bufcat(h, n->string);
1683                 PAIR_HREF_INIT(&tag[1], h->buf);
1684                 t = print_otag(h, TAG_A, 2, tag);
1685                 print_text(h, n->string);
1686                 print_tagq(h, t);
1687         }
1688
1689         return(0);
1690 }
1693 /* ARGSUSED */
1694 static int
1695 mdoc_fo_pre(MDOC_ARGS)
1696 {
1697         struct htmlpair  tag;
1698         struct tag      *t;
1700         if (MDOC_BODY == n->type) {
1701                 h->flags |= HTML_NOSPACE;
1702                 print_text(h, "(");
1703                 h->flags |= HTML_NOSPACE;
1704                 return(1);
1705         } else if (MDOC_BLOCK == n->type) {
1706                 synopsis_pre(h, n);
1707                 return(1);
1708         }
1710         /* XXX: we drop non-initial arguments as per groff. */
```

```
1712         assert(n->child);
1713         assert(n->child->string);

1715         PAIR_CLASS_INIT(&tag, "fname");
1716         t = print_otag(h, TAG_B, 1, &tag);
1717         print_text(h, n->child->string);
1718         print_tagq(h, t);
1719         return(0);
1720 }


1723 /* ARGSUSED */
1724 static void
1725 mdoc_fo_post(MDOC_ARGS)
1726 {

1728         if (MDOC_BODY != n->type)
1729                 return;
1730         h->flags |= HTML_NOSPACE;
1731         print_text(h, ")");
1732         h->flags |= HTML_NOSPACE;
1733         print_text(h, ";");
1734 }


1737 /* ARGSUSED */
1738 static int
1739 mdoc_in_pre(MDOC_ARGS)
1740 {
1741         struct tag      *t;
1742         struct htmlpair  tag[2];
1743         int              i;

1745         synopsis_pre(h, n);

1747         PAIR_CLASS_INIT(&tag[0], "includes");
1748         print_otag(h, TAG_B, 1, tag);

1750         /*
1751          * The first argument of the ‘In’ gets special treatment as
1752          * being a linked value.  Subsequent values are printed
1753          * afterward.  groff does similarly.  This also handles the case
1754          * of no children.
1755          */

1757         if (MDOC_SYNPRETTY & n->flags && MDOC_LINE & n->flags)
1758                 print_text(h, "#include");

1760         print_text(h, "<");
1761         h->flags |= HTML_NOSPACE;

1763         if (NULL != (n = n->child)) {
1764                 assert(MDOC_TEXT == n->type);

1766                 PAIR_CLASS_INIT(&tag[0], "link-includes");

1768                 i = 1;
1769                 if (h->base_includes) {
1770                         buffmt_includes(h, n->string);
1771                         PAIR_HREF_INIT(&tag[i], h->buf);
1772                         i++;
1773                 }

1775                 t = print_otag(h, TAG_A, i, tag);
1776                 print_text(h, n->string);
1777                 print_tagq(h, t);
```



```
1779                 n = n->next;
1780         }

1782         h->flags |= HTML_NOSPACE;
1783         print_text(h, ">");

1785         for ( ; n; n = n->next) {
1786                 assert(MDOC_TEXT == n->type);
1787                 print_text(h, n->string);
1788         }

1790         return(0);
1791 }


1794 /* ARGSUSED */
1795 static int
1796 mdoc_ic_pre(MDOC_ARGS)
1797 {
1798         struct htmlpair tag;

1800         PAIR_CLASS_INIT(&tag, "cmd");
1801         print_otag(h, TAG_B, 1, &tag);
1802         return(1);
1803 }


1806 /* ARGSUSED */
1807 static int
1808 mdoc_rv_pre(MDOC_ARGS)
1809 {
1810         struct htmlpair  tag;
1811         struct tag      *t;
1812         int              nchild;

1814         if (n->prev)
1815                 print_otag(h, TAG_BR, 0, NULL);

1817         PAIR_CLASS_INIT(&tag, "fname");

1819         print_text(h, "The");

1821         nchild = n->nchild;
1822         for (n = n->child; n; n = n->next) {
1823                 assert(MDOC_TEXT == n->type);

1825                 t = print_otag(h, TAG_B, 1, &tag);
1826                 print_text(h, n->string);
1827                 print_tagq(h, t);

1829                 h->flags |= HTML_NOSPACE;
1830                 print_text(h, "()");

1832                 if (nchild > 2 && n->next) {
1833                         h->flags |= HTML_NOSPACE;
1834                         print_text(h, ",");
1835                 }

1837                 if (n->next && NULL == n->next->next)
1838                         print_text(h, "and");
1839         }

1841         if (nchild > 1)
1842                 print_text(h, "functions return");
1843         else
```

```
1844                 print_text(h, "function returns");

1846         print_text(h, "the value 0 if successful; otherwise the value "
1847                         "-1 is returned and the global variable");

1849         PAIR_CLASS_INIT(&tag, "var");
1850         t = print_otag(h, TAG_B, 1, &tag);
1851         print_text(h, "errno");
1852         print_tagq(h, t);
1853         print_text(h, "is set to indicate the error.");
1854         return(0);
1855 }


1858 /* ARGSUSED */
1859 static int
1860 mdoc_va_pre(MDOC_ARGS)
1861 {
1862         struct htmlpair tag;

1864         PAIR_CLASS_INIT(&tag, "var");
1865         print_otag(h, TAG_B, 1, &tag);
1866         return(1);
1867 }


1870 /* ARGSUSED */
1871 static int
1872 mdoc_ap_pre(MDOC_ARGS)
1873 {
1874
1875         h->flags |= HTML_NOSPACE;
1876         print_text(h, "\\(aq");
1877         h->flags |= HTML_NOSPACE;
1878         return(1);
1879 }


1882 /* ARGSUSED */
1883 static int
1884 mdoc_bf_pre(MDOC_ARGS)
1885 {
1886         struct htmlpair  tag[2];
1887         struct roffsu    su;

1889         if (MDOC_HEAD == n->type)
1890                 return(0);
1891         else if (MDOC_BODY != n->type)
1892                 return(1);

1894         if (FONT_Em == n->norm->Bf.font)
1895                 PAIR_CLASS_INIT(&tag[0], "emph");
1896         else if (FONT_Sy == n->norm->Bf.font)
1897                 PAIR_CLASS_INIT(&tag[0], "symb");
1898         else if (FONT_Li == n->norm->Bf.font)
1899                 PAIR_CLASS_INIT(&tag[0], "lit");
1900         else
1901                 PAIR_CLASS_INIT(&tag[0], "none");

1903         /*
1904          * We want this to be inline-formatted, but needs to be div to
1905          * accept block children.
1906          */
1907         bufinit(h);
1908         bufcat_style(h, "display", "inline");
1909         SCALE_HS_INIT(&su, 1);
```



```
1910         /* Needs a left-margin for spacing. */
1911         bufcat_su(h, "margin-left", &su);
1912         PAIR_STYLE_INIT(&tag[1], h);
1913         print_otag(h, TAG_DIV, 2, tag);
1914         return(1);
1915 }


1918 /* ARGSUSED */
1919 static int
1920 mdoc_ms_pre(MDOC_ARGS)
1921 {
1922         struct htmlpair tag;

1924         PAIR_CLASS_INIT(&tag, "symb");
1925         print_otag(h, TAG_SPAN, 1, &tag);
1926         return(1);
1927 }


1930 /* ARGSUSED */
1931 static int
1932 mdoc_igndelim_pre(MDOC_ARGS)
1933 {

1935         h->flags |= HTML_IGNDELIM;
1936         return(1);
1937 }


1940 /* ARGSUSED */
1941 static void
1942 mdoc_pf_post(MDOC_ARGS)
1943 {

1945         h->flags |= HTML_NOSPACE;
1946 }


1949 /* ARGSUSED */
1950 static int
1951 mdoc_rs_pre(MDOC_ARGS)
1952 {
1953         struct htmlpair  tag;

1955         if (MDOC_BLOCK != n->type)
1956                 return(1);

1958         if (n->prev && SEC_SEE_ALSO == n->sec)
1959                 print_otag(h, TAG_P, 0, NULL);

1961         PAIR_CLASS_INIT(&tag, "ref");
1962         print_otag(h, TAG_SPAN, 1, &tag);
1963         return(1);
1964 }



1968 /* ARGSUSED */
1969 static int
1970 mdoc_li_pre(MDOC_ARGS)
1971 {
1972         struct htmlpair tag;

1974         PAIR_CLASS_INIT(&tag, "lit");
1975         print_otag(h, TAG_CODE, 1, &tag);
```

```
1976         return(1);
1977 }


1980 /* ARGSUSED */
1981 static int
1982 mdoc_sy_pre(MDOC_ARGS)
1983 {
1984         struct htmlpair tag;

1986         PAIR_CLASS_INIT(&tag, "symb");
1987         print_otag(h, TAG_SPAN, 1, &tag);
1988         return(1);
1989 }


1992 /* ARGSUSED */
1993 static int
1994 mdoc_bt_pre(MDOC_ARGS)
1995 {

1997         print_text(h, "is currently in beta test.");
1998         return(0);
1999 }


2002 /* ARGSUSED */
2003 static int
2004 mdoc_ud_pre(MDOC_ARGS)
2005 {

2007         print_text(h, "currently under development.");
2008         return(0);
2009 }


2012 /* ARGSUSED */
2013 static int
2014 mdoc_lb_pre(MDOC_ARGS)
2015 {
2016         struct htmlpair tag;

2018         if (SEC_LIBRARY == n->sec && MDOC_LINE & n->flags && n->prev)
2019                 print_otag(h, TAG_BR, 0, NULL);

2021         PAIR_CLASS_INIT(&tag, "lib");
2022         print_otag(h, TAG_SPAN, 1, &tag);
2023         return(1);
2024 }


2027 /* ARGSUSED */
2028 static int
2029 mdoc__x_pre(MDOC_ARGS)
2030 {
2031         struct htmlpair tag[2];
2032         enum htmltag    t;

2034         t = TAG_SPAN;

2036         switch (n->tok) {
2037         case(MDOC__A):
2038                 PAIR_CLASS_INIT(&tag[0], "ref-auth");
2039                 if (n->prev && MDOC__A == n->prev->tok)
2040                         if (NULL == n->next || MDOC__A != n->next->tok)
2041                                 print_text(h, "and");
```



```
2042                 break;
2043         case(MDOC__B):
2044                 PAIR_CLASS_INIT(&tag[0], "ref-book");
2045                 t = TAG_I;
2046                 break;
2047         case(MDOC__C):
2048                 PAIR_CLASS_INIT(&tag[0], "ref-city");
2049                 break;
2050         case(MDOC__D):
2051                 PAIR_CLASS_INIT(&tag[0], "ref-date");
2052                 break;
2053         case(MDOC__I):
2054                 PAIR_CLASS_INIT(&tag[0], "ref-issue");
2055                 t = TAG_I;
2056                 break;
2057         case(MDOC__J):
2058                 PAIR_CLASS_INIT(&tag[0], "ref-jrnl");
2059                 t = TAG_I;
2060                 break;
2061         case(MDOC__N):
2062                 PAIR_CLASS_INIT(&tag[0], "ref-num");
2063                 break;
2064         case(MDOC__O):
2065                 PAIR_CLASS_INIT(&tag[0], "ref-opt");
2066                 break;
2067         case(MDOC__P):
2068                 PAIR_CLASS_INIT(&tag[0], "ref-page");
2069                 break;
2070         case(MDOC__Q):
2071                 PAIR_CLASS_INIT(&tag[0], "ref-corp");
2072                 break;
2073         case(MDOC__R):
2074                 PAIR_CLASS_INIT(&tag[0], "ref-rep");
2075                 break;
2076         case(MDOC__T):
2077                 PAIR_CLASS_INIT(&tag[0], "ref-title");
2078                 break;
2079         case(MDOC__U):
2080                 PAIR_CLASS_INIT(&tag[0], "link-ref");
2081                 break;
2082         case(MDOC__V):
2083                 PAIR_CLASS_INIT(&tag[0], "ref-vol");
2084                 break;
2085         default:
2086                 abort();
2087                 /* NOTREACHED */
2088         }

2090         if (MDOC__U != n->tok) {
2091                 print_otag(h, t, 1, tag);
2092                 return(1);
2093         }

2095         PAIR_HREF_INIT(&tag[1], n->child->string);
2096         print_otag(h, TAG_A, 2, tag);

2098         return(1);
2099 }


2102 /* ARGSUSED */
2103 static void
2104 mdoc__x_post(MDOC_ARGS)
2105 {

2107         if (MDOC__A == n->tok && n->next && MDOC__A == n->next->tok)
```

```
2108                 if (NULL == n->next->next || MDOC__A != n->next->next->tok)
2109                         if (NULL == n->prev || MDOC__A != n->prev->tok)
2110                                 return;
2111
2112         /* TODO: %U */
2113
2114         if (NULL == n->parent || MDOC_Rs != n->parent->tok)
2115                 return;
2116
2117         h->flags |= HTML_NOSPACE;
2118         print_text(h, n->next ? "," : ".");
2119 }
2120
2121
2122 /* ARGSUSED */
2123 static int
2124 mdoc_bk_pre(MDOC_ARGS)
2125 {
2126
2127         switch (n->type) {
2128         case (MDOC_BLOCK):
2129                 break;
2130         case (MDOC_HEAD):
2131                 return(0);
2132         case (MDOC_BODY):
2133                 if (n->parent->args || 0 == n->prev->nchild)
2134                         h->flags |= HTML_PREKEEP;
2135                 break;
2136         default:
2137                 abort();
2138                 /* NOTREACHED */
2139         }
2140
2141         return(1);
2142 }
2143
2144
2145 /* ARGSUSED */
2146 static void
2147 mdoc_bk_post(MDOC_ARGS)
2148 {
2149
2150         if (MDOC_BODY == n->type)
2151                 h->flags &= ~(HTML_KEEP | HTML_PREKEEP);
2152 }
2153
2154
2155 /* ARGSUSED */
2156 static int
2157 mdoc_quote_pre(MDOC_ARGS)
2158 {
2159         struct htmlpair tag;
2160
2161         if (MDOC_BODY != n->type)
2162                 return(1);
2163
2164         switch (n->tok) {
2165         case (MDOC_Ao):
2166                 /* FALLTHROUGH */
2167         case (MDOC_Aq):
2168                 print_text(h, "\\(la");
2169                 break;
2170         case (MDOC_Bro):
2171                 /* FALLTHROUGH */
2172         case (MDOC_Brq):
2173                 print_text(h, "\\(lC");
```



```
2174                 break;
2175         case (MDOC_Bo):
2176                 /* FALLTHROUGH */
2177         case (MDOC_Bq):
2178                 print_text(h, "\\(lB");
2179                 break;
2180         case (MDOC_Oo):
2181                 /* FALLTHROUGH */
2182         case (MDOC_Op):
2183                 print_text(h, "\\(lB");
2184                 h->flags |= HTML_NOSPACE;
2185                 PAIR_CLASS_INIT(&tag, "opt");
2186                 print_otag(h, TAG_SPAN, 1, &tag);
2187                 break;
2188         case (MDOC_Eo):
2189                 break;
2190         case (MDOC_Do):
2191                 /* FALLTHROUGH */
2192         case (MDOC_Dq):
2193                 /* FALLTHROUGH */
2194         case (MDOC_Qo):
2195                 /* FALLTHROUGH */
2196         case (MDOC_Qq):
2197                 print_text(h, "\\(lq");
2198                 break;
2199         case (MDOC_Po):
2200                 /* FALLTHROUGH */
2201         case (MDOC_Pq):
2202                 print_text(h, "(");
2203                 break;
2204         case (MDOC_Ql):
2205                 print_text(h, "\\(oq");
2206                 h->flags |= HTML_NOSPACE;
2207                 PAIR_CLASS_INIT(&tag, "lit");
2208                 print_otag(h, TAG_CODE, 1, &tag);
2209                 break;
2210         case (MDOC_So):
2211                 /* FALLTHROUGH */
2212         case (MDOC_Sq):
2213                 print_text(h, "\\(oq");
2214                 break;
2215         default:
2216                 abort();
2217                 /* NOTREACHED */
2218         }
2219
2220         h->flags |= HTML_NOSPACE;
2221         return(1);
2222 }
2223
2224
2225 /* ARGSUSED */
2226 static void
2227 mdoc_quote_post(MDOC_ARGS)
2228 {
2229
2230         if (MDOC_BODY != n->type)
2231                 return;
2232
2233         h->flags |= HTML_NOSPACE;
2234
2235         switch (n->tok) {
2236         case (MDOC_Ao):
2237                 /* FALLTHROUGH */
2238         case (MDOC_Aq):
2239                 print_text(h, "\\(ra");
```

```
2240                    break;
2241            case (MDOC_Bro):
2242                    /* FALLTHROUGH */
2243            case (MDOC_Brq):
2244                    print_text(h, "\\(rC");
2245                    break;
2246            case (MDOC_Oo):
2247                    /* FALLTHROUGH */
2248            case (MDOC_Op):
2249                    /* FALLTHROUGH */
2250            case (MDOC_Bo):
2251                    /* FALLTHROUGH */
2252            case (MDOC_Bq):
2253                    print_text(h, "\\(rB");
2254                    break;
2255            case (MDOC_Eo):
2256                    break;
2257            case (MDOC_Qo):
2258                    /* FALLTHROUGH */
2259            case (MDOC_Qq):
2260                    /* FALLTHROUGH */
2261            case (MDOC_Do):
2262                    /* FALLTHROUGH */
2263            case (MDOC_Dq):
2264                    print_text(h, "\\(rq");
2265                    break;
2266            case (MDOC_Po):
2267                    /* FALLTHROUGH */
2268            case (MDOC_Pq):
2269                    print_text(h, ")");
2270                    break;
2271            case (MDOC_Ql):
2272                    /* FALLTHROUGH */
2273            case (MDOC_So):
2274                    /* FALLTHROUGH */
2275            case (MDOC_Sq):
2276                    print_text(h, "\\(aq");
2277                    break;
2278            default:
2279                    abort();
2280                    /* NOTREACHED */
2281            }
2282 }
```

```
**********************************************************
   42928 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/mdoc_macro.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_macro.c,v 1.115 2012/01/05 00:43:51 schwarze Exp $ */
/*
 * 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);
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 }, /* Ap */
        { in_line_eoln, MDOC_PROLOGUE }, /* Dd */
        { in_line_eoln, MDOC_PROLOGUE }, /* Dt */
        { in_line_eoln, MDOC_PROLOGUE }, /* Os */
        { blk_full, MDOC_PARSED }, /* Sh */
        { blk_full, MDOC_PARSED }, /* Ss */
        { in_line_eoln, 0 }, /* Pp */
        { blk_part_imp, MDOC_PARSED }, /* D1 */
        { blk_part_imp, MDOC_PARSED }, /* Dl */
        { blk_full, MDOC_EXPLICIT }, /* Bd */
        { blk_exp_close, MDOC_EXPLICIT }, /* Ed */
        { blk_full, MDOC_EXPLICIT }, /* Bl */
        { blk_exp_close, MDOC_EXPLICIT }, /* El */
        { blk_full, MDOC_PARSED }, /* It */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ad */
        { 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 }, /* 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 */
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* St */
        { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Va */
        { ctx_synopsis, MDOC_CALLABLE | MDOC_PARSED }, /* Vt */
        { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Xr */
        { in_line_eoln, 0 }, /* %A */
        { in_line_eoln, 0 }, /* %B */
        { in_line_eoln, 0 }, /* %D */
        { in_line_eoln, 0 }, /* %I */
        { in_line_eoln, 0 }, /* %J */
        { in_line_eoln, 0 }, /* %N */
        { in_line_eoln, 0 }, /* %O */
        { in_line_eoln, 0 }, /* %P */
        { in_line_eoln, 0 }, /* %R */
        { in_line_eoln, 0 }, /* %T */
        { in_line_eoln, 0 }, /* %V */
        { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Ac */
        { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Ao */
        { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Aq */
```

```
128         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* At */
129         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Bc */
130         { blk_full, MDOC_EXPLICIT }, /* Bf */
131         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bo */
132         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Bq */
133         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bsx */
134         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Bx */
135         { in_line_eoln, 0 }, /* Db */
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 */
142         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Eo */
143         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Fx */
144         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Ms */
145         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* No */
146         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Ns */
147         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Nx */
148         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ox */
149         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Pc */
150         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED | MDOC_IGNDELIM }, /* Pf */
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 */
158         { blk_full, MDOC_EXPLICIT }, /* Rs */
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 */
162         { in_line_eoln, 0 }, /* Sm */
163         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sx */
164         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Sy */
165         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Tn */
166         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Ux */
167         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Xc */
168         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Xo */
169         { blk_full, MDOC_EXPLICIT | MDOC_CALLABLE }, /* Fo */
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 */
173         { blk_full, MDOC_EXPLICIT }, /* Bk */
174         { blk_exp_close, MDOC_EXPLICIT }, /* Ek */
175         { in_line_eoln, 0 }, /* Bt */
176         { in_line_eoln, 0 }, /* Hf */
177         { obsolete, 0 }, /* Fr */
178         { in_line_eoln, 0 }, /* Ud */
179         { in_line, 0 }, /* Lb */
180         { in_line_eoln, 0 }, /* Lp */
181         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Lk */
182         { in_line, MDOC_CALLABLE | MDOC_PARSED }, /* Mt */
183         { blk_part_imp, MDOC_CALLABLE | MDOC_PARSED }, /* Brq */
184         { blk_part_exp, MDOC_CALLABLE | MDOC_PARSED | MDOC_EXPLICIT }, /* Bro */
185         { blk_exp_close, MDOC_EXPLICIT | MDOC_CALLABLE | MDOC_PARSED }, /* Brc *
186         { in_line_eoln, 0 }, /* %C */
187         { obsolete, 0 }, /* Es */
188         { obsolete, 0 }, /* En */
189         { in_line_argn, MDOC_CALLABLE | MDOC_PARSED }, /* Dx */
190         { in_line_eoln, 0 }, /* %Q */
191         { in_line_eoln, 0 }, /* br */
192         { in_line_eoln, 0 }, /* sp */
193         { in_line_eoln, 0 }, /* %U */
```



```
194         { phrase_ta, MDOC_CALLABLE | MDOC_PARSED }, /* Ta */
195 };

197 const   struct mdoc_macro * const mdoc_macros = __mdoc_macros;


200 /*
201  * This is called at the end of parsing.  It must traverse up the tree,
202  * closing out open [implicit] scopes.  Obviously, open explicit scopes
203  * are errors.
204  */
205 int
206 mdoc_macroend(struct mdoc *m)
207 {
208         struct mdoc_node *n;

210         /* Scan for open explicit scopes. */

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

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

219         /* Rewind to the first. */

221         return(rew_last(m, m->first));
222 }


225 /*
226  * Look up a macro from within a subsequent context.
227  */
228 static enum mdoct
229 lookup(enum mdoct from, const char *p)
230 {

232         if ( ! (MDOC_PARSED & mdoc_macros[from].flags))
233                 return(MDOC_MAX);
234         return(lookup_raw(p));
235 }


238 /*
239  * Lookup a macro following the initial line macro.
240  */
241 static enum mdoct
242 lookup_raw(const char *p)
243 {
244         enum mdoct       res;

246         if (MDOC_MAX == (res = mdoc_hash_find(p)))
247                 return(MDOC_MAX);
248         if (MDOC_CALLABLE & mdoc_macros[res].flags)
249                 return(res);
250         return(MDOC_MAX);
251 }


254 static int
255 rew_last(struct mdoc *mdoc, const struct mdoc_node *to)
256 {
257         struct mdoc_node *n, *np;

259         assert(to);
```

```
260         mdoc->next = MDOC_NEXT_SIBLING;

262         /* LINTED */
263         while (mdoc->last != to) {
264                 /*
265                  * Save the parent here, because we may delete the
266                  * m->last node in the post-validation phase and reset
267                  * it to m->last->parent, causing a step in the closing
268                  * out to be lost.
269                  */
270                 np = mdoc->last->parent;
271                 if ( ! mdoc_valid_post(mdoc))
272                         return(0);
273                 n = mdoc->last;
274                 mdoc->last = np;
275                 assert(mdoc->last);
276                 mdoc->last->last = n;
277         }

279         return(mdoc_valid_post(mdoc));
280 }


283 /*
284  * For a block closing macro, return the corresponding opening one.
285  * Otherwise, return the macro itself.
286  */
287 static enum mdoct
288 rew_alt(enum mdoct tok)
289 {
290         switch (tok) {
291         case (MDOC_Ac):
292                 return(MDOC_Ao);
293         case (MDOC_Bc):
294                 return(MDOC_Bo);
295         case (MDOC_Brc):
296                 return(MDOC_Bro);
297         case (MDOC_Dc):
298                 return(MDOC_Do);
299         case (MDOC_Ec):
300                 return(MDOC_Eo);
301         case (MDOC_Ed):
302                 return(MDOC_Bd);
303         case (MDOC_Ef):
304                 return(MDOC_Bf);
305         case (MDOC_Ek):
306                 return(MDOC_Bk);
307         case (MDOC_El):
308                 return(MDOC_Bl);
309         case (MDOC_Fc):
310                 return(MDOC_Fo);
311         case (MDOC_Oc):
312                 return(MDOC_Oo);
313         case (MDOC_Pc):
314                 return(MDOC_Po);
315         case (MDOC_Qc):
316                 return(MDOC_Qo);
317         case (MDOC_Re):
318                 return(MDOC_Rs);
319         case (MDOC_Sc):
320                 return(MDOC_So);
321         case (MDOC_Xc):
322                 return(MDOC_Xo);
323         default:
324                 return(tok);
325         }
```



```
326         /* NOTREACHED */
327 }


330 /*
331  * Rewinding to tok, how do we have to handle *p?
332  * REWIND_NONE: *p would delimit tok, but no tok scope is open
333  *   inside *p, so there is no need to rewind anything at all.
334  * REWIND_THIS: *p matches tok, so rewind *p and nothing else.
335  * REWIND_MORE: *p is implicit, rewind it and keep searching for tok.
336  * REWIND_FORCE: *p is explicit, but tok is full, force rewinding *p.
337  * REWIND_LATER: *p is explicit and still open, postpone rewinding.
338  * REWIND_ERROR: No tok block is open at all.
339  */
340 static enum rew
341 rew_dohalt(enum mdoct tok, enum mdoc_type type,
342                 const struct mdoc_node *p)
343 {

345         /*
346          * No matching token, no delimiting block, no broken block.
347          * This can happen when full implicit macros are called for
348          * the first time but try to rewind their previous
349          * instance anyway.
350          */
351         if (MDOC_ROOT == p->type)
352                 return(MDOC_BLOCK == type &&
353                     MDOC_EXPLICIT & mdoc_macros[tok].flags ?
354                     REWIND_ERROR : REWIND_NONE);

356         /*
357          * When starting to rewind, skip plain text
358          * and nodes that have already been rewound.
359          */
360         if (MDOC_TEXT == p->type || MDOC_VALID & p->flags)
361                 return(REWIND_MORE);

363         /*
364          * The easiest case:  Found a matching token.
365          * This applies to both blocks and elements.
366          */
367         tok = rew_alt(tok);
368         if (tok == p->tok)
369                 return(p->end ? REWIND_NONE :
370                     type == p->type ? REWIND_THIS : REWIND_MORE);

372         /*
373          * While elements do require rewinding for themselves,
374          * they never affect rewinding of other nodes.
375          */
376         if (MDOC_ELEM == p->type)
377                 return(REWIND_MORE);

379         /*
380          * Blocks delimited by our target token get REWIND_MORE.
381          * Blocks delimiting our target token get REWIND_NONE.
382          */
383         switch (tok) {
384         case (MDOC_Bl):
385                 if (MDOC_It == p->tok)
386                         return(REWIND_MORE);
387                 break;
388         case (MDOC_It):
389                 if (MDOC_BODY == p->type && MDOC_Bl == p->tok)
390                         return(REWIND_NONE);
391                 break;
```

```
392         /*
393          * XXX Badly nested block handling still fails badly
394          * when one block is breaking two blocks of the same type.
395          * This is an incomplete and extremely ugly workaround,
396          * required to let the OpenBSD tree build.
397          */
398         case (MDOC_Oo):
399                 if (MDOC_Op == p->tok)
400                         return(REWIND_MORE);
401                 break;
402         case (MDOC_Nm):
403                 return(REWIND_NONE);
404         case (MDOC_Nd):
405                 /* FALLTHROUGH */
406         case (MDOC_Ss):
407                 if (MDOC_BODY == p->type && MDOC_Sh == p->tok)
408                         return(REWIND_NONE);
409                 /* FALLTHROUGH */
410         case (MDOC_Sh):
411                 if (MDOC_Nd == p->tok || MDOC_Ss == p->tok ||
412                     MDOC_Sh == p->tok)
413                         return(REWIND_MORE);
414                 break;
415         default:
416                 break;
417         }

419         /*
420          * Default block rewinding rules.
421          * In particular, always skip block end markers,
422          * and let all blocks rewind Nm children.
423          */
424         if (ENDBODY_NOT != p->end || MDOC_Nm == p->tok ||
425             (MDOC_BLOCK == p->type &&
426             ! (MDOC_EXPLICIT & mdoc_macros[tok].flags)))
427                 return(REWIND_MORE);

429         /*
430          * By default, closing out full blocks
431          * forces closing of broken explicit blocks,
432          * while closing out partial blocks
433          * allows delayed rewinding by default.
434          */
435         return (&blk_full == mdoc_macros[tok].fp ?
436             REWIND_FORCE : REWIND_LATER);
437 }


440 static int
441 rew_elem(struct mdoc *mdoc, enum mdoct tok)
442 {
443         struct mdoc_node *n;

445         n = mdoc->last;
446         if (MDOC_ELEM != n->type)
447                 n = n->parent;
448         assert(MDOC_ELEM == n->type);
449         assert(tok == n->tok);

451         return(rew_last(mdoc, n));
452 }


455 /*
456  * We are trying to close a block identified by tok,
457  * but the child block *broken is still open.
```



```
458  * Thus, postpone closing the tok block
459  * until the rew_sub call closing *broken.
460  */
461 static int
462 make_pending(struct mdoc_node *broken, enum mdoct tok,
463                 struct mdoc *m, int line, int ppos)
464 {
465         struct mdoc_node *breaker;

467         /*
468          * Iterate backwards, searching for the block matching tok,
469          * that is, the block breaking the *broken block.
470          */
471         for (breaker = broken->parent; breaker; breaker = breaker->parent) {

473                 /*
474                  * If the *broken block had already been broken before
475                  * and we encounter its breaker, make the tok block
476                  * pending on the inner breaker.
477                  * Graphically, "[A breaker=[B broken=[C->B B] tok=A] C]"
478                  * becomes "[A broken=[B [C->B B] tok=A] C]"
479                  * and finally "[A [B->A [C->B B] A] C]".
480                  */
481                 if (breaker == broken->pending) {
482                         broken = breaker;
483                         continue;
484                 }

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

491                 /*
492                  * Found the breaker.
493                  * If another, outer breaker is already pending on
494                  * the *broken block, we must not clobber the link
495                  * to the outer breaker, but make it pending on the
496                  * new, now inner breaker.
497                  * Graphically, "[A breaker=[B broken=[C->A A] tok=B] C]"
498                  * becomes "[A breaker=[B->A broken=[C A] tok=B] C]"
499                  * and finally "[A [B->A [C->B A] B] C]".
500                  */
501                 if (broken->pending) {
502                         struct mdoc_node *taker;

504                         /*
505                          * If the breaker had also been broken before,
506                          * it cannot take on the outer breaker itself,
507                          * but must hand it on to its own breakers.
508                          * Graphically, this is the following situation:
509                          * "[A [B breaker=[C->B B] broken=[D->A A] tok=C] D]"
510                          * "[A taker=[B->A breaker=[C->B B] [D->C A] C] D]"
511                          */
512                         taker = breaker;
513                         while (taker->pending)
514                                 taker = taker->pending;
515                         taker->pending = broken->pending;
516                 }
517                 broken->pending = breaker;
518                 mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
519                                 "%s breaks %s", mdoc_macronames[tok],
520                                 mdoc_macronames[broken->tok]);
521                 return(1);
522         }
```

```
524         /*
525          * Found no matching block for tok.
526          * Are you trying to close a block that is not open?
527          */
528         return(0);
529 }


532 static int
533 rew_sub(enum mdoc_type t, struct mdoc *m,
534                 enum mdoct tok, int line, int ppos)
535 {
536         struct mdoc_node *n;

538         n = m->last;
539         while (n) {
540                 switch (rew_dohalt(tok, t, n)) {
541                 case (REWIND_NONE):
542                         return(1);
543                 case (REWIND_THIS):
544                         break;
545                 case (REWIND_FORCE):
546                         mandoc_vmsg(MANDOCERR_SCOPEBROKEN, m->parse,
547                                         line, ppos, "%s breaks %s",
548                                         mdoc_macronames[tok],
549                                         mdoc_macronames[n->tok]);
550                         /* FALLTHROUGH */
551                 case (REWIND_MORE):
552                         n = n->parent;
553                         continue;
554                 case (REWIND_LATER):
555                         if (make_pending(n, tok, m, line, ppos) ||
556                             MDOC_BLOCK != t)
557                                 return(1);
558                         /* FALLTHROUGH */
559                 case (REWIND_ERROR):
560                         mdoc_pmsg(m, line, ppos, MANDOCERR_NOSCOPE);
561                         return(1);
562                 }
563                 break;
564         }

566         assert(n);
567         if ( ! rew_last(m, n))
568                 return(0);

570         /*
571          * The current block extends an enclosing block.
572          * Now that the current block ends, close the enclosing block, too.
573          */
574         while (NULL != (n = n->pending)) {
575                 if ( ! rew_last(m, n))
576                         return(0);
577                 if (MDOC_HEAD == n->type &&
578                     ! mdoc_body_alloc(m, n->line, n->pos, n->tok))
579                         return(0);
580         }

582         return(1);
583 }

585 /*
586  * Allocate a word and check whether it's punctuation or not.
587  * Punctuation consists of those tokens found in mdoc_isdelim().
588  */
589 static int
```



```
590 dword(struct mdoc *m, int line,
591                 int col, const char *p, enum mdelim d)
592 {
593
594         if (DELIM_MAX == d)
595                 d = mdoc_isdelim(p);

597         if ( ! mdoc_word_alloc(m, line, col, p))
598                 return(0);

600         if (DELIM_OPEN == d)
601                 m->last->flags |= MDOC_DELIMO;

603         /*
604          * Closing delimiters only suppress the preceding space
605          * when they follow something, not when they start a new
606          * block or element, and not when they follow 'No'.
607          *
608          * XXX  Explicitly special-casing MDOC_No here feels
609          *      like a layering violation.  Find a better way
610          *      and solve this in the code related to 'No'!
611          */

613         else if (DELIM_CLOSE == d && m->last->prev &&
614                         m->last->prev->tok != MDOC_No)
615                 m->last->flags |= MDOC_DELIMC;

617         return(1);
618 }

620 static int
621 append_delims(struct mdoc *m, int line, int *pos, char *buf)
622 {
623         int              la;
624         enum margserr    ac;
625         char            *p;

627         if ('\0' == buf[*pos])
628                 return(1);

630         for (;;) {
631                 la = *pos;
632                 ac = mdoc_zargs(m, line, pos, buf, &p);

634                 if (ARGS_ERROR == ac)
635                         return(0);
636                 else if (ARGS_EOLN == ac)
637                         break;

639                 dword(m, line, la, p, DELIM_MAX);

641                 /*
642                  * If we encounter end-of-sentence symbols, then trigger
643                  * the double-space.
644                  *
645                  * XXX: it's easy to allow this to propagate outward to
646                  * the last symbol, such that '. )' will cause the
647                  * correct double-spacing.  However, (1) groff isn't
648                  * smart enough to do this and (2) it would require
649                  * knowing which symbols break this behaviour, for
650                  * example, '.  ;' shouldn't propagate the double-space.
651                  */
652                 if (mandoc_eos(p, strlen(p), 0))
653                         m->last->flags |= MDOC_EOS;
654         }
```

```
656         return(1);
657 }


660 /*
661  * Close out block partial/full explicit.
662  */
663 static int
664 blk_exp_close(MACRO_PROT_ARGS)
665 {
666         struct mdoc_node *body;         /* Our own body. */
667         struct mdoc_node *later;        /* A sub-block starting later. */
668         struct mdoc_node *n;            /* For searching backwards. */

670         int              j, lastarg, maxargs, flushed, nl;
671         enum margserr    ac;
672         enum mdoct       atok, ntok;
673         char            *p;

675         nl = MDOC_NEWLINE & m->flags;

677         switch (tok) {
678         case (MDOC_Ec):
679                 maxargs = 1;
680                 break;
681         default:
682                 maxargs = 0;
683                 break;
684         }

686         /*
687          * Search backwards for beginnings of blocks,
688          * both of our own and of pending sub-blocks.
689          */
690         atok = rew_alt(tok);
691         body = later = NULL;
692         for (n = m->last; n; n = n->parent) {
693                 if (MDOC_VALID & n->flags)
694                         continue;

696                 /* Remember the start of our own body. */
697                 if (MDOC_BODY == n->type && atok == n->tok) {
698                         if (ENDBODY_NOT == n->end)
699                                 body = n;
700                         continue;
701                 }

703                 if (MDOC_BLOCK != n->type || MDOC_Nm == n->tok)
704                         continue;
705                 if (atok == n->tok) {
706                         assert(body);

708                         /*
709                          * Found the start of our own block.
710                          * When there is no pending sub block,
711                          * just proceed to closing out.
712                          */
713                         if (NULL == later)
714                                 break;

716                         /*
717                          * When there is a pending sub block,
718                          * postpone closing out the current block
719                          * until the rew_sub() closing out the sub-block.
720                          */
721                         make_pending(later, tok, m, line, ppos);
```



```
723                         /*
724                          * Mark the place where the formatting - but not
725                          * the scope - of the current block ends.
726                          */
727                         if ( ! mdoc_endbody_alloc(m, line, ppos,
728                             atok, body, ENDBODY_SPACE))
729                                 return(0);
730                         break;
731                 }

733                 /*
734                  * When finding an open sub block, remember the last
735                  * open explicit block, or, in case there are only
736                  * implicit ones, the first open implicit block.
737                  */
738                 if (later &&
739                     MDOC_EXPLICIT & mdoc_macros[later->tok].flags)
740                         continue;
741                 if (MDOC_CALLABLE & mdoc_macros[n->tok].flags)
742                         later = n;
743         }

745         if ( ! (MDOC_CALLABLE & mdoc_macros[tok].flags)) {
746                 /* FIXME: do this in validate */
747                 if (buf[*pos])
748                         mdoc_pmsg(m, line, ppos, MANDOCERR_ARGSLOST);

750                 if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
751                         return(0);
752                 return(rew_sub(MDOC_BLOCK, m, tok, line, ppos));
753         }

755         if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
756                 return(0);

758         if (NULL == later && maxargs > 0)
759                 if ( ! mdoc_tail_alloc(m, line, ppos, rew_alt(tok)))
760                         return(0);

762         for (flushed = j = 0; ; j++) {
763                 lastarg = *pos;

765                 if (j == maxargs && ! flushed) {
766                         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
767                                 return(0);
768                         flushed = 1;
769                 }

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

773                 if (ARGS_ERROR == ac)
774                         return(0);
775                 if (ARGS_PUNCT == ac)
776                         break;
777                 if (ARGS_EOLN == ac)
778                         break;

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

782                 if (MDOC_MAX == ntok) {
783                         if ( ! dword(m, line, lastarg, p, DELIM_MAX))
784                                 return(0);
785                         continue;
786                 }
```

```
788                 if ( ! flushed) {
789                         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
790                                 return(0);
791                         flushed = 1;
792                 }
793                 if ( ! mdoc_macro(m, ntok, line, lastarg, pos, buf))
794                         return(0);
795                 break;
796         }

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

801         if ( ! nl)
802                 return(1);
803         return(append_delims(m, line, pos, buf));
804 }


807 static int
808 in_line(MACRO_PROT_ARGS)
809 {
810         int              la, scope, cnt, nc, nl;
811         enum margverr    av;
812         enum mdoct       ntok;
813         enum margserr    ac;
814         enum mdelim      d;
815         struct mdoc_arg *arg;
816         char            *p;

818         nl = MDOC_NEWLINE & m->flags;

820         /*
821          * Whether we allow ignored elements (those without content,
822          * usually because of reserved words) to squeak by.
823          */

825         switch (tok) {
826         case (MDOC_An):
827                 /* FALLTHROUGH */
828         case (MDOC_Ar):
829                 /* FALLTHROUGH */
830         case (MDOC_Fl):
831                 /* FALLTHROUGH */
832         case (MDOC_Mt):
833                 /* FALLTHROUGH */
834         case (MDOC_Nm):
835                 /* FALLTHROUGH */
836         case (MDOC_Pa):
837                 nc = 1;
838                 break;
839         default:
840                 nc = 0;
841                 break;
842         }

844         for (arg = NULL;; ) {
845                 la = *pos;
846                 av = mdoc_argv(m, line, tok, &arg, pos, buf);

848                 if (ARGV_WORD == av) {
849                         *pos = la;
850                         break;
851                 }
852                 if (ARGV_EOLN == av)
853                         break;
```



```
854                 if (ARGV_ARG == av)
855                         continue;

857                 mdoc_argv_free(arg);
858                 return(0);
859         }

861         for (cnt = scope = 0;; ) {
862                 la = *pos;
863                 ac = mdoc_args(m, line, pos, buf, tok, &p);

865                 if (ARGS_ERROR == ac)
866                         return(0);
867                 if (ARGS_EOLN == ac)
868                         break;
869                 if (ARGS_PUNCT == ac)
870                         break;

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

874                 /*
875                  * In this case, we've located a submacro and must
876                  * execute it.  Close out scope, if open.  If no
877                  * elements have been generated, either create one (nc)
878                  * or raise a warning.
879                  */

881                 if (MDOC_MAX != ntok) {
882                         if (scope && ! rew_elem(m, tok))
883                                 return(0);
884                         if (nc && 0 == cnt) {
885                                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
886                                         return(0);
887                                 if ( ! rew_last(m, m->last))
888                                         return(0);
889                         } else if ( ! nc && 0 == cnt) {
890                                 mdoc_argv_free(arg);
891                                 mdoc_pmsg(m, line, ppos, MANDOCERR_MACROEMPTY);
892                         }

894                         if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
895                                 return(0);
896                         if ( ! nl)
897                                 return(1);
898                         return(append_delims(m, line, pos, buf));
899                 }

901                 /*
902                  * Non-quote-enclosed punctuation.  Set up our scope, if
903                  * a word; rewind the scope, if a delimiter; then append
904                  * the word.
905                  */

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

909                 if (DELIM_NONE != d) {
910                         /*
911                          * If we encounter closing punctuation, no word
912                          * has been omitted, no scope is open, and we're
913                          * allowed to have an empty element, then start
914                          * a new scope.  'Ar', 'Fl', and 'Li', only do
915                          * this once per invocation.  There may be more
916                          * of these (all of them?).
917                          */
918                         if (0 == cnt && (nc || MDOC_Li == tok) &&
919                                         DELIM_CLOSE == d && ! scope) {
```

```
920                                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
921                                         return(0);
922                                 if (MDOC_Ar == tok || MDOC_Li == tok ||
923                                                 MDOC_Fl == tok)
924                                         cnt++;
925                                 scope = 1;
926                         }
927                         /*
928                          * Close out our scope, if one is open, before
929                          * any punctuation.
930                          */
931                         if (scope && ! rew_elem(m, tok))
932                                 return(0);
933                         scope = 0;
934                 } else if ( ! scope) {
935                         if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
936                                 return(0);
937                         scope = 1;
938                 }
939
940                 if (DELIM_NONE == d)
941                         cnt++;
942
943                 if ( ! dword(m, line, la, p, d))
944                         return(0);
945
946                 /*
947                  * ‘Fl’ macros have their scope re-opened with each new
948                  * word so that the ‘-’ can be added to each one without
949                  * having to parse out spaces.
950                  */
951                 if (scope && MDOC_Fl == tok) {
952                         if ( ! rew_elem(m, tok))
953                                 return(0);
954                         scope = 0;
955                 }
956         }
957
958         if (scope && ! rew_elem(m, tok))
959                 return(0);
960
961         /*
962          * If no elements have been collected and we’re allowed to have
963          * empties (nc), open a scope and close it out.  Otherwise,
964          * raise a warning.
965          */
966
967         if (nc && 0 == cnt) {
968                 if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
969                         return(0);
970                 if ( ! rew_last(m, m->last))
971                         return(0);
972         } else if ( ! nc && 0 == cnt) {
973                 mdoc_argv_free(arg);
974                 mdoc_pmsg(m, line, ppos, MANDOCERR_MACROEMPTY);
975         }
976
977         if ( ! nl)
978                 return(1);
979         return(append_delims(m, line, pos, buf));
980 }
981
982
983 static int
984 blk_full(MACRO_PROT_ARGS)
985 {
```



```
986         int               la, nl, nparsed;
987         struct mdoc_arg  *arg;
988         struct mdoc_node *head; /* save of head macro */
989         struct mdoc_node *body; /* save of body macro */
990         struct mdoc_node *n;
991         enum mdoc_type    mtt;
992         enum mdoct        ntok;
993         enum margserr     ac, lac;
994         enum margverr     av;
995         char             *p;
996
997         nl = MDOC_NEWLINE & m->flags;
998
999         /* Close out prior implicit scope. */
1000
1001        if ( ! (MDOC_EXPLICIT & mdoc_macros[tok].flags)) {
1002                if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1003                        return(0);
1004                if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1005                        return(0);
1006        }
1007
1008        /*
1009         * This routine accommodates implicitly- and explicitly-scoped
1010         * macro openings.  Implicit ones first close out prior scope
1011         * (seen above).  Delay opening the head until necessary to
1012         * allow leading punctuation to print.  Special consideration
1013         * for ‘It -column’, which has phrase-part syntax instead of
1014         * regular child nodes.
1015         */
1016
1017        for (arg = NULL;; ) {
1018                la = *pos;
1019                av = mdoc_argv(m, line, tok, &arg, pos, buf);
1020
1021                if (ARGV_WORD == av) {
1022                        *pos = la;
1023                        break;
1024                }
1025
1026                if (ARGV_EOLN == av)
1027                        break;
1028                if (ARGV_ARG == av)
1029                        continue;
1030
1031                mdoc_argv_free(arg);
1032                return(0);
1033        }
1034
1035        if ( ! mdoc_block_alloc(m, line, ppos, tok, arg))
1036                return(0);
1037
1038        head = body = NULL;
1039
1040        /*
1041         * Exception: Heads of ‘It’ macros in ‘-diag’ lists are not
1042         * parsed, even though ‘It’ macros in general are parsed.
1043         */
1044        nparsed = MDOC_It == tok &&
1045                MDOC_Bl == m->last->parent->tok &&
1046                LIST_diag == m->last->parent->norm->Bl.type;
1047
1048        /*
1049         * The ‘Nd’ macro has all arguments in its body: it’s a hybrid
1050         * of block partial-explicit and full-implicit.  Stupid.
1051         */
```

```
1053         if (MDOC_Nd == tok) {
1054                 if ( ! mdoc_head_alloc(m, line, ppos, tok))
1055                         return(0);
1056                 head = m->last;
1057                 if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1058                         return(0);
1059                 if ( ! mdoc_body_alloc(m, line, ppos, tok))
1060                         return(0);
1061                 body = m->last;
1062         }

1064         ac = ARGS_ERROR;

1066         for ( ; ; ) {
1067                 la = *pos;
1068                 /* Initialise last-phrase-type with ARGS_PEND. */
1069                 lac = ARGS_ERROR == ac ? ARGS_PEND : ac;
1070                 ac = mdoc_args(m, line, pos, buf, tok, &p);

1072                 if (ARGS_PUNCT == ac)
1073                         break;

1075                 if (ARGS_ERROR == ac)
1076                         return(0);

1078                 if (ARGS_EOLN == ac) {
1079                         if (ARGS_PPHRASE != lac && ARGS_PHRASE != lac)
1080                                 break;
1081                         /*
1082                          * This is necessary: if the last token on a
1083                          * line is a 'Ta' or tab, then we'll get
1084                          * ARGS_EOLN, so we must be smart enough to
1085                          * reopen our scope if the last parse was a
1086                          * phrase or partial phrase.
1087                          */
1088                         if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1089                                 return(0);
1090                         if ( ! mdoc_body_alloc(m, line, ppos, tok))
1091                                 return(0);
1092                         body = m->last;
1093                         break;
1094                 }

1096                 /*
1097                  * Emit leading punctuation (i.e., punctuation before
1098                  * the MDOC_HEAD) for non-phrase types.
1099                  */

1101                 if (NULL == head &&
1102                                 ARGS_PEND != ac &&
1103                                 ARGS_PHRASE != ac &&
1104                                 ARGS_PPHRASE != ac &&
1105                                 ARGS_QWORD != ac &&
1106                                 DELIM_OPEN == mdoc_isdelim(p)) {
1107                         if ( ! dword(m, line, la, p, DELIM_OPEN))
1108                                 return(0);
1109                         continue;
1110                 }

1112                 /* Open a head if one hasn't been opened. */

1114                 if (NULL == head) {
1115                         if ( ! mdoc_head_alloc(m, line, ppos, tok))
1116                                 return(0);
1117                         head = m->last;
```



```
1118                 }

1120                 if (ARGS_PHRASE == ac ||
1121                                 ARGS_PEND == ac ||
1122                                 ARGS_PPHRASE == ac) {
1123                         /*
1124                          * If we haven't opened a body yet, rewind the
1125                          * head; if we have, rewind that instead.
1126                          */

1128                         mtt = body ? MDOC_BODY : MDOC_HEAD;
1129                         if ( ! rew_sub(mtt, m, tok, line, ppos))
1130                                 return(0);
1131
1132                         /* Then allocate our body context. */

1134                         if ( ! mdoc_body_alloc(m, line, ppos, tok))
1135                                 return(0);
1136                         body = m->last;

1138                         /*
1139                          * Process phrases: set whether we're in a
1140                          * partial-phrase (this effects line handling)
1141                          * then call down into the phrase parser.
1142                          */

1144                         if (ARGS_PPHRASE == ac)
1145                                 m->flags |= MDOC_PPHRASE;
1146                         if (ARGS_PEND == ac && ARGS_PPHRASE == lac)
1147                                 m->flags |= MDOC_PPHRASE;

1149                         if ( ! phrase(m, line, la, buf))
1150                                 return(0);

1152                         m->flags &= ~MDOC_PPHRASE;
1153                         continue;
1154                 }

1156                 ntok = nparsed || ARGS_QWORD == ac ?
1157                         MDOC_MAX : lookup(tok, p);

1159                 if (MDOC_MAX == ntok) {
1160                         if ( ! dword(m, line, la, p, DELIM_MAX))
1161                                 return(0);
1162                         continue;
1163                 }

1165                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1166                         return(0);
1167                 break;
1168         }

1170         if (NULL == head) {
1171                 if ( ! mdoc_head_alloc(m, line, ppos, tok))
1172                         return(0);
1173                 head = m->last;
1174         }
1175
1176         if (nl && ! append_delims(m, line, pos, buf))
1177                 return(0);

1179         /* If we've already opened our body, exit now. */

1181         if (NULL != body)
1182                 goto out;
```

```
1184         /*
1185          * If there is an open (i.e., unvalidated) sub-block requiring
1186          * explicit close-out, postpone switching the current block from
1187          * head to body until the rew_sub() call closing out that
1188          * sub-block.
1189          */
1190         for (n = m->last; n && n != head; n = n->parent) {
1191                 if (MDOC_BLOCK == n->type &&
1192                                 MDOC_EXPLICIT & mdoc_macros[n->tok].flags &&
1193                                 ! (MDOC_VALID & n->flags)) {
1194                         n->pending = head;
1195                         return(1);
1196                 }
1197         }

1199         /* Close out scopes to remain in a consistent state. */

1201         if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1202                 return(0);
1203         if ( ! mdoc_body_alloc(m, line, ppos, tok))
1204                 return(0);

1206 out:
1207         if ( ! (MDOC_FREECOL & m->flags))
1208                 return(1);

1210         if ( ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1211                 return(0);
1212         if ( ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1213                 return(0);

1215         m->flags &= ~MDOC_FREECOL;
1216         return(1);
1217 }


1220 static int
1221 blk_part_imp(MACRO_PROT_ARGS)
1222 {
1223         int               la, nl;
1224         enum mdoct        ntok;
1225         enum margserr     ac;
1226         char             *p;
1227         struct mdoc_node *blk; /* saved block context */
1228         struct mdoc_node *body; /* saved body context */
1229         struct mdoc_node *n;

1231         nl = MDOC_NEWLINE & m->flags;

1233         /*
1234          * A macro that spans to the end of the line.  This is generally
1235          * (but not necessarily) called as the first macro.  The block
1236          * has a head as the immediate child, which is always empty,
1237          * followed by zero or more opening punctuation nodes, then the
1238          * body (which may be empty, depending on the macro), then zero
1239          * or more closing punctuation nodes.
1240          */

1242         if ( ! mdoc_block_alloc(m, line, ppos, tok, NULL))
1243                 return(0);

1245         blk = m->last;

1247         if ( ! mdoc_head_alloc(m, line, ppos, tok))
1248                 return(0);
1249         if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
```



```
1250                 return(0);

1252         /*
1253          * Open the body scope "on-demand", that is, after we've
1254          * processed all our the leading delimiters (open parenthesis,
1255          * etc.).
1256          */

1258         for (body = NULL; ; ) {
1259                 la = *pos;
1260                 ac = mdoc_args(m, line, pos, buf, tok, &p);

1262                 if (ARGS_ERROR == ac)
1263                         return(0);
1264                 if (ARGS_EOLN == ac)
1265                         break;
1266                 if (ARGS_PUNCT == ac)
1267                         break;

1269                 if (NULL == body && ARGS_QWORD != ac &&
1270                                 DELIM_OPEN == mdoc_isdelim(p)) {
1271                         if ( ! dword(m, line, la, p, DELIM_OPEN))
1272                                 return(0);
1273                         continue;
1274                 }

1276                 if (NULL == body) {
1277                        if ( ! mdoc_body_alloc(m, line, ppos, tok))
1278                                 return(0);
1279                         body = m->last;
1280                 }

1282                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1284                 if (MDOC_MAX == ntok) {
1285                         if ( ! dword(m, line, la, p, DELIM_MAX))
1286                                 return(0);
1287                         continue;
1288                 }

1290                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1291                         return(0);
1292                 break;
1293         }

1295         /* Clean-ups to leave in a consistent state. */

1297         if (NULL == body) {
1298                 if ( ! mdoc_body_alloc(m, line, ppos, tok))
1299                         return(0);
1300                 body = m->last;
1301         }

1303         for (n = body->child; n && n->next; n = n->next)
1304                 /* Do nothing. */ ;
1305
1306         /*
1307          * End of sentence spacing: if the last node is a text node and
1308          * has a trailing period, then mark it as being end-of-sentence.
1309          */

1311         if (n && MDOC_TEXT == n->type && n->string)
1312                 if (mandoc_eos(n->string, strlen(n->string), 1))
1313                         n->flags |= MDOC_EOS;

1315         /* Up-propagate the end-of-space flag. */
```

```
1317         if (n && (MDOC_EOS & n->flags)) {
1318                 body->flags |= MDOC_EOS;
1319                 body->parent->flags |= MDOC_EOS;
1320         }
1321
1322         /*
1323          * If there is an open sub-block requiring explicit close-out,
1324          * postpone closing out the current block
1325          * until the rew_sub() call closing out the sub-block.
1326          */
1327         for (n = m->last; n && n != body && n != blk->parent; n = n->parent) {
1328                 if (MDOC_BLOCK == n->type &&
1329                     MDOC_EXPLICIT & mdoc_macros[n->tok].flags &&
1330                     ! (MDOC_VALID & n->flags)) {
1331                         make_pending(n, tok, m, line, ppos);
1332                         if ( ! mdoc_endbody_alloc(m, line, ppos,
1333                             tok, body, ENDBODY_NOSPACE))
1334                                 return(0);
1335                         return(1);
1336                 }
1337         }
1338
1339         /*
1340          * If we can't rewind to our body, then our scope has already
1341          * been closed by another macro (like 'Oc' closing 'Op').  This
1342          * is ugly behaviour nodding its head to OpenBSD's overwhelming
1343          * crufty use of 'Op' breakage.
1344          */
1345         if (n != body)
1346                 mandoc_vmsg(MANDOCERR_SCOPENEST, m->parse, line, ppos,
1347                                 "%s broken", mdoc_macronames[tok]);
1348
1349         if (n && ! rew_sub(MDOC_BODY, m, tok, line, ppos))
1350                 return(0);
1351
1352         /* Standard appending of delimiters. */
1353
1354         if (nl && ! append_delims(m, line, pos, buf))
1355                 return(0);
1356
1357         /* Rewind scope, if applicable. */
1358
1359         if (n && ! rew_sub(MDOC_BLOCK, m, tok, line, ppos))
1360                 return(0);
1361
1362         return(1);
1363 }
1364
1365
1366 static int
1367 blk_part_exp(MACRO_PROT_ARGS)
1368 {
1369         int              la, nl;
1370         enum margserr    ac;
1371         struct mdoc_node *head; /* keep track of head */
1372         struct mdoc_node *body; /* keep track of body */
1373         char             *p;
1374         enum mdoct       ntok;
1375
1376         nl = MDOC_NEWLINE & m->flags;
1377
1378         /*
1379          * The opening of an explicit macro having zero or more leading
1380          * punctuation nodes; a head with optional single element (the
1381          * case of 'Eo'); and a body that may be empty.
```



```
1382          */
1383
1384         if ( ! mdoc_block_alloc(m, line, ppos, tok, NULL))
1385                 return(0);
1386
1387         for (head = body = NULL; ; ) {
1388                 la = *pos;
1389                 ac = mdoc_args(m, line, pos, buf, tok, &p);
1390
1391                 if (ARGS_ERROR == ac)
1392                         return(0);
1393                 if (ARGS_PUNCT == ac)
1394                         break;
1395                 if (ARGS_EOLN == ac)
1396                         break;
1397
1398                 /* Flush out leading punctuation. */
1399
1400                 if (NULL == head && ARGS_QWORD != ac &&
1401                                 DELIM_OPEN == mdoc_isdelim(p)) {
1402                         assert(NULL == body);
1403                         if ( ! dword(m, line, la, p, DELIM_OPEN))
1404                                 return(0);
1405                         continue;
1406                 }
1407
1408                 if (NULL == head) {
1409                         assert(NULL == body);
1410                         if ( ! mdoc_head_alloc(m, line, ppos, tok))
1411                                 return(0);
1412                         head = m->last;
1413                 }
1414
1415                 /*
1416                  * 'Eo' gobbles any data into the head, but most other
1417                  * macros just immediately close out and begin the body.
1418                  */
1419
1420                 if (NULL == body) {
1421                         assert(head);
1422                         /* No check whether it's a macro! */
1423                         if (MDOC_Eo == tok)
1424                                 if ( ! dword(m, line, la, p, DELIM_MAX))
1425                                         return(0);
1426
1427                         if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1428                                 return(0);
1429                         if ( ! mdoc_body_alloc(m, line, ppos, tok))
1430                                 return(0);
1431                         body = m->last;
1432
1433                         if (MDOC_Eo == tok)
1434                                 continue;
1435                 }
1436
1437                 assert(NULL != head && NULL != body);
1438
1439                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);
1440
1441                 if (MDOC_MAX == ntok) {
1442                         if ( ! dword(m, line, la, p, DELIM_MAX))
1443                                 return(0);
1444                         continue;
1445                 }
1446
1447                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
```

```
1448                        return(0);
1449                break;
1450        }

1452        /* Clean-up to leave in a consistent state. */

1454        if (NULL == head)
1455                if ( ! mdoc_head_alloc(m, line, ppos, tok))
1456                        return(0);

1458        if (NULL == body) {
1459                if ( ! rew_sub(MDOC_HEAD, m, tok, line, ppos))
1460                        return(0);
1461                if ( ! mdoc_body_alloc(m, line, ppos, tok))
1462                        return(0);
1463        }

1465        /* Standard appending of delimiters. */

1467        if ( ! nl)
1468                return(1);
1469        return(append_delims(m, line, pos, buf));
1470 }


1473 /* ARGSUSED */
1474 static int
1475 in_line_argn(MACRO_PROT_ARGS)
1476 {
1477        int              la, flushed, j, maxargs, nl;
1478        enum margserr    ac;
1479        enum margverr    av;
1480        struct mdoc_arg *arg;
1481        char            *p;
1482        enum mdoct       ntok;

1484        nl = MDOC_NEWLINE & m->flags;

1486        /*
1487         * A line macro that has a fixed number of arguments (maxargs).
1488         * Only open the scope once the first non-leading-punctuation is
1489         * found (unless MDOC_IGNDELIM is noted, like in 'Pf'), then
1490         * keep it open until the maximum number of arguments are
1491         * exhausted.
1492         */

1494        switch (tok) {
1495        case (MDOC_Ap):
1496                /* FALLTHROUGH */
1497        case (MDOC_No):
1498                /* FALLTHROUGH */
1499        case (MDOC_Ns):
1500                /* FALLTHROUGH */
1501        case (MDOC_Ux):
1502                maxargs = 0;
1503                break;
1504        case (MDOC_Bx):
1505                /* FALLTHROUGH */
1506        case (MDOC_Xr):
1507                maxargs = 2;
1508                break;
1509        default:
1510                maxargs = 1;
1511                break;
1512        }
```



```
1514        for (arg = NULL; ; ) {
1515                la = *pos;
1516                av = mdoc_argv(m, line, tok, &arg, pos, buf);

1518                if (ARGV_WORD == av) {
1519                        *pos = la;
1520                        break;
1521                }

1523                if (ARGV_EOLN == av)
1524                        break;
1525                if (ARGV_ARG == av)
1526                        continue;

1528                mdoc_argv_free(arg);
1529                return(0);
1530        }

1532        for (flushed = j = 0; ; ) {
1533                la = *pos;
1534                ac = mdoc_args(m, line, pos, buf, tok, &p);

1536                if (ARGS_ERROR == ac)
1537                        return(0);
1538                if (ARGS_PUNCT == ac)
1539                        break;
1540                if (ARGS_EOLN == ac)
1541                        break;

1543                if ( ! (MDOC_IGNDELIM & mdoc_macros[tok].flags) &&
1544                                ARGS_QWORD != ac && 0 == j &&
1545                                DELIM_OPEN == mdoc_isdelim(p)) {
1546                        if ( ! dword(m, line, la, p, DELIM_OPEN))
1547                                return(0);
1548                        continue;
1549                } else if (0 == j)
1550                        if ( ! mdoc_elem_alloc(m, line, la, tok, arg))
1551                                return(0);

1553                if (j == maxargs && ! flushed) {
1554                        if ( ! rew_elem(m, tok))
1555                                return(0);
1556                        flushed = 1;
1557                }

1559                ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1561                if (MDOC_MAX != ntok) {
1562                        if ( ! flushed && ! rew_elem(m, tok))
1563                                return(0);
1564                        flushed = 1;
1565                        if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1566                                return(0);
1567                        j++;
1568                        break;
1569                }

1571                if ( ! (MDOC_IGNDELIM & mdoc_macros[tok].flags) &&
1572                                ARGS_QWORD != ac &&
1573                                ! flushed &&
1574                                DELIM_NONE != mdoc_isdelim(p)) {
1575                        if ( ! rew_elem(m, tok))
1576                                return(0);
1577                        flushed = 1;
1578                }
```

```
1580                 if ( ! dword(m, line, la, p, DELIM_MAX))
1581                         return(0);
1582                 j++;
1583         }

1585         if (0 == j && ! mdoc_elem_alloc(m, line, la, tok, arg))
1586                 return(0);

1588         /* Close out in a consistent state. */

1590         if ( ! flushed && ! rew_elem(m, tok))
1591                 return(0);
1592         if ( ! nl)
1593                 return(1);
1594         return(append_delims(m, line, pos, buf));
1595 }


1598 static int
1599 in_line_eoln(MACRO_PROT_ARGS)
1600 {
1601         int              la;
1602         enum margserr    ac;
1603         enum margverr    av;
1604         struct mdoc_arg *arg;
1605         char            *p;
1606         enum mdoct       ntok;

1608         assert( ! (MDOC_PARSED & mdoc_macros[tok].flags));

1610         if (tok == MDOC_Pp)
1611                 rew_sub(MDOC_BLOCK, m, MDOC_Nm, line, ppos);

1613         /* Parse macro arguments. */

1615         for (arg = NULL; ; ) {
1616                 la = *pos;
1617                 av = mdoc_argv(m, line, tok, &arg, pos, buf);

1619                 if (ARGV_WORD == av) {
1620                         *pos = la;
1621                         break;
1622                 }
1623                 if (ARGV_EOLN == av)
1624                         break;
1625                 if (ARGV_ARG == av)
1626                         continue;

1628                 mdoc_argv_free(arg);
1629                 return(0);
1630         }

1632         /* Open element scope. */

1634         if ( ! mdoc_elem_alloc(m, line, ppos, tok, arg))
1635                 return(0);

1637         /* Parse argument terms. */

1639         for (;;) {
1640                 la = *pos;
1641                 ac = mdoc_args(m, line, pos, buf, tok, &p);

1643                 if (ARGS_ERROR == ac)
1644                         return(0);
1645                 if (ARGS_EOLN == ac)
```



```
1646                         break;

1648                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup(tok, p);

1650                 if (MDOC_MAX == ntok) {
1651                         if ( ! dword(m, line, la, p, DELIM_MAX))
1652                                 return(0);
1653                         continue;
1654                 }

1656                 if ( ! rew_elem(m, tok))
1657                         return(0);
1658                 return(mdoc_macro(m, ntok, line, la, pos, buf));
1659         }

1661         /* Close out (no delimiters). */

1663         return(rew_elem(m, tok));
1664 }


1667 /* ARGSUSED */
1668 static int
1669 ctx_synopsis(MACRO_PROT_ARGS)
1670 {
1671         int              nl;

1673         nl = MDOC_NEWLINE & m->flags;

1675         /* If we're not in the SYNOPSIS, go straight to in-line. */
1676         if ( ! (MDOC_SYNOPSIS & m->flags))
1677                 return(in_line(m, tok, line, ppos, pos, buf));

1679         /* If we're a nested call, same place. */
1680         if ( ! nl)
1681                 return(in_line(m, tok, line, ppos, pos, buf));

1683         /*
1684          * XXX: this will open a block scope; however, if later we end
1685          * up formatting the block scope, then child nodes will inherit
1686          * the formatting.  Be careful.
1687          */
1688         if (MDOC_Nm == tok)
1689                 return(blk_full(m, tok, line, ppos, pos, buf));
1690         assert(MDOC_Vt == tok);
1691         return(blk_part_imp(m, tok, line, ppos, pos, buf));
1692 }


1695 /* ARGSUSED */
1696 static int
1697 obsolete(MACRO_PROT_ARGS)
1698 {

1700         mdoc_pmsg(m, line, ppos, MANDOCERR_MACROOBS);
1701         return(1);
1702 }


1705 /*
1706  * Phrases occur within 'Bl -column' entries, separated by 'Ta' or tabs.
1707  * They're unusual because they're basically free-form text until a
1708  * macro is encountered.
1709  */
1710 static int
1711 phrase(struct mdoc *m, int line, int ppos, char *buf)
```

```
1712 {
1713         int              la, pos;
1714         enum margserr    ac;
1715         enum mdoct       ntok;
1716         char            *p;

1718         for (pos = ppos; ; ) {
1719                 la = pos;

1721                 ac = mdoc_zargs(m, line, &pos, buf, &p);

1723                 if (ARGS_ERROR == ac)
1724                         return(0);
1725                 if (ARGS_EOLN == ac)
1726                         break;

1728                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup_raw(p);

1730                 if (MDOC_MAX == ntok) {
1731                         if ( ! dword(m, line, la, p, DELIM_MAX))
1732                                 return(0);
1733                         continue;
1734                 }

1736                 if ( ! mdoc_macro(m, ntok, line, la, &pos, buf))
1737                         return(0);
1738                 return(append_delims(m, line, &pos, buf));
1739         }

1741         return(1);
1742 }


1745 /* ARGSUSED */
1746 static int
1747 phrase_ta(MACRO_PROT_ARGS)
1748 {
1749         int              la;
1750         enum mdoct       ntok;
1751         enum margserr    ac;
1752         char            *p;

1754         /*
1755          * FIXME: this is overly restrictive: if the ‘Ta’ is unexpected,
1756          * it should simply error out with ARGSLOST.
1757          */

1759         if ( ! rew_sub(MDOC_BODY, m, MDOC_It, line, ppos))
1760                 return(0);
1761         if ( ! mdoc_body_alloc(m, line, ppos, MDOC_It))
1762                 return(0);

1764         for (;;) {
1765                 la = *pos;
1766                 ac = mdoc_zargs(m, line, pos, buf, &p);

1768                 if (ARGS_ERROR == ac)
1769                         return(0);
1770                 if (ARGS_EOLN == ac)
1771                         break;

1773                 ntok = ARGS_QWORD == ac ? MDOC_MAX : lookup_raw(p);

1775                 if (MDOC_MAX == ntok) {
1776                         if ( ! dword(m, line, la, p, DELIM_MAX))
1777                                 return(0);
```



```
1778                         continue;
1779                 }

1781                 if ( ! mdoc_macro(m, ntok, line, la, pos, buf))
1782                         return(0);
1783                 return(append_delims(m, line, pos, buf));
1784         }

1786         return(1);
1787 }
```

```
**********************************************************
   15647 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/mdoc_man.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_man.c,v 1.9 2011/10/24 21:47:59 schwarze Exp $ */
/*
 * 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 <stdio.h>
#include <string.h>

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

#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      post_bd(DECL_ARGS);
static  void      post_dl(DECL_ARGS);
static  void      post_enc(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  int       pre_ap(DECL_ARGS);
static  int       pre_bd(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_it(DECL_ARGS);
static  int       pre_nm(DECL_ARGS);
static  int       pre_ns(DECL_ARGS);
static  int       pre_pp(DECL_ARGS);
static  int       pre_sp(DECL_ARGS);
static  int       pre_sect(DECL_ARGS);
static  int       pre_ux(DECL_ARGS);
static  int       pre_xr(DECL_ARGS);
static  void      print_word(struct mman *, const char *);
static  void      print_node(DECL_ARGS);

static  const struct manact manacts[MDOC_MAX + 1] = {
        { NULL, pre_ap, NULL, NULL, NULL }, /* Ap */
        { NULL, NULL, NULL, NULL, NULL }, /* Dd */
        { NULL, NULL, NULL, NULL, NULL }, /* Dt */
        { NULL, NULL, NULL, NULL, NULL }, /* Os */
        { NULL, pre_sect, post_sect, ".SH", NULL }, /* Sh */
        { NULL, pre_sect, post_sect, ".SS", NULL }, /* Ss */
        { NULL, pre_pp, NULL, NULL, NULL }, /* Pp */
        { cond_body, pre_dl, post_dl, NULL, NULL }, /* D1 */
        { cond_body, pre_dl, post_dl, NULL, NULL }, /* Dl */
        { cond_body, pre_bd, post_bd, NULL, NULL }, /* Bd */
        { NULL, NULL, NULL, NULL, NULL }, /* Ed */
        { NULL, NULL, NULL, NULL, NULL }, /* Bl */
        { NULL, NULL, NULL, NULL, NULL }, /* El */
        { NULL, pre_it, NULL, NULL, NULL }, /* It */
        { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Ad */
        { NULL, NULL, NULL, NULL, NULL }, /* An */
        { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Ar */
        { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Cd */
        { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Cm */
        { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Dv */
        { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Er */
        { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Ev */
        { NULL, pre_enc, post_enc, "The \\fB",
          "\\fP\nutility exits 0 on success, and >0 if an error occurs."
          }, /* Ex */
        { NULL, NULL, NULL, NULL, NULL }, /* Fa */
        { NULL, NULL, NULL, NULL, NULL }, /* Fd */
        { NULL, pre_enc, post_enc, "\\fB-", "\\fP" }, /* Fl */
        { NULL, NULL, NULL, NULL, NULL }, /* Fn */
        { NULL, NULL, NULL, NULL, NULL }, /* Ft */
        { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Ic */
        { NULL, NULL, NULL, NULL, NULL }, /* In */
        { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Li */
        { cond_head, pre_enc, NULL, "\\- ", NULL }, /* Nd */
        { NULL, pre_nm, post_nm, NULL, NULL }, /* Nm */
        { cond_body, pre_enc, post_enc, "[", "]" }, /* Op */
        { NULL, NULL, NULL, NULL, NULL }, /* Ot */
        { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Pa */
        { NULL, pre_enc, post_enc, "The \\fB",
                "\\fP\nfunction returns the value 0 if successful;\n"
                "otherwise the value -1 is returned and the global\n"
                "variable \\fIerrno\\fP is set to indicate the error."
                }, /* Rv */
        { NULL, NULL, NULL, NULL, NULL }, /* St */
        { NULL, NULL, NULL, NULL, NULL }, /* Va */
        { NULL, NULL, NULL, NULL, NULL }, /* Vt */
        { NULL, pre_xr, NULL, NULL, NULL }, /* Xr */
        { NULL, NULL, post_percent, NULL, NULL }, /* %A */
        { NULL, NULL, NULL, NULL, NULL }, /* %B */
        { NULL, NULL, post_percent, NULL, NULL }, /* %D */
        { NULL, NULL, NULL, NULL, NULL }, /* %I */
        { NULL, pre_enc, post_percent, "\\fI", "\\fP" }, /* %J */
        { NULL, NULL, NULL, NULL, NULL }, /* %N */
        { 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 */
132         { NULL, NULL, NULL, NULL, NULL }, /* Ac */
133         { cond_body, pre_enc, post_enc, "<", ">" }, /* Ao */
134         { cond_body, pre_enc, post_enc, "<", ">" }, /* Aq */
135         { NULL, NULL, NULL, NULL, NULL }, /* At */
136         { NULL, NULL, NULL, NULL, NULL }, /* Bc */
137         { NULL, NULL, NULL, NULL, NULL }, /* _Bf */
138         { cond_body, pre_enc, post_enc, "[", "]" }, /* Bo */
139         { cond_body, pre_enc, post_enc, "[", "]" }, /* Bq */
140         { NULL, pre_ux, NULL, "BSD/OS", NULL }, /* Bsx */
141         { NULL, pre_bx, NULL, NULL, NULL }, /* Bx */
142         { NULL, NULL, NULL, NULL, NULL }, /* Db */
143         { NULL, NULL, NULL, NULL, NULL }, /* Dc */
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 */
150         { NULL, pre_ux, NULL, "FreeBSD", NULL }, /* Fx */
151         { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Ms */
152         { NULL, NULL, NULL, NULL, NULL }, /* No */
153         { NULL, pre_ns, NULL, NULL, NULL }, /* Ns */
154         { NULL, pre_ux, NULL, "NetBSD", NULL }, /* Nx */
155         { NULL, pre_ux, NULL, "OpenBSD", NULL }, /* Ox */
156         { NULL, NULL, NULL, NULL, NULL }, /* Pc */
157         { NULL, NULL, post_pf, NULL, NULL }, /* Pf */
158         { cond_body, pre_enc, post_enc, "(", ")" }, /* Po */
159         { cond_body, pre_enc, post_enc, "(", ")" }, /* Pq */
160         { NULL, NULL, NULL, NULL, NULL }, /* Qc */
161         { cond_body, pre_enc, post_enc, "`", "'" }, /* Ql */
162         { cond_body, pre_enc, post_enc, "\"", "\"" }, /* Qo */
163         { cond_body, pre_enc, post_enc, "\"", "\"" }, /* Qq */
164         { NULL, NULL, NULL, NULL, NULL }, /* Re */
165         { cond_body, pre_pp, NULL, NULL, NULL }, /* Rs */
166         { NULL, NULL, NULL, NULL, NULL }, /* Sc */
167         { cond_body, pre_enc, post_enc, "`", "'" }, /* So */
168         { cond_body, pre_enc, post_enc, "`", "'" }, /* Sq */
169         { NULL, NULL, NULL, NULL, NULL }, /* _Sm */
170         { NULL, pre_enc, post_enc, "\\fI", "\\fP" }, /* Sx */
171         { NULL, pre_enc, post_enc, "\\fB", "\\fP" }, /* Sy */
172         { NULL, pre_enc, post_enc, "\\fR", "\\fP" }, /* Tn */
173         { NULL, pre_ux, NULL, "UNIX", NULL }, /* Ux */
174         { NULL, NULL, NULL, NULL, NULL }, /* _Xc */
175         { NULL, NULL, NULL, NULL, NULL }, /* _Xo */
176         { NULL, NULL, NULL, NULL, NULL }, /* _Fo */
177         { NULL, NULL, NULL, NULL, NULL }, /* _Fc */
178         { cond_body, pre_enc, post_enc, "[", "]" }, /* Oo */
179         { NULL, NULL, NULL, NULL, NULL }, /* Oc */
180         { NULL, NULL, NULL, NULL, NULL }, /* _Bk */
181         { NULL, NULL, NULL, NULL, NULL }, /* _Ek */
182         { NULL, pre_ux, NULL, "is currently in beta test.", NULL }, /* Bt */
183         { NULL, NULL, NULL, NULL, NULL }, /* Hf */
184         { NULL, NULL, NULL, NULL, NULL }, /* Fr */
185         { NULL, pre_ux, NULL, "currently under development.", NULL }, /* Ud */
186         { NULL, NULL, NULL, NULL, NULL }, /* _Lb */
187         { NULL, pre_pp, NULL, NULL, NULL }, /* Lp */
188         { NULL, NULL, NULL, NULL, NULL }, /* _Lk */
189         { NULL, NULL, NULL, NULL, NULL }, /* _Mt */
190         { cond_body, pre_enc, post_enc, "{", "}" }, /* Brq */
191         { cond_body, pre_enc, post_enc, "{", "}" }, /* Bro */
192         { NULL, NULL, NULL, NULL, NULL }, /* Brc */
193         { NULL, NULL, NULL, NULL, NULL }, /* _%C */
```



```
194         { NULL, NULL, NULL, NULL, NULL }, /* _Es */
195         { NULL, NULL, NULL, NULL, NULL }, /* _En */
196         { NULL, pre_ux, NULL, "DragonFly", NULL }, /* Dx */
197         { NULL, NULL, NULL, NULL, NULL }, /* _%Q */
198         { NULL, pre_br, NULL, NULL, NULL }, /* br */
199         { NULL, pre_sp, post_sp, NULL, NULL }, /* sp */
200         { NULL, NULL, NULL, NULL, NULL }, /* _%U */
201         { NULL, NULL, NULL, NULL, NULL }, /* _Ta */
202         { NULL, NULL, NULL, NULL, NULL }, /* ROOT */
203 };

205 static void
206 print_word(struct mman *mm, const char *s)
207 {

209         if (mm->need_nl) {
210                 /*
211                  * If we need a newline, print it now and start afresh.
212                  */
213                 putchar('\n');
214                 mm->need_space = 0;
215                 mm->need_nl = 0;
216         } else if (mm->need_space && '\0' != s[0])
217                 /*
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.
222                  */
223                 if (NULL == strchr(".,:;)]?!", s[0]) || '\0' != s[1])
224                         putchar(' ');

226         /*
227          * Reassign needing space if we're not following opening
228          * punctuation.
229          */
230         mm->need_space =
231                 ('(' != s[0] && '[' != s[0]) || '\0' != s[1];

233         for ( ; *s; s++) {
234                 switch (*s) {
235                 case (ASCII_NBRSP):
236                         printf("\\~");
237                         break;
238                 case (ASCII_HYPH):
239                         putchar('-');
240                         break;
241                 default:
242                         putchar((unsigned char)*s);
243                         break;
244                 }
245         }
246 }

248 void
249 man_man(void *arg, const struct man *man)
250 {

252         /*
253          * Dump the keep buffer.
254          * We're guaranteed by now that this exists (is non-NULL).
255          * Flush stdout afterward, just in case.
256          */
257         fputs(mparse_getkeep(man_mparse(man)), stdout);
258         fflush(stdout);
259 }
```

```
261 void
262 man_mdoc(void *arg, const struct mdoc *mdoc)
263 {
264         const struct mdoc_meta *m;
265         const struct mdoc_node *n;
266         struct mman             mm;
267
268         m = mdoc_meta(mdoc);
269         n = mdoc_node(mdoc);
270
271         printf(".TH \"%s\" \"%s\" \"%s\" \"%s\" \"%s\"",
272                         m->title, m->msec, m->date, m->os, m->vol);
273
274         memset(&mm, 0, sizeof(struct mman));
275
276         mm.need_nl = 1;
277         print_node(m, n, &mm);
278         putchar('\n');
279 }
280
281 static void
282 print_node(DECL_ARGS)
283 {
284         const struct mdoc_node  *prev, *sub;
285         const struct manact     *act;
286         int                      cond, do_sub;
287
288         /*
289          * Break the line if we were parsed subsequent the current node.
290          * This makes the page structure be more consistent.
291          */
292         prev = n->prev ? n->prev : n->parent;
293         if (prev && prev->line < n->line)
294                 mm->need_nl = 1;
295
296         act = NULL;
297         cond = 0;
298         do_sub = 1;
299
300         if (MDOC_TEXT == n->type) {
301                 /*
302                  * Make sure that we don't happen to start with a
303                  * control character at the start of a line.
304                  */
305                 if (mm->need_nl && ('.' == *n->string ||
306                                     '\'' == *n->string)) {
307                         print_word(mm, "\\&");
308                         mm->need_space = 0;
309                 }
310                 print_word(mm, n->string);
311         } else {
312                 /*
313                  * Conditionally run the pre-node action handler for a
314                  * node.
315                  */
316                 act = manacts + n->tok;
317                 cond = NULL == act->cond || (*act->cond)(m, n, mm);
318                 if (cond && act->pre)
319                         do_sub = (*act->pre)(m, n, mm);
320         }
321
322         /*
323          * Conditionally run all child nodes.
324          * Note that this iterates over children instead of using
325          * recursion.  This prevents unnecessary depth in the stack.
```



```
326          */
327         if (do_sub)
328                 for (sub = n->child; sub; sub = sub->next)
329                         print_node(m, sub, mm);
330
331         /*
332          * Lastly, conditionally run the post-node handler.
333          */
334         if (cond && act->post)
335                 (*act->post)(m, n, mm);
336 }
337
338 static int
339 cond_head(DECL_ARGS)
340 {
341
342         return(MDOC_HEAD == n->type);
343 }
344
345 static int
346 cond_body(DECL_ARGS)
347 {
348
349         return(MDOC_BODY == n->type);
350 }
351
352 /*
353  * Output a font encoding before a node, e.g., \fR.
354  * This obviously has no trailing space.
355  */
356 static int
357 pre_enc(DECL_ARGS)
358 {
359         const char      *prefix;
360
361         prefix = manacts[n->tok].prefix;
362         if (NULL == prefix)
363                 return(1);
364         print_word(mm, prefix);
365         mm->need_space = 0;
366         return(1);
367 }
368
369 /*
370  * Output a font encoding subsequent a node, e.g., \fP.
371  */
372 static void
373 post_enc(DECL_ARGS)
374 {
375         const char *suffix;
376
377         suffix = manacts[n->tok].suffix;
378         if (NULL == suffix)
379                 return;
380         mm->need_space = 0;
381         print_word(mm, suffix);
382 }
383
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  */
389 static void
390 post_percent(DECL_ARGS)
391 {
```

```
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;
399         }
400 }

402 /*
403  * Print before a section header.
404  */
405 static int
406 pre_sect(DECL_ARGS)
407 {

409         if (MDOC_HEAD != n->type)
410                 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);
416 }

418 /*
419  * Print subsequent a section header.
420  */
421 static void
422 post_sect(DECL_ARGS)
423 {

425         if (MDOC_HEAD != n->type)
426                 return;
427         mm->need_space = 0;
428         print_word(mm, "\"");
429         mm->need_nl = 1;
430 }

432 static int
433 pre_ap(DECL_ARGS)
434 {

436         mm->need_space = 0;
437         print_word(mm, "‘");
438         mm->need_space = 0;
439         return(0);
440 }

442 static int
443 pre_bd(DECL_ARGS)
444 {

446         if (DISP_unfilled == n->norm->Bd.type ||
447             DISP_literal  == n->norm->Bd.type) {
448                 mm->need_nl = 1;
449                 print_word(mm, ".nf");
450         }
451         mm->need_nl = 1;
452         return(1);
453 }

455 static void
456 post_bd(DECL_ARGS)
457 {
```



```
459         if (DISP_unfilled == n->norm->Bd.type ||
460             DISP_literal  == n->norm->Bd.type) {
461                 mm->need_nl = 1;
462                 print_word(mm, ".fi");
463         }
464         mm->need_nl = 1;
465 }

467 static int
468 pre_br(DECL_ARGS)
469 {

471         mm->need_nl = 1;
472         print_word(mm, ".br");
473         mm->need_nl = 1;
474         return(0);
475 }

477 static int
478 pre_bx(DECL_ARGS)
479 {

481         n = n->child;
482         if (n) {
483                 print_word(mm, n->string);
484                 mm->need_space = 0;
485                 n = n->next;
486         }
487         print_word(mm, "BSD");
488         if (NULL == n)
489                 return(0);
490         mm->need_space = 0;
491         print_word(mm, "-");
492         mm->need_space = 0;
493         print_word(mm, n->string);
494         return(0);
495 }

497 static int
498 pre_dl(DECL_ARGS)
499 {

501         mm->need_nl = 1;
502         print_word(mm, ".RS 6n");
503         mm->need_nl = 1;
504         return(1);
505 }

507 static void
508 post_dl(DECL_ARGS)
509 {

511         mm->need_nl = 1;
512         print_word(mm, ".RE");
513         mm->need_nl = 1;
514 }

516 static int
517 pre_it(DECL_ARGS)
518 {
519         const struct mdoc_node *bln;

521         if (MDOC_HEAD == n->type) {
522                 mm->need_nl = 1;
523                 print_word(mm, ".TP");
```

```
524                 bln = n->parent->parent->prev;
525                 switch (bln->norm->Bl.type) {
526                 case (LIST_bullet):
527                         print_word(mm, "4n");
528                         mm->need_nl = 1;
529                         print_word(mm, "\\fBo\\fP");
530                         break;
531                 default:
532                         if (bln->norm->Bl.width)
533                                 print_word(mm, bln->norm->Bl.width);
534                         break;
535                 }
536                 mm->need_nl = 1;
537         }
538         return(1);
539 }

541 static int
542 pre_nm(DECL_ARGS)
543 {

545         if (MDOC_ELEM != n->type && MDOC_HEAD != n->type)
546                 return(1);
547         print_word(mm, "\\fB");
548         mm->need_space = 0;
549         if (NULL == n->child)
550                 print_word(mm, m->name);
551         return(1);
552 }

554 static void
555 post_nm(DECL_ARGS)
556 {

558         if (MDOC_ELEM != n->type && MDOC_HEAD != n->type)
559                 return;
560         mm->need_space = 0;
561         print_word(mm, "\\fP");
562 }

564 static int
565 pre_ns(DECL_ARGS)
566 {

568         mm->need_space = 0;
569         return(0);
570 }

572 static void
573 post_pf(DECL_ARGS)
574 {

576         mm->need_space = 0;
577 }

579 static int
580 pre_pp(DECL_ARGS)
581 {

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;
589         return(1);
```



```
590 }

592 static int
593 pre_sp(DECL_ARGS)
594 {

596         mm->need_nl = 1;
597         print_word(mm, ".sp");
598         return(1);
599 }

601 static void
602 post_sp(DECL_ARGS)
603 {

605         mm->need_nl = 1;
606 }

608 static int
609 pre_xr(DECL_ARGS)
610 {

612         n = n->child;
613         if (NULL == n)
614                 return(0);
615         print_node(m, n, mm);
616         n = n->next;
617         if (NULL == n)
618                 return(0);
619         mm->need_space = 0;
620         print_word(mm, "(");
621         print_node(m, n, mm);
622         print_word(mm, ")");
623         return(0);
624 }

626 static int
627 pre_ux(DECL_ARGS)
628 {

630         print_word(mm, manacts[n->tok].prefix);
631         if (NULL == n->child)
632                 return(0);
633         mm->need_space = 0;
634         print_word(mm, "\\~");
635         mm->need_space = 0;
636         return(1);
637 }
```

```
**********************************************************
   44751 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/mdoc_term.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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 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 *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_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);
static  int       termp_vt_pre(DECL_ARGS);
static  int       termp_xr_pre(DECL_ARGS);
static  int       termp_xx_pre(DECL_ARGS);

static  const struct termact termacts[MDOC_MAX] = {
        { termp_ap_pre, NULL }, /* Ap */
        { NULL, NULL }, /* Dd */
        { 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_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, 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         { 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         { 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         { 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  *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         m = mdoc_meta(mdoc);

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

274         if (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, m, n);
286         if (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;
297         struct termpair  npair;
298         size_t           offset, rmargin;

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

305         memset(&npair, 0, sizeof(struct termpair));
306         npair.ppair = pair;

308         /*
309          * Keeps only work until the end of a line.  If a keep was
310          * 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.
314          */

316         if (TERMP_KEEP & p->flags || MDOC_SYNPRETTY & n->flags) {
317                 if (n->prev && n->prev->line != n->line) {
318                         p->flags &= ~TERMP_KEEP;
319                         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;
324                         }
325                 }
```



```
326         }

328         /*
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         /*
339          * After the keep flags have been set up, we may now
340          * produce output.  Note that some pre-handlers do so.
341          */

343         switch (n->type) {
344         case (MDOC_TEXT):
345                 if (' ' == *n->string && MDOC_LINE & n->flags)
346                         term_newln(p);
347                 if (MDOC_DELIMC & n->flags)
348                         p->flags |= TERMP_NOSPACE;
349                 term_word(p, n->string);
350                 if (MDOC_DELIMO & n->flags)
351                         p->flags |= TERMP_NOSPACE;
352                 break;
353         case (MDOC_EQN):
354                 term_eqn(p, n->eqn);
355                 break;
356         case (MDOC_TBL):
357                 term_tbl(p, n->span);
358                 break;
359         default:
360                 if (termacts[n->tok].pre && ENDBODY_NOT == n->end)
361                         chld = (*termacts[n->tok].pre)
362                                 (p, &npair, m, n);
363                 break;
364         }

366         if (chld && n->child)
367                 print_mdoc_nodelist(p, &npair, m, n->child);

369         term_fontpopq(p, font);

371         switch (n->type) {
372         case (MDOC_TEXT):
373                 break;
374         case (MDOC_TBL):
375                 break;
376         case (MDOC_EQN):
377                 break;
378         default:
379                 if ( ! termacts[n->tok].post || MDOC_ENDED & n->flags)
380                         break;
381                 (void)(*termacts[n->tok].post)(p, &npair, m, n);

383                 /*
384                  * Explicit end tokens not only call the post
385                  * handler, but also tell the respective block
386                  * that it must not call the post handler again.
387                  */
388                 if (ENDBODY_NOT != n->end)
389                         n->pending->flags |= MDOC_ENDED;

391                 /*
```

```
392                  * End of line terminating an implicit block
393                  * while an explicit block is still open.
394                  * Continue the explicit block without spacing.
395                  */
396                 if (ENDBODY_NOSPACE == n->end)
397                         p->flags |= TERMP_NOSPACE;
398                 break;
399         }
401         if (MDOC_EOS & n->flags)
402                 p->flags |= TERMP_SENTENCE;
404         p->offset = offset;
405         p->rmargin = rmargin;
406 }
409 static void
410 print_mdoc_foot(struct termp *p, const void *arg)
411 {
412         const struct mdoc_meta *m;
414         m = (const struct mdoc_meta *)arg;
416         term_fontrepl(p, TERMFONT_NONE);
418         /*
419          * Output the footer in new-groff style, that is, three columns
420          * with the middle being the manual date and flanking columns
421          * being the operating system:
422          *
423          * SYSTEM                  DATE                    SYSTEM
424          */
426         term_vspace(p);
428         p->offset = 0;
429         p->rmargin = (p->maxrmargin -
430                       term_strlen(p, m->date) + term_len(p, 1)) / 2;
431         p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;
433         term_word(p, m->os);
434         term_flushln(p);
436         p->offset = p->rmargin;
437         p->rmargin = p->maxrmargin - term_strlen(p, m->os);
438         p->flags |= TERMP_NOSPACE;
440         term_word(p, m->date);
441         term_flushln(p);
443         p->offset = p->rmargin;
444         p->rmargin = p->maxrmargin;
445         p->flags &= ~TERMP_NOBREAK;
446         p->flags |= TERMP_NOSPACE;
448         term_word(p, m->os);
449         term_flushln(p);
451         p->offset = 0;
452         p->rmargin = p->maxrmargin;
453         p->flags = 0;
454 }
457 static void
```



```
458 print_mdoc_head(struct termp *p, const void *arg)
459 {
460         char            buf[BUFSIZ], title[BUFSIZ];
461         size_t          buflen, titlen;
462         const struct mdoc_meta *m;
464         m = (const struct mdoc_meta *)arg;
466         /*
467          * The header is strange.  It has three components, which are
468          * really two with the first duplicated.  It goes like this:
469          *
470          * IDENTIFIER              TITLE                   IDENTIFIER
471          *
472          * The IDENTIFIER is NAME(SECTION), which is the command-name
473          * (if given, or "unknown" if not) followed by the manual page
474          * section.  These are given in 'Dt'.  The TITLE is a free-form
475          * string depending on the manual volume.  If not specified, it
476          * switches on the manual section.
477          */
479         p->offset = 0;
480         p->rmargin = p->maxrmargin;
482         assert(m->vol);
483         strlcpy(buf, m->vol, BUFSIZ);
484         buflen = term_strlen(p, buf);
486         if (m->arch) {
487                 strlcat(buf, " (", BUFSIZ);
488                 strlcat(buf, m->arch, BUFSIZ);
489                 strlcat(buf, ")", BUFSIZ);
490         }
492         snprintf(title, BUFSIZ, "%s(%s)", m->title, m->msec);
493         titlen = term_strlen(p, title);
495         p->flags |= TERMP_NOBREAK | TERMP_NOSPACE;
496         p->offset = 0;
497         p->rmargin = 2 * (titlen+1) + buflen < p->maxrmargin ?
498             (p->maxrmargin -
499              term_strlen(p, buf) + term_len(p, 1)) / 2 :
500             p->maxrmargin - buflen;
502         term_word(p, title);
503         term_flushln(p);
505         p->flags |= TERMP_NOSPACE;
506         p->offset = p->rmargin;
507         p->rmargin = p->offset + buflen + titlen < p->maxrmargin ?
508             p->maxrmargin - titlen : p->maxrmargin;
510         term_word(p, buf);
511         term_flushln(p);
513         p->flags &= ~TERMP_NOBREAK;
514         if (p->rmargin + titlen <= p->maxrmargin) {
515                 p->flags |= TERMP_NOSPACE;
516                 p->offset = p->rmargin;
517                 p->rmargin = p->maxrmargin;
518                 term_word(p, title);
519                 term_flushln(p);
520         }
522         p->flags &= ~TERMP_NOSPACE;
523         p->offset = 0;
```

```
524         p->rmargin = p->maxrmargin;
525 }


528 static size_t
529 a2height(const struct termp *p, const char *v)
530 {
531         struct roffsu    su;


534         assert(v);
535         if ( ! a2roffsu(v, &su, SCALE_VS))
536                 SCALE_VS_INIT(&su, atoi(v));

538         return(term_vspan(p, &su));
539 }


542 static size_t
543 a2width(const struct termp *p, const char *v)
544 {
545         struct roffsu    su;

547         assert(v);
548         if ( ! a2roffsu(v, &su, SCALE_MAX))
549                 SCALE_HS_INIT(&su, term_strlen(p, v));

551         return(term_hspan(p, &su));
552 }


555 static size_t
556 a2offs(const struct termp *p, const char *v)
557 {
558         struct roffsu    su;

560         if ('\0' == *v)
561                 return(0);
562         else if (0 == strcmp(v, "left"))
563                 return(0);
564         else if (0 == strcmp(v, "indent"))
565                 return(term_len(p, p->defindent + 1));
566         else if (0 == strcmp(v, "indent-two"))
567                 return(term_len(p, (p->defindent + 1) * 2));
568         else if ( ! a2roffsu(v, &su, SCALE_MAX))
569                 SCALE_HS_INIT(&su, term_strlen(p, v));

571         return(term_hspan(p, &su));
572 }


575 /*
576  * Determine how much space to print out before block elements of 'It'
577  * (and thus 'Bl') and 'Bd'.  And then go ahead and print that space,
578  * too.
579  */
580 static void
581 print_bvspace(struct termp *p,
582                 const struct mdoc_node *bl,
583                 const struct mdoc_node *n)
584 {
585         const struct mdoc_node  *nn;

587         assert(n);

589         term_newln(p);
```



```
591         if (MDOC_Bd == bl->tok && bl->norm->Bd.comp)
592                 return;
593         if (MDOC_Bl == bl->tok && bl->norm->Bl.comp)
594                 return;

596         /* Do not vspace directly after Ss/Sh. */

598         for (nn = n; nn; nn = nn->parent) {
599                 if (MDOC_BLOCK != nn->type)
600                         continue;
601                 if (MDOC_Ss == nn->tok)
602                         return;
603                 if (MDOC_Sh == nn->tok)
604                         return;
605                 if (NULL == nn->prev)
606                         continue;
607                 break;
608         }

610         /* A '-column' does not assert vspace within the list. */

612         if (MDOC_Bl == bl->tok && LIST_column == bl->norm->Bl.type)
613                 if (n->prev && MDOC_It == n->prev->tok)
614                         return;

616         /* A '-diag' without body does not vspace. */

618         if (MDOC_Bl == bl->tok && LIST_diag == bl->norm->Bl.type)
619                 if (n->prev && MDOC_It == n->prev->tok) {
620                         assert(n->prev->body);
621                         if (NULL == n->prev->body->child)
622                                 return;
623                 }

625         term_vspace(p);
626 }


629 /* ARGSUSED */
630 static int
631 termp_it_pre(DECL_ARGS)
632 {
633         const struct mdoc_node *bl, *nn;
634         char                    buf[7];
635         int                     i;
636         size_t                  width, offset, ncols, dcol;
637         enum mdoc_list          type;

639         if (MDOC_BLOCK == n->type) {
640                 print_bvspace(p, n->parent->parent, n);
641                 return(1);
642         }

644         bl = n->parent->parent->parent;
645         type = bl->norm->Bl.type;

647         /*
648          * First calculate width and offset.  This is pretty easy unless
649          * we're a -column list, in which case all prior columns must
650          * be accounted for.
651          */

653         width = offset = 0;

655         if (bl->norm->Bl.offs)
```

```
656                 offset = a2offs(p, bl->norm->Bl.offs);

658         switch (type) {
659         case (LIST_column):
660                 if (MDOC_HEAD == n->type)
661                         break;

663                 /*
664                  * Imitate groff's column handling:
665                  * - For each earlier column, add its width.
666                  * - For less than 5 columns, add four more blanks per
667                  *   column.
668                  * - For exactly 5 columns, add three more blank per
669                  *   column.
670                  * - For more than 5 columns, add only one column.
671                  */
672                 ncols = bl->norm->Bl.ncols;

674                 /* LINTED */
675                 dcol = ncols < 5 ? term_len(p, 4) :
676                         ncols == 5 ? term_len(p, 3) : term_len(p, 1);

678                 /*
679                  * Calculate the offset by applying all prior MDOC_BODY,
680                  * so we stop at the MDOC_HEAD (NULL == nn->prev).
681                  */

683                 for (i = 0, nn = n->prev;
684                                 nn->prev && i < (int)ncols;
685                                 nn = nn->prev, i++)
686                         offset += dcol + a2width
687                                 (p, bl->norm->Bl.cols[i]);

689                 /*
690                  * When exceeding the declared number of columns, leave
691                  * the remaining widths at 0.  This will later be
692                  * adjusted to the default width of 10, or, for the last
693                  * column, stretched to the right margin.
694                  */
695                 if (i >= (int)ncols)
696                         break;

698                 /*
699                  * Use the declared column widths, extended as explained
700                  * in the preceding paragraph.
701                  */
702                 width = a2width(p, bl->norm->Bl.cols[i]) + dcol;
703                 break;
704         default:
705                 if (NULL == bl->norm->Bl.width)
706                         break;

708                 /*
709                  * Note: buffer the width by 2, which is groff's magic
710                  * number for buffering single arguments.  See the above
711                  * handling for column for how this changes.
712                  */
713                 assert(bl->norm->Bl.width);
714                 width = a2width(p, bl->norm->Bl.width) + term_len(p, 2);
715                 break;
716         }

718         /*
719          * List-type can override the width in the case of fixed-head
720          * values (bullet, dash/hyphen, enum).  Tags need a non-zero
721          * offset.
```



```
722          */

724         switch (type) {
725         case (LIST_bullet):
726                 /* FALLTHROUGH */
727         case (LIST_dash):
728                 /* FALLTHROUGH */
729         case (LIST_hyphen):
730                 if (width < term_len(p, 4))
731                         width = term_len(p, 4);
732                 break;
733         case (LIST_enum):
734                 if (width < term_len(p, 5))
735                         width = term_len(p, 5);
736                 break;
737         case (LIST_hang):
738                 if (0 == width)
739                         width = term_len(p, 8);
740                 break;
741         case (LIST_column):
742                 /* FALLTHROUGH */
743         case (LIST_tag):
744                 if (0 == width)
745                         width = term_len(p, 10);
746                 break;
747         default:
748                 break;
749         }

751         /*
752          * Whitespace control.  Inset bodies need an initial space,
753          * while diagonal bodies need two.
754          */

756         p->flags |= TERMP_NOSPACE;

758         switch (type) {
759         case (LIST_diag):
760                 if (MDOC_BODY == n->type)
761                         term_word(p, "\\ \\ ");
762                 break;
763         case (LIST_inset):
764                 if (MDOC_BODY == n->type)
765                         term_word(p, "\\ ");
766                 break;
767         default:
768                 break;
769         }

771         p->flags |= TERMP_NOSPACE;

773         switch (type) {
774         case (LIST_diag):
775                 if (MDOC_HEAD == n->type)
776                         term_fontpush(p, TERMFONT_BOLD);
777                 break;
778         default:
779                 break;
780         }

782         /*
783          * Pad and break control.  This is the tricky part.  These flags
784          * are documented in term_flushln() in term.c.  Note that we're
785          * going to unset all of these flags in termp_it_post() when we
786          * exit.
787          */
```

```
789         switch (type) {
790         case (LIST_bullet):
791                 /* FALLTHROUGH */
792         case (LIST_dash):
793                 /* FALLTHROUGH */
794         case (LIST_enum):
795                 /* FALLTHROUGH */
796         case (LIST_hyphen):
797                 if (MDOC_HEAD == n->type)
798                         p->flags |= TERMP_NOBREAK;
799                 break;
800         case (LIST_hang):
801                 if (MDOC_HEAD == n->type)
802                         p->flags |= TERMP_NOBREAK;
803                 else
804                         break;
805
806                 /*
807                  * This is ugly.  If ‘-hang’ is specified and the body
808                  * is a ‘Bl’ or ‘Bd’, then we want basically to nullify
809                  * the "overstep" effect in term_flushln() and treat
810                  * this as a ‘-ohang’ list instead.
811                  */
812                 if (n->next->child &&
813                                 (MDOC_Bl == n->next->child->tok ||
814                                  MDOC_Bd == n->next->child->tok))
815                         p->flags &= ~TERMP_NOBREAK;
816                 else
817                         p->flags |= TERMP_HANG;
818                 break;
819         case (LIST_tag):
820                 if (MDOC_HEAD == n->type)
821                         p->flags |= TERMP_NOBREAK | TERMP_TWOSPACE;
822
823                 if (MDOC_HEAD != n->type)
824                         break;
825                 if (NULL == n->next || NULL == n->next->child)
826                         p->flags |= TERMP_DANGLE;
827                 break;
828         case (LIST_column):
829                 if (MDOC_HEAD == n->type)
830                         break;
831
832                 if (NULL == n->next)
833                         p->flags &= ~TERMP_NOBREAK;
834                 else
835                         p->flags |= TERMP_NOBREAK;
836
837                 break;
838         case (LIST_diag):
839                 if (MDOC_HEAD == n->type)
840                         p->flags |= TERMP_NOBREAK;
841                 break;
842         default:
843                 break;
844         }
845
846         /*
847          * Margin control.  Set-head-width lists have their right
848          * margins shortened.  The body for these lists has the offset
849          * necessarily lengthened.  Everybody gets the offset.
850          */
851
852         p->offset += offset;
```



```
854         switch (type) {
855         case (LIST_hang):
856                 /*
857                  * Same stipulation as above, regarding ‘-hang’.  We
858                  * don’t want to recalculate rmargin and offsets when
859                  * using ‘Bd’ or ‘Bl’ within ‘-hang’ overstep lists.
860                  */
861                 if (MDOC_HEAD == n->type && n->next->child &&
862                                 (MDOC_Bl == n->next->child->tok ||
863                                  MDOC_Bd == n->next->child->tok))
864                         break;
865                 /* FALLTHROUGH */
866         case (LIST_bullet):
867                 /* FALLTHROUGH */
868         case (LIST_dash):
869                 /* FALLTHROUGH */
870         case (LIST_enum):
871                 /* FALLTHROUGH */
872         case (LIST_hyphen):
873                 /* FALLTHROUGH */
874         case (LIST_tag):
875                 assert(width);
876                 if (MDOC_HEAD == n->type)
877                         p->rmargin = p->offset + width;
878                 else
879                         p->offset += width;
880                 break;
881         case (LIST_column):
882                 assert(width);
883                 p->rmargin = p->offset + width;
884                 /*
885                  * XXX - this behaviour is not documented: the
886                  * right-most column is filled to the right margin.
887                  */
888                 if (MDOC_HEAD == n->type)
889                         break;
890                 if (NULL == n->next && p->rmargin < p->maxrmargin)
891                         p->rmargin = p->maxrmargin;
892                 break;
893         default:
894                 break;
895         }
896
897         /*
898          * The dash, hyphen, bullet and enum lists all have a special
899          * HEAD character (temporarily bold, in some cases).
900          */
901
902         if (MDOC_HEAD == n->type)
903                 switch (type) {
904                 case (LIST_bullet):
905                         term_fontpush(p, TERMFONT_BOLD);
906                         term_word(p, "\\[bu]");
907                         term_fontpop(p);
908                         break;
909                 case (LIST_dash):
910                         /* FALLTHROUGH */
911                 case (LIST_hyphen):
912                         term_fontpush(p, TERMFONT_BOLD);
913                         term_word(p, "\\(hy");
914                         term_fontpop(p);
915                         break;
916                 case (LIST_enum):
917                         (pair->ppair->ppair->count)++;
918                         snprintf(buf, sizeof(buf), "%d.",
919                                         pair->ppair->ppair->count);
```

```
920                         term_word(p, buf);
921                         break;
922                 default:
923                         break;
924                 }

926         /*
927          * If we're not going to process our children, indicate so here.
928          */

930         switch (type) {
931         case (LIST_bullet):
932                 /* FALLTHROUGH */
933         case (LIST_item):
934                 /* FALLTHROUGH */
935         case (LIST_dash):
936                 /* FALLTHROUGH */
937         case (LIST_hyphen):
938                 /* FALLTHROUGH */
939         case (LIST_enum):
940                 if (MDOC_HEAD == n->type)
941                         return(0);
942                 break;
943         case (LIST_column):
944                 if (MDOC_HEAD == n->type)
945                         return(0);
946                 break;
947         default:
948                 break;
949         }

951         return(1);
952 }


955 /* ARGSUSED */
956 static void
957 termp_it_post(DECL_ARGS)
958 {
959         enum mdoc_list     type;

961         if (MDOC_BLOCK == n->type)
962                 return;

964         type = n->parent->parent->parent->norm->Bl.type;

966         switch (type) {
967         case (LIST_item):
968                 /* FALLTHROUGH */
969         case (LIST_diag):
970                 /* FALLTHROUGH */
971         case (LIST_inset):
972                 if (MDOC_BODY == n->type)
973                         term_newln(p);
974                 break;
975         case (LIST_column):
976                 if (MDOC_BODY == n->type)
977                         term_flushln(p);
978                 break;
979         default:
980                 term_newln(p);
981                 break;
982         }

984         /*
985          * Now that our output is flushed, we can reset our tags.  Since
```



```
986          * only 'It' sets these flags, we're free to assume that nobody
987          * has munged them in the meanwhile.
988          */

990         p->flags &= ~TERMP_DANGLE;
991         p->flags &= ~TERMP_NOBREAK;
992         p->flags &= ~TERMP_TWOSPACE;
993         p->flags &= ~TERMP_HANG;
994 }


997 /* ARGSUSED */
998 static int
999 termp_nm_pre(DECL_ARGS)
1000 {

1002         if (MDOC_BLOCK == n->type)
1003                 return(1);

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 ? term_strlen(p, m->name) :
1011                      MDOC_TEXT == n->prev->child->type ?
1012                         term_strlen(p, n->prev->child->string) :
1013                      term_len(p, 5));
1014                 return(1);
1015         }

1017         if (NULL == n->child && NULL == m->name)
1018                 return(0);

1020         if (MDOC_HEAD == n->type)
1021                 synopsis_pre(p, n->parent);

1023         if (MDOC_HEAD == n->type && n->next->child) {
1024                 p->flags |= TERMP_NOSPACE | TERMP_NOBREAK;
1025                 p->rmargin = p->offset + term_len(p, 1);
1026                 if (NULL == n->child) {
1027                         p->rmargin += term_strlen(p, m->name);
1028                 } else if (MDOC_TEXT == n->child->type) {
1029                         p->rmargin += term_strlen(p, n->child->string);
1030                         if (n->child->next)
1031                                 p->flags |= TERMP_HANG;
1032                 } else {
1033                         p->rmargin += term_len(p, 5);
1034                         p->flags |= TERMP_HANG;
1035                 }
1036         }

1038         term_fontpush(p, TERMFONT_BOLD);
1039         if (NULL == n->child)
1040                 term_word(p, m->name);
1041         return(1);
1042 }


1045 /* ARGSUSED */
1046 static void
1047 termp_nm_post(DECL_ARGS)
1048 {

1050         if (MDOC_HEAD == n->type && n->next->child) {
1051                 term_flushln(p);
```

```
1052                 p->flags &= ~(TERMP_NOBREAK | TERMP_HANG);
1053         } else if (MDOC_BODY == n->type && n->child)
1054                 term_flushln(p);
1055 }
1057
1058 /* ARGSUSED */
1059 static int
1060 termp_fl_pre(DECL_ARGS)
1061 {
1063         term_fontpush(p, TERMFONT_BOLD);
1064         term_word(p, "\\-");
1066         if (n->child)
1067                 p->flags |= TERMP_NOSPACE;
1068         else if (n->next && n->next->line == n->line)
1069                 p->flags |= TERMP_NOSPACE;
1071         return(1);
1072 }
1075 /* ARGSUSED */
1076 static int
1077 termp__a_pre(DECL_ARGS)
1078 {
1080         if (n->prev && MDOC__A == n->prev->tok)
1081                 if (NULL == n->next || MDOC__A != n->next->tok)
1082                         term_word(p, "and");
1084         return(1);
1085 }
1088 /* ARGSUSED */
1089 static int
1090 termp_an_pre(DECL_ARGS)
1091 {
1093         if (NULL == n->child)
1094                 return(1);
1096         /*
1097          * If not in the AUTHORS section, ‘An -split’ will cause
1098          * newlines to occur before the author name.  If in the AUTHORS
1099          * section, by default, the first ‘An’ invocation is nosplit,
1100          * then all subsequent ones, regardless of whether interspersed
1101          * with other macros/text, are split.  -split, in this case,
1102          * will override the condition of the implied first -nosplit.
1103          */
1104
1105         if (n->sec == SEC_AUTHORS) {
1106                 if ( ! (TERMP_ANPREC & p->flags)) {
1107                         if (TERMP_SPLIT & p->flags)
1108                                 term_newln(p);
1109                         return(1);
1110                 }
1111                 if (TERMP_NOSPLIT & p->flags)
1112                         return(1);
1113                 term_newln(p);
1114                 return(1);
1115         }
1117         if (TERMP_SPLIT & p->flags)
```



```
1118                 term_newln(p);
1120         return(1);
1121 }
1124 /* ARGSUSED */
1125 static void
1126 termp_an_post(DECL_ARGS)
1127 {
1129         if (n->child) {
1130                 if (SEC_AUTHORS == n->sec)
1131                         p->flags |= TERMP_ANPREC;
1132                 return;
1133         }
1135         if (AUTH_split == n->norm->An.auth) {
1136                 p->flags &= ~TERMP_NOSPLIT;
1137                 p->flags |= TERMP_SPLIT;
1138         } else if (AUTH_nosplit == n->norm->An.auth) {
1139                 p->flags &= ~TERMP_SPLIT;
1140                 p->flags |= TERMP_NOSPLIT;
1141         }
1143 }
1146 /* ARGSUSED */
1147 static int
1148 termp_ns_pre(DECL_ARGS)
1149 {
1151         if ( ! (MDOC_LINE & n->flags))
1152                 p->flags |= TERMP_NOSPACE;
1153         return(1);
1154 }
1157 /* ARGSUSED */
1158 static int
1159 termp_rs_pre(DECL_ARGS)
1160 {
1162         if (SEC_SEE_ALSO != n->sec)
1163                 return(1);
1164         if (MDOC_BLOCK == n->type && n->prev)
1165                 term_vspace(p);
1166         return(1);
1167 }
1170 /* ARGSUSED */
1171 static int
1172 termp_rv_pre(DECL_ARGS)
1173 {
1174         int              nchild;
1176         term_newln(p);
1177         term_word(p, "The");
1179         nchild = n->nchild;
1180         for (n = n->child; n; n = n->next) {
1181                 term_fontpush(p, TERMFONT_BOLD);
1182                 term_word(p, n->string);
1183                 term_fontpop(p);
```

```
1185                 p->flags |= TERMP_NOSPACE;
1186                 term_word(p, "()");
1187
1188                 if (nchild > 2 && n->next) {
1189                         p->flags |= TERMP_NOSPACE;
1190                         term_word(p, ",");
1191                 }
1192
1193                 if (n->next && NULL == n->next->next)
1194                         term_word(p, "and");
1195         }
1196
1197         if (nchild > 1)
1198                 term_word(p, "functions return");
1199         else
1200                 term_word(p, "function returns");
1201
1202         term_word(p, "the value 0 if successful; otherwise the value "
1203                         "-1 is returned and the global variable");
1204
1205         term_fontpush(p, TERMFONT_UNDER);
1206         term_word(p, "errno");
1207         term_fontpop(p);
1208
1209         term_word(p, "is set to indicate the error.");
1210         p->flags |= TERMP_SENTENCE;
1211
1212         return(0);
1213 }
1214
1215
1216 /* ARGSUSED */
1217 static int
1218 termp_ex_pre(DECL_ARGS)
1219 {
1220         int              nchild;
1221
1222         term_newln(p);
1223         term_word(p, "The");
1224
1225         nchild = n->nchild;
1226         for (n = n->child; n; n = n->next) {
1227                 term_fontpush(p, TERMFONT_BOLD);
1228                 term_word(p, n->string);
1229                 term_fontpop(p);
1230
1231                 if (nchild > 2 && n->next) {
1232                         p->flags |= TERMP_NOSPACE;
1233                         term_word(p, ",");
1234                 }
1235
1236                 if (n->next && NULL == n->next->next)
1237                         term_word(p, "and");
1238         }
1239
1240         if (nchild > 1)
1241                 term_word(p, "utilities exit");
1242         else
1243                 term_word(p, "utility exits");
1244
1245         term_word(p, "0 on success, and >0 if an error occurs.");
1246
1247         p->flags |= TERMP_SENTENCE;
1248         return(0);
1249 }
```



```
1252 /* ARGSUSED */
1253 static int
1254 termp_nd_pre(DECL_ARGS)
1255 {
1256
1257         if (MDOC_BODY != n->type)
1258                 return(1);
1259
1260 #if defined(__OpenBSD__) || defined(__linux__)
1261         term_word(p, "\\(en");
1262 #else
1263         term_word(p, "\\(em");
1264 #endif
1265         return(1);
1266 }
1267
1268
1269 /* ARGSUSED */
1270 static int
1271 termp_bl_pre(DECL_ARGS)
1272 {
1273
1274         return(MDOC_HEAD != n->type);
1275 }
1276
1277
1278 /* ARGSUSED */
1279 static void
1280 termp_bl_post(DECL_ARGS)
1281 {
1282
1283         if (MDOC_BLOCK == n->type)
1284                 term_newln(p);
1285 }
1286
1287 /* ARGSUSED */
1288 static int
1289 termp_xr_pre(DECL_ARGS)
1290 {
1291
1292         if (NULL == (n = n->child))
1293                 return(0);
1294
1295         assert(MDOC_TEXT == n->type);
1296         term_word(p, n->string);
1297
1298         if (NULL == (n = n->next))
1299                 return(0);
1300
1301         p->flags |= TERMP_NOSPACE;
1302         term_word(p, "(");
1303         p->flags |= TERMP_NOSPACE;
1304
1305         assert(MDOC_TEXT == n->type);
1306         term_word(p, n->string);
1307
1308         p->flags |= TERMP_NOSPACE;
1309         term_word(p, ")");
1310
1311         return(0);
1312 }
1313
1314 /*
1315  * This decides how to assert whitespace before any of the SYNOPSIS set
```

```
 * of macros (which, as in the case of Ft/Fo and Ft/Fn, may contain
 * macro combos).
 */
static void
synopsis_pre(struct termp *p, const struct mdoc_node *n)
{
        /*
         * Obviously, if we're not in a SYNOPSIS or no prior macros
         * exist, do nothing.
         */
        if (NULL == n->prev || ! (MDOC_SYNPRETTY & n->flags))
                return;

        /*
         * If we're the second in a pair of like elements, emit our
         * newline and return.  UNLESS we're 'Fo', 'Fn', 'Fn', in which
         * case we soldier on.
         */
        if (n->prev->tok == n->tok &&
                        MDOC_Ft != n->tok &&
                        MDOC_Fo != n->tok &&
                        MDOC_Fn != n->tok) {
                term_newln(p);
                return;
        }

        /*
         * If we're one of the SYNOPSIS set and non-like pair-wise after
         * another (or Fn/Fo, which we've let slip through) then assert
         * vertical space, else only newline and move on.
         */
        switch (n->prev->tok) {
        case (MDOC_Fd):
                /* FALLTHROUGH */
        case (MDOC_Fn):
                /* FALLTHROUGH */
        case (MDOC_Fo):
                /* FALLTHROUGH */
        case (MDOC_In):
                /* FALLTHROUGH */
        case (MDOC_Vt):
                term_vspace(p);
                break;
        case (MDOC_Ft):
                if (MDOC_Fn != n->tok && MDOC_Fo != n->tok) {
                        term_vspace(p);
                        break;
                }
                /* FALLTHROUGH */
        default:
                term_newln(p);
                break;
        }
}


static int
termp_vt_pre(DECL_ARGS)
{

        if (MDOC_ELEM == n->type) {
                synopsis_pre(p, n);
                return(termp_under_pre(p, pair, m, n));
        } else if (MDOC_BLOCK == n->type) {
                synopsis_pre(p, n);
                return(1);
```



```
        } else if (MDOC_HEAD == n->type)
                return(0);

        return(termp_under_pre(p, pair, m, n));
}


/* ARGSUSED */
static int
termp_bold_pre(DECL_ARGS)
{

        term_fontpush(p, TERMFONT_BOLD);
        return(1);
}


/* ARGSUSED */
static int
termp_fd_pre(DECL_ARGS)
{

        synopsis_pre(p, n);
        return(termp_bold_pre(p, pair, m, n));
}


/* ARGSUSED */
static int
termp_sh_pre(DECL_ARGS)
{

        /* No vspace between consecutive 'Sh' calls. */

        switch (n->type) {
        case (MDOC_BLOCK):
                if (n->prev && MDOC_Sh == n->prev->tok)
                        if (NULL == n->prev->body->child)
                                break;
                term_vspace(p);
                break;
        case (MDOC_HEAD):
                term_fontpush(p, TERMFONT_BOLD);
                break;
        case (MDOC_BODY):
                p->offset = term_len(p, p->defindent);
                break;
        default:
                break;
        }
        return(1);
}


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

        switch (n->type) {
        case (MDOC_HEAD):
                term_newln(p);
                break;
        case (MDOC_BODY):
                term_newln(p);
                p->offset = 0;
```

```
1448                 break;
1449         default:
1450                 break;
1451         }
1452 }


1455 /* ARGSUSED */
1456 static int
1457 termp_bt_pre(DECL_ARGS)
1458 {

1460         term_word(p, "is currently in beta test.");
1461         p->flags |= TERMP_SENTENCE;
1462         return(0);
1463 }


1466 /* ARGSUSED */
1467 static void
1468 termp_lb_post(DECL_ARGS)
1469 {

1471         if (SEC_LIBRARY == n->sec && MDOC_LINE & n->flags)
1472                 term_newln(p);
1473 }


1476 /* ARGSUSED */
1477 static int
1478 termp_ud_pre(DECL_ARGS)
1479 {

1481         term_word(p, "currently under development.");
1482         p->flags |= TERMP_SENTENCE;
1483         return(0);
1484 }


1487 /* ARGSUSED */
1488 static int
1489 termp_d1_pre(DECL_ARGS)
1490 {

1492         if (MDOC_BLOCK != n->type)
1493                 return(1);
1494         term_newln(p);
1495         p->offset += term_len(p, p->defindent + 1);
1496         return(1);
1497 }


1500 /* 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 */
1512 static int
1513 termp_ft_pre(DECL_ARGS)
```



```
1514 {

1516         /* NB: MDOC_LINE does not effect this! */
1517         synopsis_pre(p, n);
1518         term_fontpush(p, TERMFONT_UNDER);
1519         return(1);
1520 }


1523 /* ARGSUSED */
1524 static int
1525 termp_fn_pre(DECL_ARGS)
1526 {
1527         int              pretty;

1529         pretty = MDOC_SYNPRETTY & n->flags;

1531         synopsis_pre(p, n);

1533         if (NULL == (n = n->child))
1534                 return(0);

1536         assert(MDOC_TEXT == n->type);
1537         term_fontpush(p, TERMFONT_BOLD);
1538         term_word(p, n->string);
1539         term_fontpop(p);

1541         p->flags |= TERMP_NOSPACE;
1542         term_word(p, "(");
1543         p->flags |= TERMP_NOSPACE;

1545         for (n = n->next; n; n = n->next) {
1546                 assert(MDOC_TEXT == n->type);
1547                 term_fontpush(p, TERMFONT_UNDER);
1548                 term_word(p, n->string);
1549                 term_fontpop(p);

1551                 if (n->next) {
1552                         p->flags |= TERMP_NOSPACE;
1553                         term_word(p, ",");
1554                 }
1555         }

1557         p->flags |= TERMP_NOSPACE;
1558         term_word(p, ")");

1560         if (pretty) {
1561                 p->flags |= TERMP_NOSPACE;
1562                 term_word(p, ";");
1563         }

1565         return(0);
1566 }


1569 /* ARGSUSED */
1570 static int
1571 termp_fa_pre(DECL_ARGS)
1572 {
1573         const struct mdoc_node  *nn;

1575         if (n->parent->tok != MDOC_Fo) {
1576                 term_fontpush(p, TERMFONT_UNDER);
1577                 return(1);
1578         }
```

```
1580         for (nn = n->child; nn; nn = nn->next) {
1581                 term_fontpush(p, TERMFONT_UNDER);
1582                 term_word(p, nn->string);
1583                 term_fontpop(p);
1584
1585                 if (nn->next) {
1586                         p->flags |= TERMP_NOSPACE;
1587                         term_word(p, ",");
1588                 }
1589         }
1590
1591         if (n->child && n->next && n->next->tok == MDOC_Fa) {
1592                 p->flags |= TERMP_NOSPACE;
1593                 term_word(p, ",");
1594         }
1595
1596         return(0);
1597 }
1598
1599
1600 /* ARGSUSED */
1601 static int
1602 termp_bd_pre(DECL_ARGS)
1603 {
1604         size_t                  tabwidth, rm, rmax;
1605         const struct mdoc_node  *nn;
1606
1607         if (MDOC_BLOCK == n->type) {
1608                 print_bvspace(p, n, n);
1609                 return(1);
1610         } else if (MDOC_HEAD == n->type)
1611                 return(0);
1612
1613         if (n->norm->Bd.offs)
1614                 p->offset += a2offs(p, n->norm->Bd.offs);
1615
1616         /*
1617          * If -ragged or -filled are specified, the block does nothing
1618          * but change the indentation.  If -unfilled or -literal are
1619          * specified, text is printed exactly as entered in the display:
1620          * for macro lines, a newline is appended to the line.  Blank
1621          * lines are allowed.
1622          */
1623
1624         if (DISP_literal != n->norm->Bd.type &&
1625                         DISP_unfilled != n->norm->Bd.type)
1626                 return(1);
1627
1628         tabwidth = p->tabwidth;
1629         if (DISP_literal == n->norm->Bd.type)
1630                 p->tabwidth = term_len(p, 8);
1631
1632         rm = p->rmargin;
1633         rmax = p->maxrmargin;
1634         p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
1635
1636         for (nn = n->child; nn; nn = nn->next) {
1637                 print_mdoc_node(p, pair, m, nn);
1638                 /*
1639                  * If the printed node flushes its own line, then we
1640                  * needn't do it here as well.  This is hacky, but the
1641                  * notion of selective eoln whitespace is pretty dumb
1642                  * anyway, so don't sweat it.
1643                  */
1644                 switch (nn->tok) {
1645                 case (MDOC_Sm):
```



```
1646                         /* FALLTHROUGH */
1647                 case (MDOC_br):
1648                         /* FALLTHROUGH */
1649                 case (MDOC_sp):
1650                         /* FALLTHROUGH */
1651                 case (MDOC_Bl):
1652                         /* FALLTHROUGH */
1653                 case (MDOC_D1):
1654                         /* FALLTHROUGH */
1655                 case (MDOC_Dl):
1656                         /* FALLTHROUGH */
1657                 case (MDOC_Lp):
1658                         /* FALLTHROUGH */
1659                 case (MDOC_Pp):
1660                         continue;
1661                 default:
1662                         break;
1663                 }
1664                 if (nn->next && nn->next->line == nn->line)
1665                         continue;
1666                 term_flushln(p);
1667                 p->flags |= TERMP_NOSPACE;
1668         }
1669
1670         p->tabwidth = tabwidth;
1671         p->rmargin = rm;
1672         p->maxrmargin = rmax;
1673         return(0);
1674 }
1675
1676
1677 /* ARGSUSED */
1678 static void
1679 termp_bd_post(DECL_ARGS)
1680 {
1681         size_t           rm, rmax;
1682
1683         if (MDOC_BODY != n->type)
1684                 return;
1685
1686         rm = p->rmargin;
1687         rmax = p->maxrmargin;
1688
1689         if (DISP_literal == n->norm->Bd.type ||
1690                         DISP_unfilled == n->norm->Bd.type)
1691                 p->rmargin = p->maxrmargin = TERM_MAXMARGIN;
1692
1693         p->flags |= TERMP_NOSPACE;
1694         term_newln(p);
1695
1696         p->rmargin = rm;
1697         p->maxrmargin = rmax;
1698 }
1699
1700
1701 /* ARGSUSED */
1702 static int
1703 termp_bx_pre(DECL_ARGS)
1704 {
1705
1706         if (NULL != (n = n->child)) {
1707                 term_word(p, n->string);
1708                 p->flags |= TERMP_NOSPACE;
1709                 term_word(p, "BSD");
1710         } else {
1711                 term_word(p, "BSD");
```

```
1712                 return(0);
1713         }

1715         if (NULL != (n = n->next)) {
1716                 p->flags |= TERMP_NOSPACE;
1717                 term_word(p, "-");
1718                 p->flags |= TERMP_NOSPACE;
1719                 term_word(p, n->string);
1720         }

1722         return(0);
1723 }


1726 /* ARGSUSED */
1727 static int
1728 termp_xx_pre(DECL_ARGS)
1729 {
1730         const char      *pp;
1731         int              flags;

1733         pp = NULL;
1734         switch (n->tok) {
1735         case (MDOC_Bsx):
1736                 pp = "BSD/OS";
1737                 break;
1738         case (MDOC_Dx):
1739                 pp = "DragonFly";
1740                 break;
1741         case (MDOC_Fx):
1742                 pp = "FreeBSD";
1743                 break;
1744         case (MDOC_Nx):
1745                 pp = "NetBSD";
1746                 break;
1747         case (MDOC_Ox):
1748                 pp = "OpenBSD";
1749                 break;
1750         case (MDOC_Ux):
1751                 pp = "UNIX";
1752                 break;
1753         default:
1754                 break;
1755         }

1757         term_word(p, pp);
1758         if (n->child) {
1759                 flags = p->flags;
1760                 p->flags |= TERMP_KEEP;
1761                 term_word(p, n->child->string);
1762                 p->flags = flags;
1763         }
1764         return(0);
1765 }


1768 /* ARGSUSED */
1769 static int
1770 termp_igndelim_pre(DECL_ARGS)
1771 {

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



```
1778 /* ARGSUSED */
1779 static void
1780 termp_pf_post(DECL_ARGS)
1781 {

1783         p->flags |= TERMP_NOSPACE;
1784 }


1787 /* ARGSUSED */
1788 static int
1789 termp_ss_pre(DECL_ARGS)
1790 {

1792         switch (n->type) {
1793         case (MDOC_BLOCK):
1794                 term_newln(p);
1795                 if (n->prev)
1796                         term_vspace(p);
1797                 break;
1798         case (MDOC_HEAD):
1799                 term_fontpush(p, TERMFONT_BOLD);
1800                 p->offset = term_len(p, (p->defindent+1)/2);
1801                 break;
1802         default:
1803                 break;
1804         }

1806         return(1);
1807 }


1810 /* ARGSUSED */
1811 static void
1812 termp_ss_post(DECL_ARGS)
1813 {

1815         if (MDOC_HEAD == n->type)
1816                 term_newln(p);
1817 }


1820 /* ARGSUSED */
1821 static int
1822 termp_cd_pre(DECL_ARGS)
1823 {

1825         synopsis_pre(p, n);
1826         term_fontpush(p, TERMFONT_BOLD);
1827         return(1);
1828 }


1831 /* ARGSUSED */
1832 static int
1833 termp_in_pre(DECL_ARGS)
1834 {

1836         synopsis_pre(p, n);

1838         if (MDOC_SYNPRETTY & n->flags && MDOC_LINE & n->flags) {
1839                 term_fontpush(p, TERMFONT_BOLD);
1840                 term_word(p, "#include");
1841                 term_word(p, "<");
1842         } else {
1843                 term_word(p, "<");
```

```
1844                 term_fontpush(p, TERMFONT_UNDER);
1845         }

1847         p->flags |= TERMP_NOSPACE;
1848         return(1);
1849 }


1852 /* ARGSUSED */
1853 static void
1854 termp_in_post(DECL_ARGS)
1855 {

1857         if (MDOC_SYNPRETTY & n->flags)
1858                 term_fontpush(p, TERMFONT_BOLD);

1860         p->flags |= TERMP_NOSPACE;
1861         term_word(p, ">");

1863         if (MDOC_SYNPRETTY & n->flags)
1864                 term_fontpop(p);
1865 }


1868 /* ARGSUSED */
1869 static int
1870 termp_sp_pre(DECL_ARGS)
1871 {
1872         size_t           i, len;

1874         switch (n->tok) {
1875         case (MDOC_sp):
1876                 len = n->child ? a2height(p, n->child->string) : 1;
1877                 break;
1878         case (MDOC_br):
1879                 len = 0;
1880                 break;
1881         default:
1882                 len = 1;
1883                 break;
1884         }

1886         if (0 == len)
1887                 term_newln(p);
1888         for (i = 0; i < len; i++)
1889                 term_vspace(p);

1891         return(0);
1892 }


1895 /* ARGSUSED */
1896 static int
1897 termp_quote_pre(DECL_ARGS)
1898 {

1900         if (MDOC_BODY != n->type && MDOC_ELEM != n->type)
1901                 return(1);

1903         switch (n->tok) {
1904         case (MDOC_Ao):
1905                 /* FALLTHROUGH */
1906         case (MDOC_Aq):
1907                 term_word(p, "<");
1908                 break;
1909         case (MDOC_Bro):
```



```
1910                 /* FALLTHROUGH */
1911         case (MDOC_Brq):
1912                 term_word(p, "{");
1913                 break;
1914         case (MDOC_Oo):
1915                 /* FALLTHROUGH */
1916         case (MDOC_Op):
1917                 /* FALLTHROUGH */
1918         case (MDOC_Bo):
1919                 /* FALLTHROUGH */
1920         case (MDOC_Bq):
1921                 term_word(p, "[");
1922                 break;
1923         case (MDOC_Do):
1924                 /* FALLTHROUGH */
1925         case (MDOC_Dq):
1926                 term_word(p, "“");
1927                 break;
1928         case (MDOC_Eo):
1929                 break;
1930         case (MDOC_Po):
1931                 /* FALLTHROUGH */
1932         case (MDOC_Pq):
1933                 term_word(p, "(");
1934                 break;
1935         case (MDOC__T):
1936                 /* FALLTHROUGH */
1937         case (MDOC_Qo):
1938                 /* FALLTHROUGH */
1939         case (MDOC_Qq):
1940                 term_word(p, "\"");
1941                 break;
1942         case (MDOC_Ql):
1943                 /* FALLTHROUGH */
1944         case (MDOC_So):
1945                 /* FALLTHROUGH */
1946         case (MDOC_Sq):
1947                 term_word(p, "‘");
1948                 break;
1949         default:
1950                 abort();
1951                 /* NOTREACHED */
1952         }

1954         p->flags |= TERMP_NOSPACE;
1955         return(1);
1956 }


1959 /* ARGSUSED */
1960 static void
1961 termp_quote_post(DECL_ARGS)
1962 {

1964         if (MDOC_BODY != n->type && MDOC_ELEM != n->type)
1965                 return;

1967         p->flags |= TERMP_NOSPACE;

1969         switch (n->tok) {
1970         case (MDOC_Ao):
1971                 /* FALLTHROUGH */
1972         case (MDOC_Aq):
1973                 term_word(p, ">");
1974                 break;
1975         case (MDOC_Bro):
```

```
1976                 /* FALLTHROUGH */
1977         case (MDOC_Brq):
1978                 term_word(p, "}");
1979                 break;
1980         case (MDOC_Oo):
1981                 /* FALLTHROUGH */
1982         case (MDOC_Op):
1983                 /* FALLTHROUGH */
1984         case (MDOC_Bo):
1985                 /* FALLTHROUGH */
1986         case (MDOC_Bq):
1987                 term_word(p, "]");
1988                 break;
1989         case (MDOC_Do):
1990                 /* FALLTHROUGH */
1991         case (MDOC_Dq):
1992                 term_word(p, "’’");
1993                 break;
1994         case (MDOC_Eo):
1995                 break;
1996         case (MDOC_Po):
1997                 /* FALLTHROUGH */
1998         case (MDOC_Pq):
1999                 term_word(p, ")");
2000                 break;
2001         case (MDOC__T):
2002                 /* FALLTHROUGH */
2003         case (MDOC_Qo):
2004                 /* FALLTHROUGH */
2005         case (MDOC_Qq):
2006                 term_word(p, "\"");
2007                 break;
2008         case (MDOC_Ql):
2009                 /* FALLTHROUGH */
2010         case (MDOC_So):
2011                 /* FALLTHROUGH */
2012         case (MDOC_Sq):
2013                 term_word(p, "’");
2014                 break;
2015         default:
2016                 abort();
2017                 /* NOTREACHED */
2018         }
2019 }


2022 /* ARGSUSED */
2023 static int
2024 termp_fo_pre(DECL_ARGS)
2025 {

2027         if (MDOC_BLOCK == n->type) {
2028                 synopsis_pre(p, n);
2029                 return(1);
2030         } else if (MDOC_BODY == n->type) {
2031                 p->flags |= TERMP_NOSPACE;
2032                 term_word(p, "(");
2033                 p->flags |= TERMP_NOSPACE;
2034                 return(1);
2035         }

2037         if (NULL == n->child)
2038                 return(0);

2040         /* XXX: we drop non-initial arguments as per groff. */
```



```
2042         assert(n->child->string);
2043         term_fontpush(p, TERMFONT_BOLD);
2044         term_word(p, n->child->string);
2045         return(0);
2046 }


2049 /* ARGSUSED */
2050 static void
2051 termp_fo_post(DECL_ARGS)
2052 {

2054         if (MDOC_BODY != n->type)
2055                 return;

2057         p->flags |= TERMP_NOSPACE;
2058         term_word(p, ")");

2060         if (MDOC_SYNPRETTY & n->flags) {
2061                 p->flags |= TERMP_NOSPACE;
2062                 term_word(p, ";");
2063         }
2064 }


2067 /* ARGSUSED */
2068 static int
2069 termp_bf_pre(DECL_ARGS)
2070 {

2072         if (MDOC_HEAD == n->type)
2073                 return(0);
2074         else if (MDOC_BLOCK != n->type)
2075                 return(1);

2077         if (FONT_Em == n->norm->Bf.font)
2078                 term_fontpush(p, TERMFONT_UNDER);
2079         else if (FONT_Sy == n->norm->Bf.font)
2080                 term_fontpush(p, TERMFONT_BOLD);
2081         else
2082                 term_fontpush(p, TERMFONT_NONE);

2084         return(1);
2085 }


2088 /* ARGSUSED */
2089 static int
2090 termp_sm_pre(DECL_ARGS)
2091 {

2093         assert(n->child && MDOC_TEXT == n->child->type);
2094         if (0 == strcmp("on", n->child->string)) {
2095                 if (p->col)
2096                         p->flags &= ~TERMP_NOSPACE;
2097                 p->flags &= ~TERMP_NONOSPACE;
2098         } else
2099                 p->flags |= TERMP_NONOSPACE;

2101         return(0);
2102 }


2105 /* ARGSUSED */
2106 static int
2107 termp_ap_pre(DECL_ARGS)
```

```
2108 {
2110         p->flags |= TERMP_NOSPACE;
2111         term_word(p, "’");
2112         p->flags |= TERMP_NOSPACE;
2113         return(1);
2114 }


2117 /* ARGSUSED */
2118 static void
2119 termp____post(DECL_ARGS)
2120 {
2122         /*
2123          * Handle lists of authors.  In general, print each followed by
2124          * a comma.  Don’t print the comma if there are only two
2125          * authors.
2126          */
2127         if (MDOC__A == n->tok && n->next && MDOC__A == n->next->tok)
2128                 if (NULL == n->next->next || MDOC__A != n->next->next->tok)
2129                         if (NULL == n->prev || MDOC__A != n->prev->tok)
2130                                 return;
2132         /* TODO: %U. */
2134         if (NULL == n->parent || MDOC_Rs != n->parent->tok)
2135                 return;
2137         p->flags |= TERMP_NOSPACE;
2138         if (NULL == n->next) {
2139                 term_word(p, ".");
2140                 p->flags |= TERMP_SENTENCE;
2141         } else
2142                 term_word(p, ",");
2143 }


2146 /* ARGSUSED */
2147 static int
2148 termp_li_pre(DECL_ARGS)
2149 {
2151         term_fontpush(p, TERMFONT_NONE);
2152         return(1);
2153 }


2156 /* ARGSUSED */
2157 static int
2158 termp_lk_pre(DECL_ARGS)
2159 {
2160         const struct mdoc_node *nn, *sv;
2162         term_fontpush(p, TERMFONT_UNDER);
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);
```



```
2174         p->flags |= TERMP_NOSPACE;
2175         term_word(p, ":");
2177         term_fontpush(p, TERMFONT_BOLD);
2178         term_word(p, sv->string);
2179         term_fontpop(p);
2181         return(0);
2182 }


2185 /* ARGSUSED */
2186 static int
2187 termp_bk_pre(DECL_ARGS)
2188 {
2190         switch (n->type) {
2191         case (MDOC_BLOCK):
2192                 break;
2193         case (MDOC_HEAD):
2194                 return(0);
2195         case (MDOC_BODY):
2196                 if (n->parent->args || 0 == n->prev->nchild)
2197                         p->flags |= TERMP_PREKEEP;
2198                 break;
2199         default:
2200                 abort();
2201                 /* NOTREACHED */
2202         }
2204         return(1);
2205 }


2208 /* ARGSUSED */
2209 static void
2210 termp_bk_post(DECL_ARGS)
2211 {
2213         if (MDOC_BODY == n->type)
2214                 p->flags &= ~(TERMP_KEEP | TERMP_PREKEEP);
2215 }
2217 /* ARGSUSED */
2218 static void
2219 termp__t_post(DECL_ARGS)
2220 {
2222         /*
2223          * If we’re in an ‘Rs’ and there’s a journal present, then quote
2224          * us instead of underlining us (for disambiguation).
2225          */
2226         if (n->parent && MDOC_Rs == n->parent->tok &&
2227                         n->parent->norm->Rs.quote_T)
2228                 termp_quote_post(p, pair, m, n);
2230         termp____post(p, pair, m, n);
2231 }
2233 /* ARGSUSED */
2234 static int
2235 termp__t_pre(DECL_ARGS)
2236 {
2238         /*
2239          * If we’re in an ‘Rs’ and there’s a journal present, then quote
```

```
2240          * us instead of underlining us (for disambiguation).
2241          */
2242         if (n->parent && MDOC_Rs == n->parent->tok &&
2243                         n->parent->norm->Rs.quote_T)
2244                 return(termp_quote_pre(p, pair, m, n));

2246         term_fontpush(p, TERMFONT_UNDER);
2247         return(1);
2248 }

2250 /* ARGSUSED */
2251 static int
2252 termp_under_pre(DECL_ARGS)
2253 {

2255         term_fontpush(p, TERMFONT_UNDER);
2256         return(1);
2257 }
```

```
**********************************************************
   51934 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/mdoc_validate.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: mdoc_validate.c,v 1.182 2012/03/23 05:50:25 kristaps Exp $ */
/*
 * 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

#ifndef OSNAME
#include <sys/utsname.h>
#endif

#include <sys/types.h>

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

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

/* FIXME: .Bl -diag can't have non-text children in HEAD. */

#define PRE_ARGS  struct mdoc *mdoc, struct mdoc_node *n
#define POST_ARGS struct mdoc *mdoc

#define NUMSIZ    32
#define DATESIZE  32

enum    check_ineq {
        CHECK_LT,
        CHECK_GT,
        CHECK_EQ
};

enum    check_lvl {
        CHECK_WARN,
        CHECK_ERROR,
};

typedef int     (*v_pre)(PRE_ARGS);
typedef int     (*v_post)(POST_ARGS);
```



```
struct  valids {
        v_pre   *pre;
        v_post  *post;
};

static  int      check_count(struct mdoc *, enum mdoc_type,
                        enum check_lvl, enum check_ineq, int);
static  int      check_parent(PRE_ARGS, enum mdoct, enum mdoc_type);
static  void     check_text(struct mdoc *, int, int, char *);
static  void     check_argv(struct mdoc *,
                        struct mdoc_node *, struct mdoc_argv *);
static  void     check_args(struct mdoc *, struct mdoc_node *);
static  int      concat(char *, const struct mdoc_node *, size_t);
static  enum mdoc_sec   a2sec(const char *);
static  size_t          macro2len(enum mdoct);

static  int      ebool(POST_ARGS);
static  int      berr_ge1(POST_ARGS);
static  int      bwarn_ge1(POST_ARGS);
static  int      ewarn_eq0(POST_ARGS);
static  int      ewarn_eq1(POST_ARGS);
static  int      ewarn_ge1(POST_ARGS);
static  int      ewarn_le1(POST_ARGS);
static  int      hwarn_eq0(POST_ARGS);
static  int      hwarn_eq1(POST_ARGS);
static  int      hwarn_ge1(POST_ARGS);
static  int      hwarn_le1(POST_ARGS);

static  int      post_an(POST_ARGS);
static  int      post_at(POST_ARGS);
static  int      post_bf(POST_ARGS);
static  int      post_bl(POST_ARGS);
static  int      post_bl_block(POST_ARGS);
static  int      post_bl_block_width(POST_ARGS);
static  int      post_bl_block_tag(POST_ARGS);
static  int      post_bl_head(POST_ARGS);
static  int      post_bx(POST_ARGS);
static  int      post_dd(POST_ARGS);
static  int      post_dt(POST_ARGS);
static  int      post_defaults(POST_ARGS);
static  int      post_literal(POST_ARGS);
static  int      post_eoln(POST_ARGS);
static  int      post_it(POST_ARGS);
static  int      post_lb(POST_ARGS);
static  int      post_nm(POST_ARGS);
static  int      post_ns(POST_ARGS);
static  int      post_os(POST_ARGS);
static  int      post_ignpar(POST_ARGS);
static  int      post_prol(POST_ARGS);
static  int      post_root(POST_ARGS);
static  int      post_rs(POST_ARGS);
static  int      post_sh(POST_ARGS);
static  int      post_sh_body(POST_ARGS);
static  int      post_sh_head(POST_ARGS);
static  int      post_st(POST_ARGS);
static  int      post_std(POST_ARGS);
static  int      post_vt(POST_ARGS);
static  int      pre_an(PRE_ARGS);
static  int      pre_bd(PRE_ARGS);
static  int      pre_bl(PRE_ARGS);
static  int      pre_dd(PRE_ARGS);
static  int      pre_display(PRE_ARGS);
static  int      pre_dt(PRE_ARGS);
static  int      pre_it(PRE_ARGS);
static  int      pre_literal(PRE_ARGS);
```

```
128 static  int      pre_os(PRE_ARGS);
129 static  int      pre_par(PRE_ARGS);
130 static  int      pre_sh(PRE_ARGS);
131 static  int      pre_ss(PRE_ARGS);
132 static  int      pre_std(PRE_ARGS);

134 static  v_post   posts_an[] = { post_an, NULL };
135 static  v_post   posts_at[] = { post_at, post_defaults, NULL };
136 static  v_post   posts_bd[] = { post_literal, hwarn_eq0, bwarn_ge1, NULL };
137 static  v_post   posts_bf[] = { hwarn_le1, post_bf, NULL };
138 static  v_post   posts_bk[] = { hwarn_eq0, bwarn_ge1, NULL };
139 static  v_post   posts_bl[] = { bwarn_ge1, post_bl, NULL };
140 static  v_post   posts_bx[] = { post_bx, NULL };
141 static  v_post   posts_bool[] = { ebool, NULL };
142 static  v_post   posts_eoln[] = { post_eoln, NULL };
143 static  v_post   posts_defaults[] = { post_defaults, NULL };
144 static  v_post   posts_dd[] = { post_dd, post_prol, NULL };
145 static  v_post   posts_dl[] = { post_literal, bwarn_ge1, NULL };
146 static  v_post   posts_dt[] = { post_dt, post_prol, NULL };
147 static  v_post   posts_fo[] = { hwarn_eq1, bwarn_ge1, NULL };
148 static  v_post   posts_it[] = { post_it, NULL };
149 static  v_post   posts_lb[] = { post_lb, NULL };
150 static  v_post   posts_nd[] = { berr_ge1, NULL };
151 static  v_post   posts_nm[] = { post_nm, NULL };
152 static  v_post   posts_notext[] = { ewarn_eq0, NULL };
153 static  v_post   posts_ns[] = { post_ns, NULL };
154 static  v_post   posts_os[] = { post_os, post_prol, NULL };
155 static  v_post   posts_rs[] = { post_rs, 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 };
159 static  v_post   posts_st[] = { post_st, NULL };
160 static  v_post   posts_std[] = { post_std, NULL };
161 static  v_post   posts_text[] = { ewarn_ge1, NULL };
162 static  v_post   posts_text1[] = { ewarn_eq1, NULL };
163 static  v_post   posts_vt[] = { post_vt, NULL };
164 static  v_post   posts_wline[] = { bwarn_ge1, NULL };
165 static  v_pre    pres_an[] = { pre_an, NULL };
166 static  v_pre    pres_bd[] = { pre_display, pre_bd, pre_literal, pre_par, NULL }
167 static  v_pre    pres_bl[] = { pre_bl, pre_par, NULL };
168 static  v_pre    pres_d1[] = { pre_display, NULL };
169 static  v_pre    pres_dl[] = { pre_literal, pre_display, NULL };
170 static  v_pre    pres_dd[] = { pre_dd, NULL };
171 static  v_pre    pres_dt[] = { pre_dt, NULL };
172 static  v_pre    pres_er[] = { NULL, NULL };
173 static  v_pre    pres_fd[] = { NULL, NULL };
174 static  v_pre    pres_it[] = { pre_it, pre_par, NULL };
175 static  v_pre    pres_os[] = { pre_os, NULL };
176 static  v_pre    pres_pp[] = { pre_par, NULL };
177 static  v_pre    pres_sh[] = { pre_sh, NULL };
178 static  v_pre    pres_ss[] = { pre_ss, NULL };
179 static  v_pre    pres_std[] = { pre_std, NULL };

181 static  const struct valids mdoc_valids[MDOC_MAX] = {
182         { NULL, NULL },                         /* Ap */
183         { pres_dd, posts_dd },                  /* Dd */
184         { pres_dt, posts_dt },                  /* Dt */
185         { pres_os, posts_os },                  /* Os */
186         { pres_sh, posts_sh },                  /* Sh */
187         { pres_ss, posts_ss },                  /* Ss */
188         { pres_pp, posts_notext },              /* Pp */
189         { pres_d1, posts_wline },               /* D1 */
190         { pres_dl, posts_dl },                  /* Dl */
191         { pres_bd, posts_bd },                  /* Bd */
192         { NULL, NULL },                         /* Ed */
193         { pres_bl, posts_bl },                  /* Bl */
```



```
194         { NULL, NULL },                         /* El */
195         { pres_it, posts_it },                  /* It */
196         { NULL, NULL },                         /* Ad */
197         { pres_an, posts_an },                  /* An */
198         { NULL, posts_defaults },               /* Ar */
199         { NULL, NULL },                         /* Cd */
200         { NULL, NULL },                         /* Cm */
201         { NULL, NULL },                         /* Dv */
202         { pres_er, NULL },                      /* Er */
203         { NULL, NULL },                         /* Ev */
204         { pres_std, posts_std },                /* Ex */
205         { NULL, NULL },                         /* Fa */
206         { pres_fd, posts_text },                /* Fd */
207         { NULL, NULL },                         /* Fl */
208         { NULL, NULL },                         /* Fn */
209         { NULL, NULL },                         /* Ft */
210         { NULL, NULL },                         /* Ic */
211         { NULL, posts_text1 },                  /* In */
212         { NULL, posts_defaults },               /* Li */
213         { NULL, posts_nd },                     /* Nd */
214         { NULL, posts_nm },                     /* Nm */
215         { NULL, NULL },                         /* Op */
216         { NULL, NULL },                         /* Ot */
217         { NULL, posts_defaults },               /* Pa */
218         { pres_std, posts_std },                /* Rv */
219         { NULL, posts_st },                     /* St */
220         { NULL, NULL },                         /* Va */
221         { NULL, posts_vt },                     /* Vt */
222         { NULL, posts_text },                   /* Xr */
223         { NULL, posts_text },                   /* %A */
224         { NULL, posts_text },                   /* %B */ /* FIXME: can be used o
225         { NULL, posts_text },                   /* %D */
226         { NULL, posts_text },                   /* %I */
227         { NULL, posts_text },                   /* %J */
228         { NULL, posts_text },                   /* %N */
229         { NULL, posts_text },                   /* %O */
230         { NULL, posts_text },                   /* %P */
231         { NULL, posts_text },                   /* %R */
232         { NULL, posts_text },                   /* %T */ /* FIXME: can be used o
233         { NULL, posts_text },                   /* %V */
234         { NULL, NULL },                         /* Ac */
235         { NULL, NULL },                         /* Ao */
236         { NULL, NULL },                         /* Aq */
237         { NULL, posts_at },                     /* At */
238         { NULL, NULL },                         /* Bc */
239         { NULL, posts_bf },                     /* Bf */
240         { NULL, NULL },                         /* Bo */
241         { NULL, NULL },                         /* Bq */
242         { NULL, NULL },                         /* Bsx */
243         { NULL, posts_bx },                     /* Bx */
244         { NULL, posts_bool },                   /* Db */
245         { NULL, NULL },                         /* Dc */
246         { NULL, NULL },                         /* Do */
247         { NULL, NULL },                         /* Dq */
248         { NULL, NULL },                         /* Ec */
249         { NULL, NULL },                         /* Ef */
250         { NULL, NULL },                         /* Em */
251         { NULL, NULL },                         /* Eo */
252         { NULL, NULL },                         /* Fx */
253         { NULL, NULL },                         /* Ms */
254         { NULL, posts_notext },                 /* No */
255         { NULL, posts_ns },                     /* Ns */
256         { NULL, NULL },                         /* Nx */
257         { NULL, NULL },                         /* Ox */
258         { NULL, NULL },                         /* Pc */
259         { NULL, posts_text1 },                  /* Pf */
```

```
260         { NULL, NULL },                         /* Po */
261         { NULL, NULL },                         /* Pq */
262         { NULL, NULL },                         /* Qc */
263         { NULL, NULL },                         /* Ql */
264         { NULL, NULL },                         /* Qo */
265         { NULL, NULL },                         /* Qq */
266         { NULL, NULL },                         /* Re */
267         { NULL, posts_rs },                     /* Rs */
268         { NULL, NULL },                         /* Sc */
269         { NULL, NULL },                         /* So */
270         { NULL, NULL },                         /* Sq */
271         { NULL, posts_bool },                   /* Sm */
272         { NULL, NULL },                         /* Sx */
273         { NULL, NULL },                         /* Sy */
274         { NULL, NULL },                         /* Tn */
275         { NULL, NULL },                         /* Ux */
276         { NULL, NULL },                         /* Xc */
277         { NULL, NULL },                         /* Xo */
278         { NULL, posts_fo },                     /* Fo */
279         { NULL, NULL },                         /* Fc */
280         { NULL, NULL },                         /* Oo */
281         { NULL, NULL },                         /* Oc */
282         { NULL, posts_bk },                     /* Bk */
283         { NULL, NULL },                         /* Ek */
284         { NULL, posts_eoln },                   /* Bt */
285         { NULL, NULL },                         /* Hf */
286         { NULL, NULL },                         /* Fr */
287         { NULL, posts_eoln },                   /* Ud */
288         { NULL, posts_lb },                     /* Lb */
289         { NULL, posts_notext },                 /* Lp */
290         { NULL, NULL },                         /* Lk */
291         { NULL, posts_defaults },               /* Mt */
292         { NULL, NULL },                         /* Brq */
293         { NULL, NULL },                         /* Bro */
294         { NULL, NULL },                         /* Brc */
295         { NULL, posts_text },                   /* %C */
296         { NULL, NULL },                         /* Es */
297         { NULL, NULL },                         /* En */
298         { NULL, NULL },                         /* Dx */
299         { NULL, posts_text },                   /* %Q */
300         { NULL, posts_notext },                 /* br */
301         { pres_pp, posts_sp },                  /* sp */
302         { NULL, posts_text1 },                  /* %U */
303         { NULL, NULL },                         /* Ta */
304 };

306 #define RSORD_MAX 14 /* Number of ‘Rs’ blocks. */

308 static  const enum mdoct rsord[RSORD_MAX] = {
309         MDOC__A,
310         MDOC__T,
311         MDOC__B,
312         MDOC__I,
313         MDOC__J,
314         MDOC__R,
315         MDOC__N,
316         MDOC__V,
317         MDOC__P,
318         MDOC__Q,
319         MDOC__D,
320         MDOC__O,
321         MDOC__C,
322         MDOC__U
323 };

325 static  const char * const secnames[SEC__MAX] = {
```



```
326         NULL,
327         "NAME",
328         "LIBRARY",
329         "SYNOPSIS",
330         "DESCRIPTION",
331         "IMPLEMENTATION NOTES",
332         "RETURN VALUES",
333         "ENVIRONMENT",
334         "FILES",
335         "EXIT STATUS",
336         "EXAMPLES",
337         "DIAGNOSTICS",
338         "COMPATIBILITY",
339         "ERRORS",
340         "SEE ALSO",
341         "STANDARDS",
342         "HISTORY",
343         "AUTHORS",
344         "CAVEATS",
345         "BUGS",
346         "SECURITY CONSIDERATIONS",
347         NULL
348 };

350 int
351 mdoc_valid_pre(struct mdoc *mdoc, struct mdoc_node *n)
352 {
353         v_pre           *p;
354         int              line, pos;
355         char            *tp;

357         switch (n->type) {
358         case (MDOC_TEXT):
359                 tp = n->string;
360                 line = n->line;
361                 pos = n->pos;
362                 check_text(mdoc, line, pos, tp);
363                 /* FALLTHROUGH */
364         case (MDOC_TBL):
365                 /* FALLTHROUGH */
366         case (MDOC_EQN):
367                 /* FALLTHROUGH */
368         case (MDOC_ROOT):
369                 return(1);
370         default:
371                 break;
372         }

374         check_args(mdoc, n);

376         if (NULL == mdoc_valids[n->tok].pre)
377                 return(1);
378         for (p = mdoc_valids[n->tok].pre; *p; p++)
379                 if ( ! (*p)(mdoc, n))
380                         return(0);
381         return(1);
382 }


385 int
386 mdoc_valid_post(struct mdoc *mdoc)
387 {
388         v_post          *p;

390         if (MDOC_VALID & mdoc->last->flags)
391                 return(1);
```

```
392         mdoc->last->flags |= MDOC_VALID;

394         switch (mdoc->last->type) {
395         case (MDOC_TEXT):
396                 /* FALLTHROUGH */
397         case (MDOC_EQN):
398                 /* FALLTHROUGH */
399         case (MDOC_TBL):
400                 return(1);
401         case (MDOC_ROOT):
402                 return(post_root(mdoc));
403         default:
404                 break;
405         }

407         if (NULL == mdoc_valids[mdoc->last->tok].post)
408                 return(1);
409         for (p = mdoc_valids[mdoc->last->tok].post; *p; p++)
410                 if ( ! (*p)(mdoc))
411                         return(0);

413         return(1);
414 }

416 static int
417 check_count(struct mdoc *m, enum mdoc_type type,
418                 enum check_lvl lvl, enum check_ineq ineq, int val)
419 {
420         const char      *p;
421         enum mandocerr   t;

423         if (m->last->type != type)
424                 return(1);
425
426         switch (ineq) {
427         case (CHECK_LT):
428                 p = "less than ";
429                 if (m->last->nchild < val)
430                         return(1);
431                 break;
432         case (CHECK_GT):
433                 p = "more than ";
434                 if (m->last->nchild > val)
435                         return(1);
436                 break;
437         case (CHECK_EQ):
438                 p = "";
439                 if (val == m->last->nchild)
440                         return(1);
441                 break;
442         default:
443                 abort();
444                 /* NOTREACHED */
445         }

447         t = lvl == CHECK_WARN ? MANDOCERR_ARGCWARN : MANDOCERR_ARGCOUNT;
448         mandoc_vmsg(t, m->parse, m->last->line, m->last->pos,
449                         "want %s%d children (have %d)",
450                         p, val, m->last->nchild);
451         return(1);
452 }

454 static int
455 berr_ge1(POST_ARGS)
456 {
```



```
458         return(check_count(mdoc, MDOC_BODY, CHECK_ERROR, CHECK_GT, 0));
459 }

461 static int
462 bwarn_ge1(POST_ARGS)
463 {
464         return(check_count(mdoc, MDOC_BODY, CHECK_WARN, CHECK_GT, 0));
465 }

467 static int
468 ewarn_eq0(POST_ARGS)
469 {
470         return(check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 0));
471 }

473 static int
474 ewarn_eq1(POST_ARGS)
475 {
476         return(check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 1));
477 }

479 static int
480 ewarn_ge1(POST_ARGS)
481 {
482         return(check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_GT, 0));
483 }

485 static int
486 ewarn_le1(POST_ARGS)
487 {
488         return(check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_LT, 2));
489 }

491 static int
492 hwarn_eq0(POST_ARGS)
493 {
494         return(check_count(mdoc, MDOC_HEAD, CHECK_WARN, CHECK_EQ, 0));
495 }

497 static int
498 hwarn_eq1(POST_ARGS)
499 {
500         return(check_count(mdoc, MDOC_HEAD, CHECK_WARN, CHECK_EQ, 1));
501 }

503 static int
504 hwarn_ge1(POST_ARGS)
505 {
506         return(check_count(mdoc, MDOC_HEAD, CHECK_WARN, CHECK_GT, 0));
507 }

509 static int
510 hwarn_le1(POST_ARGS)
511 {
512         return(check_count(mdoc, MDOC_HEAD, CHECK_WARN, CHECK_LT, 2));
513 }

515 static void
516 check_args(struct mdoc *m, struct mdoc_node *n)
517 {
518         int              i;

520         if (NULL == n->args)
521                 return;

523         assert(n->args->argc);
```

```
524         for (i = 0; i < (int)n->args->argc; i++)
525                 check_argv(m, n, &n->args->argv[i]);
526 }

528 static void
529 check_argv(struct mdoc *m, struct mdoc_node *n, struct mdoc_argv *v)
530 {
531         int              i;

533         for (i = 0; i < (int)v->sz; i++)
534                 check_text(m, v->line, v->pos, v->value[i]);

536         /* FIXME: move to post_std(). */

538         if (MDOC_Std == v->arg)
539                 if ( ! (v->sz || m->meta.name))
540                         mdoc_nmsg(m, n, MANDOCERR_NONAME);
541 }

543 static void
544 check_text(struct mdoc *m, int ln, int pos, char *p)
545 {
546         char            *cp;

548         if (MDOC_LITERAL & m->flags)
549                 return;

551         for (cp = p; NULL != (p = strchr(p, '\t')); p++)
552                 mdoc_pmsg(m, ln, pos + (int)(p - cp), MANDOCERR_BADTAB);
553 }

555 static int
556 check_parent(PRE_ARGS, enum mdoct tok, enum mdoc_type t)
557 {

559         assert(n->parent);
560         if ((MDOC_ROOT == t || tok == n->parent->tok) &&
561                         (t == n->parent->type))
562                 return(1);

564         mandoc_vmsg(MANDOCERR_SYNTCHILD, mdoc->parse, n->line,
565                         n->pos, "want parent %s", MDOC_ROOT == t ?
566                         "<root>" : mdoc_macronames[tok]);
567         return(0);
568 }


571 static int
572 pre_display(PRE_ARGS)
573 {
574         struct mdoc_node *node;

576         if (MDOC_BLOCK != n->type)
577                 return(1);

579         for (node = mdoc->last->parent; node; node = node->parent)
580                 if (MDOC_BLOCK == node->type)
581                         if (MDOC_Bd == node->tok)
582                                 break;

584         if (node)
585                 mdoc_nmsg(mdoc, n, MANDOCERR_NESTEDDISP);

587         return(1);
588 }
```



```
591 static int
592 pre_bl(PRE_ARGS)
593 {
594         int               i, comp, dup;
595         const char       *offs, *width;
596         enum mdoc_list    lt;
597         struct mdoc_node *np;

599         if (MDOC_BLOCK != n->type) {
600                 if (ENDBODY_NOT != n->end) {
601                         assert(n->pending);
602                         np = n->pending->parent;
603                 } else
604                         np = n->parent;

606                 assert(np);
607                 assert(MDOC_BLOCK == np->type);
608                 assert(MDOC_Bl == np->tok);
609                 return(1);
610         }

612         /*
613          * First figure out which kind of list to use: bind ourselves to
614          * the first mentioned list type and warn about any remaining
615          * ones.  If we find no list type, we default to LIST_item.
616          */

618         /* LINTED */
619         for (i = 0; n->args && i < (int)n->args->argc; i++) {
620                 lt = LIST__NONE;
621                 dup = comp = 0;
622                 width = offs = NULL;
623                 switch (n->args->argv[i].arg) {
624                 /* Set list types. */
625                 case (MDOC_Bullet):
626                         lt = LIST_bullet;
627                         break;
628                 case (MDOC_Dash):
629                         lt = LIST_dash;
630                         break;
631                 case (MDOC_Enum):
632                         lt = LIST_enum;
633                         break;
634                 case (MDOC_Hyphen):
635                         lt = LIST_hyphen;
636                         break;
637                 case (MDOC_Item):
638                         lt = LIST_item;
639                         break;
640                 case (MDOC_Tag):
641                         lt = LIST_tag;
642                         break;
643                 case (MDOC_Diag):
644                         lt = LIST_diag;
645                         break;
646                 case (MDOC_Hang):
647                         lt = LIST_hang;
648                         break;
649                 case (MDOC_Ohang):
650                         lt = LIST_ohang;
651                         break;
652                 case (MDOC_Inset):
653                         lt = LIST_inset;
654                         break;
655                 case (MDOC_Column):
```

```
656                     lt = LIST_column;
657                     break;
658             /* Set list arguments. */
659             case (MDOC_Compact):
660                     dup = n->norm->Bl.comp;
661                     comp = 1;
662                     break;
663             case (MDOC_Width):
664                     /* NB: this can be empty! */
665                     if (n->args->argv[i].sz) {
666                             width = n->args->argv[i].value[0];
667                             dup = (NULL != n->norm->Bl.width);
668                             break;
669                     }
670                     mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
671                     break;
672             case (MDOC_Offset):
673                     /* NB: this can be empty! */
674                     if (n->args->argv[i].sz) {
675                             offs = n->args->argv[i].value[0];
676                             dup = (NULL != n->norm->Bl.offs);
677                             break;
678                     }
679                     mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
680                     break;
681             default:
682                     continue;
683             }

685             /* Check: duplicate auxiliary arguments. */

687             if (dup)
688                     mdoc_nmsg(mdoc, n, MANDOCERR_ARGVREP);

690             if (comp && ! dup)
691                     n->norm->Bl.comp = comp;
692             if (offs && ! dup)
693                     n->norm->Bl.offs = offs;
694             if (width && ! dup)
695                     n->norm->Bl.width = width;

697             /* Check: multiple list types. */

699             if (LIST__NONE != lt && n->norm->Bl.type != LIST__NONE)
700                     mdoc_nmsg(mdoc, n, MANDOCERR_LISTREP);

702             /* Assign list type. */

704             if (LIST__NONE != lt && n->norm->Bl.type == LIST__NONE) {
705                     n->norm->Bl.type = lt;
706                     /* Set column information, too. */
707                     if (LIST_column == lt) {
708                             n->norm->Bl.ncols =
709                                     n->args->argv[i].sz;
710                             n->norm->Bl.cols = (void *)
711                                     n->args->argv[i].value;
712                     }
713             }

715             /* The list type should come first. */

717             if (n->norm->Bl.type == LIST__NONE)
718                     if (n->norm->Bl.width ||
719                             n->norm->Bl.offs ||
720                             n->norm->Bl.comp)
721                             mdoc_nmsg(mdoc, n, MANDOCERR_LISTFIRST);
```



```
723                     continue;
724             }

726             /* Allow lists to default to LIST_item. */

728             if (LIST__NONE == n->norm->Bl.type) {
729                     mdoc_nmsg(mdoc, n, MANDOCERR_LISTTYPE);
730                     n->norm->Bl.type = LIST_item;
731             }

733             /*
734              * Validate the width field.  Some list types don't need width
735              * types and should be warned about them.  Others should have it
736              * and must also be warned.
737              */

739             switch (n->norm->Bl.type) {
740             case (LIST_tag):
741                     if (n->norm->Bl.width)
742                             break;
743                     mdoc_nmsg(mdoc, n, MANDOCERR_NOWIDTHARG);
744                     break;
745             case (LIST_column):
746                     /* FALLTHROUGH */
747             case (LIST_diag):
748                     /* FALLTHROUGH */
749             case (LIST_ohang):
750                     /* FALLTHROUGH */
751             case (LIST_inset):
752                     /* FALLTHROUGH */
753             case (LIST_item):
754                     if (n->norm->Bl.width)
755                             mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
756                     break;
757             default:
758                     break;
759             }

761             return(1);
762 }


765 static int
766 pre_bd(PRE_ARGS)
767 {
768             int               i, dup, comp;
769             enum mdoc_disp    dt;
770             const char       *offs;
771             struct mdoc_node *np;

773             if (MDOC_BLOCK != n->type) {
774                     if (ENDBODY_NOT != n->end) {
775                             assert(n->pending);
776                             np = n->pending->parent;
777                     } else
778                             np = n->parent;

780                     assert(np);
781                     assert(MDOC_BLOCK == np->type);
782                     assert(MDOC_Bd == np->tok);
783                     return(1);
784             }

786             /* LINTED */
787             for (i = 0; n->args && i < (int)n->args->argc; i++) {
```

```
788                 dt = DISP__NONE;
789                 dup = comp = 0;
790                 offs = NULL;

792                 switch (n->args->argv[i].arg) {
793                 case (MDOC_Centred):
794                         dt = DISP_centred;
795                         break;
796                 case (MDOC_Ragged):
797                         dt = DISP_ragged;
798                         break;
799                 case (MDOC_Unfilled):
800                         dt = DISP_unfilled;
801                         break;
802                 case (MDOC_Filled):
803                         dt = DISP_filled;
804                         break;
805                 case (MDOC_Literal):
806                         dt = DISP_literal;
807                         break;
808                 case (MDOC_File):
809                         mdoc_nmsg(mdoc, n, MANDOCERR_BADDISP);
810                         return(0);
811                 case (MDOC_Offset):
812                         /* NB: this can be empty! */
813                         if (n->args->argv[i].sz) {
814                                 offs = n->args->argv[i].value[0];
815                                 dup = (NULL != n->norm->Bd.offs);
816                                 break;
817                         }
818                         mdoc_nmsg(mdoc, n, MANDOCERR_IGNARGV);
819                         break;
820                 case (MDOC_Compact):
821                         comp = 1;
822                         dup = n->norm->Bd.comp;
823                         break;
824                 default:
825                         abort();
826                         /* NOTREACHED */
827                 }

829                 /* Check whether we have duplicates. */

831                 if (dup)
832                         mdoc_nmsg(mdoc, n, MANDOCERR_ARGVREP);

834                 /* Make our auxiliary assignments. */

836                 if (offs && ! dup)
837                         n->norm->Bd.offs = offs;
838                 if (comp && ! dup)
839                         n->norm->Bd.comp = comp;

841                 /* Check whether a type has already been assigned. */

843                 if (DISP__NONE != dt && n->norm->Bd.type != DISP__NONE)
844                         mdoc_nmsg(mdoc, n, MANDOCERR_DISPREP);

846                 /* Make our type assignment. */

848                 if (DISP__NONE != dt && n->norm->Bd.type == DISP__NONE)
849                         n->norm->Bd.type = dt;
850         }

852         if (DISP__NONE == n->norm->Bd.type) {
853                 mdoc_nmsg(mdoc, n, MANDOCERR_DISPTYPE);
```



```
854                 n->norm->Bd.type = DISP_ragged;
855         }

857         return(1);
858 }


861 static int
862 pre_ss(PRE_ARGS)
863 {

865         if (MDOC_BLOCK != n->type)
866                 return(1);
867         return(check_parent(mdoc, n, MDOC_Sh, MDOC_BODY));
868 }


871 static int
872 pre_sh(PRE_ARGS)
873 {

875         if (MDOC_BLOCK != n->type)
876                 return(1);

878         roff_regunset(mdoc->roff, REG_nS);
879         return(check_parent(mdoc, n, MDOC_MAX, MDOC_ROOT));
880 }


883 static int
884 pre_it(PRE_ARGS)
885 {

887         if (MDOC_BLOCK != n->type)
888                 return(1);

890         return(check_parent(mdoc, n, MDOC_Bl, MDOC_BODY));
891 }


894 static int
895 pre_an(PRE_ARGS)
896 {
897         int              i;

899         if (NULL == n->args)
900                 return(1);
901
902         for (i = 1; i < (int)n->args->argc; i++)
903                 mdoc_pmsg(mdoc, n->args->argv[i].line,
904                         n->args->argv[i].pos, MANDOCERR_IGNARGV);

906         if (MDOC_Split == n->args->argv[0].arg)
907                 n->norm->An.auth = AUTH_split;
908         else if (MDOC_Nosplit == n->args->argv[0].arg)
909                 n->norm->An.auth = AUTH_nosplit;
910         else
911                 abort();

913         return(1);
914 }

916 static int
917 pre_std(PRE_ARGS)
918 {
```

```
920         if (n->args && 1 == n->args->argc)
921                 if (MDOC_Std == n->args->argv[0].arg)
922                         return(1);
923
924         mdoc_nmsg(mdoc, n, MANDOCERR_NOARGV);
925         return(1);
926 }
927
928 static int
929 pre_dt(PRE_ARGS)
930 {
931
932         if (NULL == mdoc->meta.date || mdoc->meta.os)
933                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGOOO);
934
935         if (mdoc->meta.title)
936                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGREP);
937
938         return(1);
939 }
940
941 static int
942 pre_os(PRE_ARGS)
943 {
944
945         if (NULL == mdoc->meta.title || NULL == mdoc->meta.date)
946                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGOOO);
947
948         if (mdoc->meta.os)
949                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGREP);
950
951         return(1);
952 }
953
954 static int
955 pre_dd(PRE_ARGS)
956 {
957
958         if (mdoc->meta.title || mdoc->meta.os)
959                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGOOO);
960
961         if (mdoc->meta.date)
962                 mdoc_nmsg(mdoc, n, MANDOCERR_PROLOGREP);
963
964         return(1);
965 }
966
967
968 static int
969 post_bf(POST_ARGS)
970 {
971         struct mdoc_node *np;
972         enum mdocargt     arg;
973
974         /*
975          * Unlike other data pointers, these are "housed" by the HEAD
976          * element, which contains the goods.
977          */
978
979         if (MDOC_HEAD != mdoc->last->type) {
980                 if (ENDBODY_NOT != mdoc->last->end) {
981                         assert(mdoc->last->pending);
982                         np = mdoc->last->pending->parent->head;
983                 } else if (MDOC_BLOCK != mdoc->last->type) {
984                         np = mdoc->last->parent->head;
985                 } else
```



```
986                         np = mdoc->last->head;
987
988                 assert(np);
989                 assert(MDOC_HEAD == np->type);
990                 assert(MDOC_Bf == np->tok);
991                 return(1);
992         }
993
994         np = mdoc->last;
995         assert(MDOC_BLOCK == np->parent->type);
996         assert(MDOC_Bf == np->parent->tok);
997
998         /*
999          * Cannot have both argument and parameter.
1000         * If neither is specified, let it through with a warning.
1001         */
1002
1003        if (np->parent->args && np->child) {
1004                mdoc_nmsg(mdoc, np, MANDOCERR_SYNTARGVCOUNT);
1005                return(0);
1006        } else if (NULL == np->parent->args && NULL == np->child) {
1007                mdoc_nmsg(mdoc, np, MANDOCERR_FONTTYPE);
1008                return(1);
1009        }
1010
1011        /* Extract argument into data. */
1012
1013        if (np->parent->args) {
1014                arg = np->parent->args->argv[0].arg;
1015                if (MDOC_Emphasis == arg)
1016                        np->norm->Bf.font = FONT_Em;
1017                else if (MDOC_Literal == arg)
1018                        np->norm->Bf.font = FONT_Li;
1019                else if (MDOC_Symbolic == arg)
1020                        np->norm->Bf.font = FONT_Sy;
1021                else
1022                        abort();
1023                return(1);
1024        }
1025
1026        /* Extract parameter into data. */
1027
1028        if (0 == strcmp(np->child->string, "Em"))
1029                np->norm->Bf.font = FONT_Em;
1030        else if (0 == strcmp(np->child->string, "Li"))
1031                np->norm->Bf.font = FONT_Li;
1032        else if (0 == strcmp(np->child->string, "Sy"))
1033                np->norm->Bf.font = FONT_Sy;
1034        else
1035                mdoc_nmsg(mdoc, np, MANDOCERR_FONTTYPE);
1036
1037        return(1);
1038 }
1039
1040 static int
1041 post_lb(POST_ARGS)
1042 {
1043         const char      *p;
1044         char            *buf;
1045         size_t           sz;
1046
1047         check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 1);
1048
1049         assert(mdoc->last->child);
1050         assert(MDOC_TEXT == mdoc->last->child->type);
```

```
1052         p = mdoc_a2lib(mdoc->last->child->string);

1054         /* If lookup ok, replace with table value. */

1056         if (p) {
1057                 free(mdoc->last->child->string);
1058                 mdoc->last->child->string = mandoc_strdup(p);
1059                 return(1);
1060         }

1062         /* If not, use "library ‘‘xxxx’’. */

1064         sz = strlen(mdoc->last->child->string) +
1065                 2 + strlen("\\(lqlibrary\\(rq");
1066         buf = mandoc_malloc(sz);
1067         snprintf(buf, sz, "library \\(lq%s\\(rq",
1068                         mdoc->last->child->string);
1069         free(mdoc->last->child->string);
1070         mdoc->last->child->string = buf;
1071         return(1);
1072 }

1074 static int
1075 post_eoln(POST_ARGS)
1076 {

1078         if (mdoc->last->child)
1079                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_ARGSLOST);
1080         return(1);
1081 }


1084 static int
1085 post_vt(POST_ARGS)
1086 {
1087         const struct mdoc_node *n;

1089         /*
1090          * The Vt macro comes in both ELEM and BLOCK form, both of which
1091          * have different syntaxes (yet more context-sensitive
1092          * behaviour).  ELEM types must have a child, which is already
1093          * guaranteed by the in_line parsing routine; BLOCK types,
1094          * specifically the BODY, should only have TEXT children.
1095          */

1097         if (MDOC_BODY != mdoc->last->type)
1098                 return(1);
1099
1100         for (n = mdoc->last->child; n; n = n->next)
1101                 if (MDOC_TEXT != n->type)
1102                         mdoc_nmsg(mdoc, n, MANDOCERR_CHILD);

1104         return(1);
1105 }


1108 static int
1109 post_nm(POST_ARGS)
1110 {
1111         char             buf[BUFSIZ];
1112         int              c;

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);
```



```
1118                 return(1);
1119         } else if (mdoc->meta.name)
1120                 return(1);

1122         /* If no meta name, set it from the child. */

1124         buf[0] = ’\0’;
1125         if (-1 == (c = concat(buf, mdoc->last->child, BUFSIZ))) {
1126                 mdoc_nmsg(mdoc, mdoc->last->child, MANDOCERR_MEM);
1127                 return(0);
1128         }

1130         assert(c);
1131         mdoc->meta.name = mandoc_strdup(buf);
1132         return(1);
1133 }

1135 static int
1136 post_literal(POST_ARGS)
1137 {
1138
1139         /*
1140          * The ‘Dl’ (note "el" not "one") and ‘Bd’ macros unset the
1141          * MDOC_LITERAL flag as they leave.  Note that ‘Bd’ only sets
1142          * this in literal mode, but it doesn’t hurt to just switch it
1143          * off in general since displays can’t be nested.
1144          */

1146         if (MDOC_BODY == mdoc->last->type)
1147                 mdoc->flags &= ~MDOC_LITERAL;

1149         return(1);
1150 }

1152 static int
1153 post_defaults(POST_ARGS)
1154 {
1155         struct mdoc_node *nn;

1157         /*
1158          * The ‘Ar’ defaults to "file ..." if no value is provided as an
1159          * argument; the ‘Mt’ and ‘Pa’ macros use "~"; the ‘Li’ just
1160          * gets an empty string.
1161          */

1163         if (mdoc->last->child)
1164                 return(1);
1165
1166         nn = mdoc->last;
1167         mdoc->next = MDOC_NEXT_CHILD;

1169         switch (nn->tok) {
1170         case (MDOC_Ar):
1171                 if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, "file"))
1172                         return(0);
1173                 if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, "..."))
1174                         return(0);
1175                 break;
1176         case (MDOC_At):
1177                 if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, "AT&T"))
1178                         return(0);
1179                 if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, "UNIX"))
1180                         return(0);
1181                 break;
1182         case (MDOC_Li):
1183                 if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, ""))
```

```
1184                        return(0);
1185                break;
1186        case (MDOC_Pa):
1187                /* FALLTHROUGH */
1188        case (MDOC_Mt):
1189                if ( ! mdoc_word_alloc(mdoc, nn->line, nn->pos, "~"))
1190                        return(0);
1191                break;
1192        default:
1193                abort();
1194                /* NOTREACHED */
1195        }

1197        mdoc->last = nn;
1198        return(1);
1199 }

1201 static int
1202 post_at(POST_ARGS)
1203 {
1204        const char       *p, *q;
1205        char             *buf;
1206        size_t            sz;

1208        /*
1209         * If we have a child, look it up in the standard keys.  If a
1210         * key exist, use that instead of the child; if it doesn't,
1211         * prefix "AT&T UNIX " to the existing data.
1212         */
1213
1214        if (NULL == mdoc->last->child)
1215                return(1);

1217        assert(MDOC_TEXT == mdoc->last->child->type);
1218        p = mdoc_a2att(mdoc->last->child->string);

1220        if (p) {
1221                free(mdoc->last->child->string);
1222                mdoc->last->child->string = mandoc_strdup(p);
1223        } else {
1224                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADATT);
1225                p = "AT&T UNIX ";
1226                q = mdoc->last->child->string;
1227                sz = strlen(p) + strlen(q) + 1;
1228                buf = mandoc_malloc(sz);
1229                strlcpy(buf, p, sz);
1230                strlcat(buf, q, sz);
1231                free(mdoc->last->child->string);
1232                mdoc->last->child->string = buf;
1233        }

1235        return(1);
1236 }

1238 static int
1239 post_an(POST_ARGS)
1240 {
1241        struct mdoc_node *np;

1243        np = mdoc->last;
1244        if (AUTH__NONE == np->norm->An.auth) {
1245                if (0 == np->child)
1246                        check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_GT, 0);
1247        } else if (np->child)
1248                check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 0);
```



```
1250        return(1);
1251 }


1254 static int
1255 post_it(POST_ARGS)
1256 {
1257        int                i, cols;
1258        enum mdoc_list     lt;
1259        struct mdoc_node *n, *c;
1260        enum mandocerr     er;

1262        if (MDOC_BLOCK != mdoc->last->type)
1263                return(1);

1265        n = mdoc->last->parent->parent;
1266        lt = n->norm->Bl.type;

1268        if (LIST__NONE == lt) {
1269                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_LISTTYPE);
1270                return(1);
1271        }

1273        switch (lt) {
1274        case (LIST_tag):
1275                if (mdoc->last->head->child)
1276                        break;
1277                /* FIXME: give this a dummy value. */
1278                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NOARGS);
1279                break;
1280        case (LIST_hang):
1281                /* FALLTHROUGH */
1282        case (LIST_ohang):
1283                /* FALLTHROUGH */
1284        case (LIST_inset):
1285                /* FALLTHROUGH */
1286        case (LIST_diag):
1287                if (NULL == mdoc->last->head->child)
1288                        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NOARGS);
1289                break;
1290        case (LIST_bullet):
1291                /* FALLTHROUGH */
1292        case (LIST_dash):
1293                /* FALLTHROUGH */
1294        case (LIST_enum):
1295                /* FALLTHROUGH */
1296        case (LIST_hyphen):
1297                if (NULL == mdoc->last->body->child)
1298                        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NOBODY);
1299                /* FALLTHROUGH */
1300        case (LIST_item):
1301                if (mdoc->last->head->child)
1302                        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_ARGSLOST);
1303                break;
1304        case (LIST_column):
1305                cols = (int)n->norm->Bl.ncols;

1307                assert(NULL == mdoc->last->head->child);

1309                if (NULL == mdoc->last->body->child)
1310                        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NOBODY);

1312                for (i = 0, c = mdoc->last->child; c; c = c->next)
1313                        if (MDOC_BODY == c->type)
1314                                i++;
```

```
1316                if (i < cols)
1317                        er = MANDOCERR_ARGCOUNT;
1318                else if (i == cols || i == cols + 1)
1319                        break;
1320                else
1321                        er = MANDOCERR_SYNTARGCOUNT;

1323                mandoc_vmsg(er, mdoc->parse, mdoc->last->line,
1324                                mdoc->last->pos,
1325                                "columns == %d (have %d)", cols, i);
1326                return(MANDOCERR_ARGCOUNT == er);
1327        default:
1328                break;
1329        }

1331        return(1);
1332 }

1334 static int
1335 post_bl_block(POST_ARGS)
1336 {
1337        struct mdoc_node *n;

1339        /*
1340         * These are fairly complicated, so we've broken them into two
1341         * functions.  post_bl_block_tag() is called when a -tag is
1342         * specified, but no -width (it must be guessed).  The second
1343         * when a -width is specified (macro indicators must be
1344         * rewritten into real lengths).
1345         */

1347        n = mdoc->last;

1349        if (LIST_tag == n->norm->Bl.type &&
1350                        NULL == n->norm->Bl.width) {
1351                if ( ! post_bl_block_tag(mdoc))
1352                        return(0);
1353        } else if (NULL != n->norm->Bl.width) {
1354                if ( ! post_bl_block_width(mdoc))
1355                        return(0);
1356        } else
1357                return(1);

1359        assert(n->norm->Bl.width);
1360        return(1);
1361 }

1363 static int
1364 post_bl_block_width(POST_ARGS)
1365 {
1366        size_t            width;
1367        int               i;
1368        enum mdoct        tok;
1369        struct mdoc_node *n;
1370        char              buf[NUMSIZ];

1372        n = mdoc->last;

1374        /*
1375         * Calculate the real width of a list from the -width string,
1376         * which may contain a macro (with a known default width), a
1377         * literal string, or a scaling width.
1378         *
1379         * If the value to -width is a macro, then we re-write it to be
1380         * the macro's width as set in share/tmac/mdoc/doc-common.
1381         */
```



```
1383        if (0 == strcmp(n->norm->Bl.width, "Ds"))
1384                width = 6;
1385        else if (MDOC_MAX == (tok = mdoc_hash_find(n->norm->Bl.width)))
1386                return(1);
1387        else if (0 == (width = macro2len(tok)))  {
1388                mdoc_nmsg(mdoc, n, MANDOCERR_BADWIDTH);
1389                return(1);
1390        }

1392        /* The value already exists: free and reallocate it. */

1394        assert(n->args);

1396        for (i = 0; i < (int)n->args->argc; i++)
1397                if (MDOC_Width == n->args->argv[i].arg)
1398                        break;

1400        assert(i < (int)n->args->argc);

1402        snprintf(buf, NUMSIZ, "%un", (unsigned int)width);
1403        free(n->args->argv[i].value[0]);
1404        n->args->argv[i].value[0] = mandoc_strdup(buf);

1406        /* Set our width! */
1407        n->norm->Bl.width = n->args->argv[i].value[0];
1408        return(1);
1409 }

1411 static int
1412 post_bl_block_tag(POST_ARGS)
1413 {
1414        struct mdoc_node *n, *nn;
1415        size_t            sz, ssz;
1416        int               i;
1417        char              buf[NUMSIZ];

1419        /*
1420         * Calculate the -width for a 'Bl -tag' list if it hasn't been
1421         * provided.  Uses the first head macro.  NOTE AGAIN: this is
1422         * ONLY if the -width argument has NOT been provided.  See
1423         * post_bl_block_width() for converting the -width string.
1424         */

1426        sz = 10;
1427        n = mdoc->last;

1429        for (nn = n->body->child; nn; nn = nn->next) {
1430                if (MDOC_It != nn->tok)
1431                        continue;

1433                assert(MDOC_BLOCK == nn->type);
1434                nn = nn->head->child;

1436                if (nn == NULL)
1437                        break;

1439                if (MDOC_TEXT == nn->type) {
1440                        sz = strlen(nn->string) + 1;
1441                        break;
1442                }

1444                if (0 != (ssz = macro2len(nn->tok)))
1445                        sz = ssz;

1447                break;
```

```
1448         }

1450         /* Defaults to ten ens. */

1452         snprintf(buf, NUMSIZ, "%un", (unsigned int)sz);

1454         /*
1455          * We have to dynamically add this to the macro's argument list.
1456          * We're guaranteed that a MDOC_Width doesn't already exist.
1457          */

1459         assert(n->args);
1460         i = (int)(n->args->argc)++;

1462         n->args->argv = mandoc_realloc(n->args->argv,
1463                         n->args->argc * sizeof(struct mdoc_argv));

1465         n->args->argv[i].arg = MDOC_Width;
1466         n->args->argv[i].line = n->line;
1467         n->args->argv[i].pos = n->pos;
1468         n->args->argv[i].sz = 1;
1469         n->args->argv[i].value = mandoc_malloc(sizeof(char *));
1470         n->args->argv[i].value[0] = mandoc_strdup(buf);

1472         /* Set our width! */
1473         n->norm->Bl.width = n->args->argv[i].value[0];
1474         return(1);
1475 }


1478 static int
1479 post_bl_head(POST_ARGS)
1480 {
1481         struct mdoc_node *np, *nn, *nnp;
1482         int               i, j;

1484         if (LIST_column != mdoc->last->norm->Bl.type)
1485                 /* FIXME: this should be ERROR class... */
1486                 return(hwarn_eq0(mdoc));

1488         /*
1489          * Convert old-style lists, where the column width specifiers
1490          * trail as macro parameters, to the new-style ("normal-form")
1491          * lists where they're argument values following -column.
1492          */

1494         /* First, disallow both types and allow normal-form. */

1496         /*
1497          * TODO: technically, we can accept both and just merge the two
1498          * lists, but I'll leave that for another day.
1499          */

1501         if (mdoc->last->norm->Bl.ncols && mdoc->last->nchild) {
1502                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_COLUMNS);
1503                 return(0);
1504         } else if (NULL == mdoc->last->child)
1505                 return(1);

1507         np = mdoc->last->parent;
1508         assert(np->args);

1510         for (j = 0; j < (int)np->args->argc; j++)
1511                 if (MDOC_Column == np->args->argv[j].arg)
1512                         break;
```



```
1514         assert(j < (int)np->args->argc);
1515         assert(0 == np->args->argv[j].sz);

1517         /*
1518          * Accommodate for new-style groff column syntax.  Shuffle the
1519          * child nodes, all of which must be TEXT, as arguments for the
1520          * column field.  Then, delete the head children.
1521          */

1523         np->args->argv[j].sz = (size_t)mdoc->last->nchild;
1524         np->args->argv[j].value = mandoc_malloc
1525                 ((size_t)mdoc->last->nchild * sizeof(char *));

1527         mdoc->last->norm->Bl.ncols = np->args->argv[j].sz;
1528         mdoc->last->norm->Bl.cols = (void *)np->args->argv[j].value;

1530         for (i = 0, nn = mdoc->last->child; nn; i++) {
1531                 np->args->argv[j].value[i] = nn->string;
1532                 nn->string = NULL;
1533                 nnp = nn;
1534                 nn = nn->next;
1535                 mdoc_node_delete(NULL, nnp);
1536         }

1538         mdoc->last->nchild = 0;
1539         mdoc->last->child = NULL;

1541         return(1);
1542 }

1544 static int
1545 post_bl(POST_ARGS)
1546 {
1547         struct mdoc_node        *n;

1549         if (MDOC_HEAD == mdoc->last->type)
1550                 return(post_bl_head(mdoc));
1551         if (MDOC_BLOCK == mdoc->last->type)
1552                 return(post_bl_block(mdoc));
1553         if (MDOC_BODY != mdoc->last->type)
1554                 return(1);

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):
1566                         continue;
1567                 default:
1568                         break;
1569                 }

1571                 mdoc_nmsg(mdoc, n, MANDOCERR_SYNTCHILD);
1572                 return(0);
1573         }

1575         return(1);
1576 }

1578 static int
1579 ebool(struct mdoc *mdoc)
```

```
{

        if (NULL == mdoc->last->child) {
                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_MACROEMPTY);
                mdoc_node_delete(mdoc, mdoc->last);
                return(1);
        }
        check_count(mdoc, MDOC_ELEM, CHECK_WARN, CHECK_EQ, 1);

        assert(MDOC_TEXT == mdoc->last->child->type);

        if (0 == strcmp(mdoc->last->child->string, "on"))
                return(1);
        if (0 == strcmp(mdoc->last->child->string, "off"))
                return(1);

        mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADBOOL);
        return(1);
}

static int
post_root(POST_ARGS)
{
        int              erc;
        struct mdoc_node *n;

        erc = 0;

        /* Check that we have a finished prologue. */

        if ( ! (MDOC_PBODY & mdoc->flags)) {
                erc++;
                mdoc_nmsg(mdoc, mdoc->first, MANDOCERR_NODOCPROLOG);
        }

        n = mdoc->first;
        assert(n);

        /* Check that we begin with a proper ‘Sh’. */

        if (NULL == n->child) {
                erc++;
                mdoc_nmsg(mdoc, n, MANDOCERR_NODOCBODY);
        } else if (MDOC_BLOCK != n->child->type ||
                        MDOC_Sh != n->child->tok) {
                erc++;
                /* Can this be lifted?  See rxdebug.1 for example. */
                mdoc_nmsg(mdoc, n, MANDOCERR_NODOCBODY);
        }

        return(erc ? 0 : 1);
}

static int
post_st(POST_ARGS)
{
        struct mdoc_node        *ch;
        const char              *p;

        if (NULL == (ch = mdoc->last->child)) {
                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_MACROEMPTY);
                mdoc_node_delete(mdoc, mdoc->last);
                return(1);
        }

        assert(MDOC_TEXT == ch->type);
```



```
        if (NULL == (p = mdoc_a2st(ch->string))) {
                mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADSTANDARD);
                mdoc_node_delete(mdoc, mdoc->last);
        } else {
                free(ch->string);
                ch->string = mandoc_strdup(p);
        }

        return(1);
}

static int
post_rs(POST_ARGS)
{
        struct mdoc_node *nn, *next, *prev;
        int               i, j;

        switch (mdoc->last->type) {
        case (MDOC_HEAD):
                check_count(mdoc, MDOC_HEAD, CHECK_WARN, CHECK_EQ, 0);
                return(1);
        case (MDOC_BODY):
                if (mdoc->last->child)
                        break;
                check_count(mdoc, MDOC_BODY, CHECK_WARN, CHECK_GT, 0);
                return(1);
        default:
                return(1);
        }

        /*
         * Make sure only certain types of nodes are allowed within the
         * the ‘Rs’ body.  Delete offending nodes and raise a warning.
         * Do this before re-ordering for the sake of clarity.
         */

        next = NULL;
        for (nn = mdoc->last->child; nn; nn = next) {
                for (i = 0; i < RSORD_MAX; i++)
                        if (nn->tok == rsord[i])
                                break;

                if (i < RSORD_MAX) {
                        if (MDOC__J == rsord[i] || MDOC__B == rsord[i])
                                mdoc->last->norm->Rs.quote_T++;
                        next = nn->next;
                        continue;
                }

                next = nn->next;
                mdoc_nmsg(mdoc, nn, MANDOCERR_CHILD);
                mdoc_node_delete(mdoc, nn);
        }

        /*
         * Nothing to sort if only invalid nodes were found
         * inside the ‘Rs’ body.
         */

        if (NULL == mdoc->last->child)
                return(1);

        /*
         * The full ‘Rs’ block needs special handling to order the
         * sub-elements according to ‘rsord’.  Pick through each element
```

```
1712          * and correctly order it.  This is a insertion sort.
1713          */
1715         next = NULL;
1716         for (nn = mdoc->last->child->next; nn; nn = next) {
1717                 /* Determine order of ‘nn’. */
1718                 for (i = 0; i < RSORD_MAX; i++)
1719                         if (rsord[i] == nn->tok)
1720                                 break;
1722                 /*
1723                  * Remove ‘nn’ from the chain.  This somewhat
1724                  * repeats mdoc_node_unlink(), but since we’re
1725                  * just re-ordering, there’s no need for the
1726                  * full unlink process.
1727                  */
1728
1729                 if (NULL != (next = nn->next))
1730                         next->prev = nn->prev;
1732                 if (NULL != (prev = nn->prev))
1733                         prev->next = nn->next;
1735                 nn->prev = nn->next = NULL;
1737                 /*
1738                  * Scan back until we reach a node that’s
1739                  * ordered before ‘nn’.
1740                  */
1742                 for ( ; prev ; prev = prev->prev) {
1743                         /* Determine order of ‘prev’. */
1744                         for (j = 0; j < RSORD_MAX; j++)
1745                                 if (rsord[j] == prev->tok)
1746                                         break;
1748                         if (j <= i)
1749                                 break;
1750                 }
1752                 /*
1753                  * Set ‘nn’ back into its correct place in front
1754                  * of the ‘prev’ node.
1755                  */
1757                 nn->prev = prev;
1759                 if (prev) {
1760                         if (prev->next)
1761                                 prev->next->prev = nn;
1762                         nn->next = prev->next;
1763                         prev->next = nn;
1764                 } else {
1765                         mdoc->last->child->prev = nn;
1766                         nn->next = mdoc->last->child;
1767                         mdoc->last->child = nn;
1768                 }
1769         }
1771         return(1);
1772 }
1774 static int
1775 post_ns(POST_ARGS)
1776 {
```



```
1778         if (MDOC_LINE & mdoc->last->flags)
1779                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_IGNNS);
1780         return(1);
1781 }
1783 static int
1784 post_sh(POST_ARGS)
1785 {
1787         if (MDOC_HEAD == mdoc->last->type)
1788                 return(post_sh_head(mdoc));
1789         if (MDOC_BODY == mdoc->last->type)
1790                 return(post_sh_body(mdoc));
1792         return(1);
1793 }
1795 static int
1796 post_sh_body(POST_ARGS)
1797 {
1798         struct mdoc_node *n;
1800         if (SEC_NAME != mdoc->lastsec)
1801                 return(1);
1803         /*
1804          * Warn if the NAME section doesn’t contain the ‘Nm’ and ‘Nd’
1805          * macros (can have multiple ‘Nm’ and one ‘Nd’).  Note that the
1806          * children of the BODY declaration can also be "text".
1807          */
1809         if (NULL == (n = mdoc->last->child)) {
1810                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADNAMESEC);
1811                 return(1);
1812         }
1814         for ( ; n && n->next; n = n->next) {
1815                 if (MDOC_ELEM == n->type && MDOC_Nm == n->tok)
1816                         continue;
1817                 if (MDOC_TEXT == n->type)
1818                         continue;
1819                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADNAMESEC);
1820         }
1822         assert(n);
1823         if (MDOC_BLOCK == n->type && MDOC_Nd == n->tok)
1824                 return(1);
1826         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_BADNAMESEC);
1827         return(1);
1828 }
1830 static int
1831 post_sh_head(POST_ARGS)
1832 {
1833         char             buf[BUFSIZ];
1834         struct mdoc_node *n;
1835         enum mdoc_sec    sec;
1836         int              c;
1838         /*
1839          * Process a new section.  Sections are either "named" or
1840          * "custom".  Custom sections are user-defined, while named ones
1841          * follow a conventional order and may only appear in certain
1842          * manual sections.
1843          */
```

```
1845         sec = SEC_CUSTOM;
1846         buf[0] = '\0';
1847         if (-1 == (c = concat(buf, mdoc->last->child, BUFSIZ))) {
1848                 mdoc_nmsg(mdoc, mdoc->last->child, MANDOCERR_MEM);
1849                 return(0);
1850         } else if (1 == c)
1851                 sec = a2sec(buf);

1853         /* The NAME should be first. */

1855         if (SEC_NAME != sec && SEC_NONE == mdoc->lastnamed)
1856                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_NAMESECFIRST);

1858         /* The SYNOPSIS gets special attention in other areas. */

1860         if (SEC_SYNOPSIS == sec)
1861                 mdoc->flags |= MDOC_SYNOPSIS;
1862         else
1863                 mdoc->flags &= ~MDOC_SYNOPSIS;

1865         /* Mark our last section. */

1867         mdoc->lastsec = sec;

1869         /*
1870          * Set the section attribute for the current HEAD, for its
1871          * parent BLOCK, and for the HEAD children; the latter can
1872          * only be TEXT nodes, so no recursion is needed.
1873          * For other blocks and elements, including .Sh BODY, this is
1874          * done when allocating the node data structures, but for .Sh
1875          * BLOCK and HEAD, the section is still unknown at that time.
1876          */

1878         mdoc->last->parent->sec = sec;
1879         mdoc->last->sec = sec;
1880         for (n = mdoc->last->child; n; n = n->next)
1881                 n->sec = sec;

1883         /* We don't care about custom sections after this. */

1885         if (SEC_CUSTOM == sec)
1886                 return(1);

1888         /*
1889          * Check whether our non-custom section is being repeated or is
1890          * out of order.
1891          */

1893         if (sec == mdoc->lastnamed)
1894                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECREP);

1896         if (sec < mdoc->lastnamed)
1897                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECOOO);

1899         /* Mark the last named section. */

1901         mdoc->lastnamed = sec;

1903         /* Check particular section/manual conventions. */

1905         assert(mdoc->meta.msec);

1907         switch (sec) {
1908         case (SEC_RETURN_VALUES):
1909                 /* FALLTHROUGH */
```



```
1910         case (SEC_ERRORS):
1911                 /* FALLTHROUGH */
1912         case (SEC_LIBRARY):
1913                 if (*mdoc->meta.msec == '2')
1914                         break;
1915                 if (*mdoc->meta.msec == '3')
1916                         break;
1917                 if (*mdoc->meta.msec == '9')
1918                         break;
1919                 mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_SECMSEC);
1920                 break;
1921         default:
1922                 break;
1923         }

1925         return(1);
1926 }

1928 static int
1929 post_ignpar(POST_ARGS)
1930 {
1931         struct mdoc_node *np;

1933         if (MDOC_BODY != mdoc->last->type)
1934                 return(1);

1936         if (NULL != (np = mdoc->last->child))
1937                 if (MDOC_Pp == np->tok || MDOC_Lp == np->tok) {
1938                         mdoc_nmsg(mdoc, np, MANDOCERR_IGNPAR);
1939                         mdoc_node_delete(mdoc, np);
1940                 }

1942         if (NULL != (np = mdoc->last->last))
1943                 if (MDOC_Pp == np->tok || MDOC_Lp == np->tok) {
1944                         mdoc_nmsg(mdoc, np, MANDOCERR_IGNPAR);
1945                         mdoc_node_delete(mdoc, np);
1946                 }

1948         return(1);
1949 }

1951 static int
1952 pre_par(PRE_ARGS)
1953 {

1955         if (NULL == mdoc->last)
1956                 return(1);
1957         if (MDOC_ELEM != n->type && MDOC_BLOCK != n->type)
1958                 return(1);

1960         /*
1961          * Don't allow prior 'Lp' or 'Pp' prior to a paragraph-type
1962          * block:  'Lp', 'Pp', or non-compact 'Bd' or 'Bl'.
1963          */

1965         if (MDOC_Pp != mdoc->last->tok && MDOC_Lp != mdoc->last->tok)
1966                 return(1);
1967         if (MDOC_Bl == n->tok && n->norm->Bl.comp)
1968                 return(1);
1969         if (MDOC_Bd == n->tok && n->norm->Bd.comp)
1970                 return(1);
1971         if (MDOC_It == n->tok && n->parent->norm->Bl.comp)
1972                 return(1);

1974         mdoc_nmsg(mdoc, mdoc->last, MANDOCERR_IGNPAR);
1975         mdoc_node_delete(mdoc, mdoc->last);
```

```
1976         return(1);
1977 }

1979 static int
1980 pre_literal(PRE_ARGS)
1981 {

1983         if (MDOC_BODY != n->type)
1984                 return(1);

1986         /*
1987          * The ‘Dl’ (note "el" not "one") and ‘Bd -literal’ and ‘Bd
1988          * -unfilled’ macros set MDOC_LITERAL on entrance to the body.
1989          */

1991         switch (n->tok) {
1992         case (MDOC_Dl):
1993                 mdoc->flags |= MDOC_LITERAL;
1994                 break;
1995         case (MDOC_Bd):
1996                 if (DISP_literal == n->norm->Bd.type)
1997                         mdoc->flags |= MDOC_LITERAL;
1998                 if (DISP_unfilled == n->norm->Bd.type)
1999                         mdoc->flags |= MDOC_LITERAL;
2000                 break;
2001         default:
2002                 abort();
2003                 /* NOTREACHED */
2004         }
2005
2006         return(1);
2007 }

2009 static int
2010 post_dd(POST_ARGS)
2011 {
2012         char              buf[DATESIZE];
2013         struct mdoc_node *n;
2014         int               c;

2016         if (mdoc->meta.date)
2017                 free(mdoc->meta.date);

2019         n = mdoc->last;
2020         if (NULL == n->child || '\0' == n->child->string[0]) {
2021                 mdoc->meta.date = mandoc_normdate
2022                         (mdoc->parse, NULL, n->line, n->pos);
2023                 return(1);
2024         }

2026         buf[0] = '\0';
2027         if (-1 == (c = concat(buf, n->child, DATESIZE))) {
2028                 mdoc_nmsg(mdoc, n->child, MANDOCERR_MEM);
2029                 return(0);
2030         }

2032         assert(c);
2033         mdoc->meta.date = mandoc_normdate
2034                 (mdoc->parse, buf, n->line, n->pos);

2036         return(1);
2037 }

2039 static int
2040 post_dt(POST_ARGS)
2041 {
```



```
2042         struct mdoc_node *nn, *n;
2043         const char       *cp;
2044         char             *p;

2046         n = mdoc->last;

2048         if (mdoc->meta.title)
2049                 free(mdoc->meta.title);
2050         if (mdoc->meta.vol)
2051                 free(mdoc->meta.vol);
2052         if (mdoc->meta.arch)
2053                 free(mdoc->meta.arch);

2055         mdoc->meta.title = mdoc->meta.vol = mdoc->meta.arch = NULL;

2057         /* First make all characters uppercase. */

2059         if (NULL != (nn = n->child))
2060                 for (p = nn->string; *p; p++) {
2061                         if (toupper((unsigned char)*p) == *p)
2062                                 continue;

2064                         /*
2065                          * FIXME: don't be lazy: have this make all
2066                          * characters be uppercase and just warn once.
2067                          */
2068                         mdoc_nmsg(mdoc, nn, MANDOCERR_UPPERCASE);
2069                         break;
2070                 }

2072         /* Handles: '.Dt'
2073          *   --> title = unknown, volume = local, msec = 0, arch = NULL
2074          */

2076         if (NULL == (nn = n->child)) {
2077                 /* XXX: make these macro values. */
2078                 /* FIXME: warn about missing values. */
2079                 mdoc->meta.title = mandoc_strdup("UNKNOWN");
2080                 mdoc->meta.vol = mandoc_strdup("LOCAL");
2081                 mdoc->meta.msec = mandoc_strdup("1");
2082                 return(1);
2083         }

2085         /* Handles: '.Dt TITLE'
2086          *   --> title = TITLE, volume = local, msec = 0, arch = NULL
2087          */

2089         mdoc->meta.title = mandoc_strdup
2090                 ('\0' == nn->string[0] ? "UNKNOWN" : nn->string);

2092         if (NULL == (nn = nn->next)) {
2093                 /* FIXME: warn about missing msec. */
2094                 /* XXX: make this a macro value. */
2095                 mdoc->meta.vol = mandoc_strdup("LOCAL");
2096                 mdoc->meta.msec = mandoc_strdup("1");
2097                 return(1);
2098         }

2100         /* Handles: '.Dt TITLE SEC'
2101          *   --> title = TITLE, volume = SEC is msec ?
2102          *           format(msec) : SEC,
2103          *       msec = SEC is msec ? atoi(msec) : 0,
2104          *       arch = NULL
2105          */

2107         cp = mandoc_a2msec(nn->string);
```

```
2108         if (cp) {
2109                 mdoc->meta.vol = mandoc_strdup(cp);
2110                 mdoc->meta.msec = mandoc_strdup(nn->string);
2111         } else {
2112                 mdoc_nmsg(mdoc, n, MANDOCERR_BADMSEC);
2113                 mdoc->meta.vol = mandoc_strdup(nn->string);
2114                 mdoc->meta.msec = mandoc_strdup(nn->string);
2115         }

2117         if (NULL == (nn = nn->next))
2118                 return(1);

2120         /* Handles: '.Dt TITLE SEC VOL'
2121          *   --> title = TITLE, volume = VOL is vol ?
2122          *       format(VOL) :
2123          *           VOL is arch ? format(arch) :
2124          *               VOL
2125          */

2127         cp = mdoc_a2vol(nn->string);
2128         if (cp) {
2129                 free(mdoc->meta.vol);
2130                 mdoc->meta.vol = mandoc_strdup(cp);
2131         } else {
2132                 /* FIXME: warn about bad arch. */
2133                 cp = mdoc_a2arch(nn->string);
2134                 if (NULL == cp) {
2135                         free(mdoc->meta.vol);
2136                         mdoc->meta.vol = mandoc_strdup(nn->string);
2137                 } else
2138                         mdoc->meta.arch = mandoc_strdup(cp);
2139         }

2141         /* Ignore any subsequent parameters... */
2142         /* FIXME: warn about subsequent parameters. */

2144         return(1);
2145 }

2147 static int
2148 post_prol(POST_ARGS)
2149 {
2150         /*
2151          * Remove prologue macros from the document after they're
2152          * processed.  The final document uses mdoc_meta for these
2153          * values and discards the originals.
2154          */

2156         mdoc_node_delete(mdoc, mdoc->last);
2157         if (mdoc->meta.title && mdoc->meta.date && mdoc->meta.os)
2158                 mdoc->flags |= MDOC_PBODY;

2160         return(1);
2161 }

2163 static int
2164 post_bx(POST_ARGS)
2165 {
2166         struct mdoc_node        *n;

2168         /*
2169          * Make 'Bx's second argument always start with an uppercase
2170          * letter.  Groff checks if it's an "accepted" term, but we just
2171          * uppercase blindly.
2172          */
```



```
2174         n = mdoc->last->child;
2175         if (n && NULL != (n = n->next))
2176                 *n->string = (char)toupper
2177                         ((unsigned char)*n->string);

2179         return(1);
2180 }

2182 static int
2183 post_os(POST_ARGS)
2184 {
2185         struct mdoc_node *n;
2186         char              buf[BUFSIZ];
2187         int               c;
2188 #ifndef OSNAME
2189         struct utsname    utsname;
2190 #endif

2192         n = mdoc->last;

2194         /*
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().
2199          */

2201         if (mdoc->meta.os)
2202                 free(mdoc->meta.os);

2204         buf[0] = '\0';
2205         if (-1 == (c = concat(buf, n->child, BUFSIZ))) {
2206                 mdoc_nmsg(mdoc, n->child, MANDOCERR_MEM);
2207                 return(0);
2208         }

2210         assert(c);

2212         /* XXX: yes, these can all be dynamically-adjusted buffers, but
2213          * it's really not worth the extra hackery.
2214          */

2216         if ('\0' == buf[0]) {
2217 #ifdef OSNAME
2218                 if (strlcat(buf, OSNAME, BUFSIZ) >= BUFSIZ) {
2219                         mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2220                         return(0);
2221                 }
2222 #else /*!OSNAME */
2223                 if (-1 == uname(&utsname)) {
2224                         mdoc_nmsg(mdoc, n, MANDOCERR_UNAME);
2225                         mdoc->meta.os = mandoc_strdup("UNKNOWN");
2226                         return(post_prol(mdoc));
2227                 }

2229                 if (strlcat(buf, utsname.sysname, BUFSIZ) >= BUFSIZ) {
2230                         mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2231                         return(0);
2232                 }
2233                 if (strlcat(buf, " ", BUFSIZ) >= BUFSIZ) {
2234                         mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2235                         return(0);
2236                 }
2237                 if (strlcat(buf, utsname.release, BUFSIZ) >= BUFSIZ) {
2238                         mdoc_nmsg(mdoc, n, MANDOCERR_MEM);
2239                         return(0);
```

```
2240                }
2241 #endif /*!OSNAME*/
2242         }

2244         mdoc->meta.os = mandoc_strdup(buf);
2245         return(1);
2246 }

2248 static int
2249 post_std(POST_ARGS)
2250 {
2251         struct mdoc_node *nn, *n;

2253         n = mdoc->last;

2255         /*
2256          * Macros accepting ‘-std’ as an argument have the name of the
2257          * current document (‘Nm’) filled in as the argument if it’s not
2258          * provided.
2259          */

2261         if (n->child)
2262                 return(1);

2264         if (NULL == mdoc->meta.name)
2265                 return(1);
2266
2267         nn = n;
2268         mdoc->next = MDOC_NEXT_CHILD;

2270         if ( ! mdoc_word_alloc(mdoc, n->line, n->pos, mdoc->meta.name))
2271                 return(0);

2273         mdoc->last = nn;
2274         return(1);
2275 }

2277 /*
2278  * Concatenate a node, stopping at the first non-text.
2279  * Concatenation is separated by a single whitespace.
2280  * Returns -1 on fatal (string overrun) error, 0 if child nodes were
2281  * encountered, 1 otherwise.
2282  */
2283 static int
2284 concat(char *p, const struct mdoc_node *n, size_t sz)
2285 {

2287         for ( ; NULL != n; n = n->next) {
2288                 if (MDOC_TEXT != n->type)
2289                         return(0);
2290                 if ('\0' != p[0] && strlcat(p, " ", sz) >= sz)
2291                         return(-1);
2292                 if (strlcat(p, n->string, sz) >= sz)
2293                         return(-1);
2294                 concat(p, n->child, sz);
2295         }

2297         return(1);
2298 }

2300 static enum mdoc_sec
2301 a2sec(const char *p)
2302 {
2303         int             i;

2305         for (i = 0; i < (int)SEC__MAX; i++)
```



```
2306                 if (secnames[i] && 0 == strcmp(p, secnames[i]))
2307                         return((enum mdoc_sec)i);

2309         return(SEC_CUSTOM);
2310 }

2312 static size_t
2313 macro2len(enum mdoct macro)
2314 {

2316         switch (macro) {
2317         case(MDOC_Ad):
2318                 return(12);
2319         case(MDOC_Ao):
2320                 return(12);
2321         case(MDOC_An):
2322                 return(12);
2323         case(MDOC_Aq):
2324                 return(12);
2325         case(MDOC_Ar):
2326                 return(12);
2327         case(MDOC_Bo):
2328                 return(12);
2329         case(MDOC_Bq):
2330                 return(12);
2331         case(MDOC_Cd):
2332                 return(12);
2333         case(MDOC_Cm):
2334                 return(10);
2335         case(MDOC_Do):
2336                 return(10);
2337         case(MDOC_Dq):
2338                 return(12);
2339         case(MDOC_Dv):
2340                 return(12);
2341         case(MDOC_Eo):
2342                 return(12);
2343         case(MDOC_Em):
2344                 return(10);
2345         case(MDOC_Er):
2346                 return(17);
2347         case(MDOC_Ev):
2348                 return(15);
2349         case(MDOC_Fa):
2350                 return(12);
2351         case(MDOC_Fl):
2352                 return(10);
2353         case(MDOC_Fo):
2354                 return(16);
2355         case(MDOC_Fn):
2356                 return(16);
2357         case(MDOC_Ic):
2358                 return(10);
2359         case(MDOC_Li):
2360                 return(16);
2361         case(MDOC_Ms):
2362                 return(6);
2363         case(MDOC_Nm):
2364                 return(10);
2365         case(MDOC_No):
2366                 return(12);
2367         case(MDOC_Oo):
2368                 return(10);
2369         case(MDOC_Op):
2370                 return(14);
2371         case(MDOC_Pa):
```

```
2372                    return(32);
2373            case(MDOC_Pf):
2374                    return(12);
2375            case(MDOC_Po):
2376                    return(12);
2377            case(MDOC_Pq):
2378                    return(12);
2379            case(MDOC_Ql):
2380                    return(16);
2381            case(MDOC_Qo):
2382                    return(12);
2383            case(MDOC_So):
2384                    return(12);
2385            case(MDOC_Sq):
2386                    return(12);
2387            case(MDOC_Sy):
2388                    return(6);
2389            case(MDOC_Sx):
2390                    return(16);
2391            case(MDOC_Tn):
2392                    return(10);
2393            case(MDOC_Va):
2394                    return(12);
2395            case(MDOC_Vt):
2396                    return(12);
2397            case(MDOC_Xr):
2398                    return(10);
2399            default:
2400                    break;
2401            };
2402            return(0);
2403 }
```

```
**********************************************************
    1127 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/msec.c
Initial import of man functionality.
**********************************************************
```

```
  1 /*      $Id: msec.c,v 1.10 2011/12/02 01:37:14 schwarze Exp $ */
  2 /*
  3  * Copyright (c) 2009 Kristaps Dzonsons <kristaps@bsd.lv>
  4  *
  5  * Permission to use, copy, modify, and distribute this software for any
  6  * purpose with or without fee is hereby granted, provided that the above
  7  * copyright notice and this permission notice appear in all copies.
  8  *
  9  * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 10  * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 11  * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 12  * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 13  * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 14  * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 15  * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 16  */
 17 #ifdef HAVE_CONFIG_H
 18 #include "config.h"
 19 #endif

 21 #include <stdlib.h>
 22 #include <string.h>

 24 #include "mandoc.h"
 25 #include "libmandoc.h"

 27 #define LINE(x, y) \
 28         if (0 == strcmp(p, x)) return(y);

 30 const char *
 31 mandoc_a2msec(const char *p)
 32 {

 34 #include "msec.in"

 36         return(NULL);
 37 }
```

```
**********************************************************
   11663 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/msec.in
Initial import of man functionality.
**********************************************************
```

```
/*
 * 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",              "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("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",              "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("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("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")
LINE("3PICLTREE",       "PICL Plug-In Library Functions")
LINE("3picltree",       "PICL Plug-In Library Functions")
LINE("3PLOT",           "Graphics Interface Library Functions")
LINE("3plot",           "Graphics Interface Library Functions")
LINE("3POOL",           "Pool Configuration Manipulation Library Functions")
LINE("3pool",           "Pool Configuration Manipulation Library Functions")
LINE("3PROC",           "Process Control Library Functions")
```

```
128 LINE("3proc",           "Process Control Library Functions")
129 LINE("3PROJECT",        "Project Database Access Library Functions")
130 LINE("3project",        "Project Database Access Library Functions")
131 LINE("3RAC",           "Remote Asynchronous Calls Library Functions")
132 LINE("3rac",           "Remote Asynchronous Calls Library Functions")
133 LINE("3RESOLV",        "Resolver Library Functions")
134 LINE("3resolv",        "Resolver Library Functions")
135 LINE("3RPC",           "RPC Library Functions")
136 LINE("3rpc",           "RPC Library Functions")
137 LINE("3RSM",           "Remote Shared Memory Library Functions")
138 LINE("3rsm",           "Remote Shared Memory Library Functions")
139 LINE("3RT",            "Realtime Library Functions")
140 LINE("3rt",            "Realtime Library Functions")
141 LINE("3SASL",          "Simple Authentication Security Layer Library Functions"
142 LINE("3sasl",          "Simple Authentication Security Layer Library Functions"
143 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")
147 LINE("3SEC",           "File Access Control Library Functions")
148 LINE("3sec",           "File Access Control Library Functions")
149 LINE("3SECDB",         "Security Attributes Database Library Functions")
150 LINE("3secdb",         "Security Attributes Database Library Functions")
151 LINE("3SIP",           "Session Initiation Protocol Library Functions")
152 LINE("3sip",           "Session Initiation Protocol Library Functions")
153 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")
157 LINE("3SOCKET",        "Sockets Library Functions")
158 LINE("3socket",        "Sockets Library Functions")
159 LINE("3STMF",          "SCSI Target Mode Framework Library Functions")
160 LINE("3stmf",          "SCSI Target Mode Framework Library Functions")
161 LINE("3SYSEVENT",      "System Event Library Functions")
162 LINE("3sysevent",      "System Event Library Functions")
163 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")
167 LINE("3TNF",           "TNF Library Functions")
168 LINE("3tnf",           "TNF Library Functions")
169 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")
173 LINE("3UUID",          "Universally Unique Identifier Library Functions")
174 LINE("3uuid",          "Universally Unique Identifier Library Functions")
175 LINE("3VOLMGT",        "Volume Management Library Functions")
176 LINE("3volmgt",        "Volume Management Library Functions")
177 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")
181 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")
187 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")
205 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")
211 LINE("4",              "File Formats")
212 LINE("4B",             "illumos/BSD Compatibility Package File Formats")
213 LINE("4b",             "illumos/BSD Compatibility Package File Formats")
214 LINE("5",              "Standards, Environments, and Macros")
215 LINE("6",              "Games and Demos")
216 LINE("7",              "Device and Network Interfaces")
217 LINE("7B",             "illumos/BSD Compatibility Special Files")
218 LINE("7b",             "illumos/BSD Compatibility Special Files")
219 LINE("7D",             "Devices")
220 LINE("7d",             "Devices")
221 LINE("7FS",            "File Systems")
222 LINE("7fs",            "File Systems")
223 LINE("7I",             "Ioctl Requests")
224 LINE("7i",             "Ioctl Requests")
225 LINE("7IPP",           "IP Quality of Service Modules")
226 LINE("7ipp",           "IP Quality of Service Modules")
227 LINE("7M",             "STREAMS Modules")
228 LINE("7m",             "STREAMS Modules")
229 LINE("7P",             "Protocols")
230 LINE("7p",             "Protocols")
231 LINE("8",              "Maintenance Procedures")
232 LINE("8C",             "Maintenance Procedures")
233 LINE("8c",             "Maintenance Procedures")
234 LINE("8S",             "Maintenance Procedures")
235 LINE("8s",             "Maintenance Procedures")
236 LINE("9",              "Device Driver Interfaces")
237 LINE("9E",             "Driver Entry Points")
238 LINE("9e",             "Driver Entry Points")
239 LINE("9F",             "Kernel Functions for Drivers")
240 LINE("9f",             "Kernel Functions for Drivers")
241 LINE("9P",             "Kernel Properties for Drivers")
242 LINE("9p",             "Kernel Properties for Drivers")
243 LINE("9S",             "Data Structures for Drivers")
244 LINE("9s",             "Data Structures for Drivers")
```

```
**********************************************************
    6614 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/out.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: out.c,v 1.43 2011/09/20 23:05:49 schwarze 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 <sys/types.h>

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

#include "mandoc.h"
#include "out.h"

static  void    tblcalc_data(struct rofftbl *, struct roffcol *,
                        const struct tbl *, const struct tbl_dat *);
static  void    tblcalc_literal(struct rofftbl *, struct roffcol *,
                        const struct tbl_dat *);
static  void    tblcalc_number(struct rofftbl *, struct roffcol *,
                        const struct tbl *, const struct tbl_dat *);

/*
 * Convert a 'scaling unit' to a consistent form, or fail.  Scaling
 * units are documented in groff.7, mdoc.7, man.7.
 */
int
a2roffsu(const char *src, struct roffsu *dst, enum roffscale def)
{
        char             buf[BUFSIZ], hasd;
        int              i;
        enum roffscale   unit;

        if ('\0' == *src)
                return(0);

        i = hasd = 0;

        switch (*src) {
        case ('+'):
                src++;
                break;
        case ('-'):
```



```
                buf[i++] = *src++;
                break;
        default:
                break;
        }

        if ('\0' == *src)
                return(0);

        while (i < BUFSIZ) {
                if ( ! isdigit((unsigned char)*src)) {
                        if ('.' != *src)
                                break;
                        else if (hasd)
                                break;
                        else
                                hasd = 1;
                }
                buf[i++] = *src++;
        }

        if (BUFSIZ == i || (*src && *(src + 1)))
                return(0);

        buf[i] = '\0';

        switch (*src) {
        case ('c'):
                unit = SCALE_CM;
                break;
        case ('i'):
                unit = SCALE_IN;
                break;
        case ('P'):
                unit = SCALE_PC;
                break;
        case ('p'):
                unit = SCALE_PT;
                break;
        case ('f'):
                unit = SCALE_FS;
                break;
        case ('v'):
                unit = SCALE_VS;
                break;
        case ('m'):
                unit = SCALE_EM;
                break;
        case ('\0'):
                if (SCALE_MAX == def)
                        return(0);
                unit = SCALE_BU;
                break;
        case ('u'):
                unit = SCALE_BU;
                break;
        case ('M'):
                unit = SCALE_MM;
                break;
        case ('n'):
                unit = SCALE_EN;
                break;
        default:
                return(0);
        }
```

```
128         /* FIXME: do this in the caller. */
129         if ((dst->scale = atof(buf)) < 0)
130                 dst->scale = 0;
131         dst->unit = unit;
132         return(1);
133 }

135 /*
136  * Calculate the abstract widths and decimal positions of columns in a
137  * table.  This routine allocates the columns structures then runs over
138  * all rows and cells in the table.  The function pointers in "tbl" are
139  * used for the actual width calculations.
140  */
141 void
142 tblcalc(struct rofftbl *tbl, const struct tbl_span *sp)
143 {
144         const struct tbl_dat    *dp;
145         const struct tbl_head   *hp;
146         struct roffcol          *col;
147         int                      spans;

149         /*
150          * Allocate the master column specifiers.  These will hold the
151          * widths and decimal positions for all cells in the column.  It
152          * must be freed and nullified by the caller.
153          */

155         assert(NULL == tbl->cols);
156         tbl->cols = mandoc_calloc
157                 ((size_t)sp->tbl->cols, sizeof(struct roffcol));

159         hp = sp->head;

161         for ( ; sp; sp = sp->next) {
162                 if (TBL_SPAN_DATA != sp->pos)
163                         continue;
164                 spans = 1;
165                 /*
166                  * Account for the data cells in the layout, matching it
167                  * to data cells in the data section.
168                  */
169                 for (dp = sp->first; dp; dp = dp->next) {
170                         /* Do not used spanned cells in the calculation. */
171                         if (0 < --spans)
172                                 continue;
173                         spans = dp->spans;
174                         if (1 < spans)
175                                 continue;
176                         assert(dp->layout);
177                         col = &tbl->cols[dp->layout->head->ident];
178                         tblcalc_data(tbl, col, sp->tbl, dp);
179                 }
180         }

182         /*
183          * Calculate width of the spanners.  These get one space for a
184          * vertical line, two for a double-vertical line.
185          */

187         for ( ; hp; hp = hp->next) {
188                 col = &tbl->cols[hp->ident];
189                 switch (hp->pos) {
190                 case (TBL_HEAD_VERT):
191                         col->width = (*tbl->len)(1, tbl->arg);
192                         break;
193                 case (TBL_HEAD_DVERT):
```



```
194                         col->width = (*tbl->len)(2, tbl->arg);
195                         break;
196                 default:
197                         break;
198                 }
199         }
200 }

202 static void
203 tblcalc_data(struct rofftbl *tbl, struct roffcol *col,
204                 const struct tbl *tp, const struct tbl_dat *dp)
205 {
206         size_t           sz;

208         /* Branch down into data sub-types. */

210         switch (dp->layout->pos) {
211         case (TBL_CELL_HORIZ):
212                 /* FALLTHROUGH */
213         case (TBL_CELL_DHORIZ):
214                 sz = (*tbl->len)(1, tbl->arg);
215                 if (col->width < sz)
216                         col->width = sz;
217                 break;
218         case (TBL_CELL_LONG):
219                 /* FALLTHROUGH */
220         case (TBL_CELL_CENTRE):
221                 /* FALLTHROUGH */
222         case (TBL_CELL_LEFT):
223                 /* FALLTHROUGH */
224         case (TBL_CELL_RIGHT):
225                 tblcalc_literal(tbl, col, dp);
226                 break;
227         case (TBL_CELL_NUMBER):
228                 tblcalc_number(tbl, col, tp, dp);
229                 break;
230         case (TBL_CELL_DOWN):
231                 break;
232         default:
233                 abort();
234                 /* NOTREACHED */
235         }
236 }

238 static void
239 tblcalc_literal(struct rofftbl *tbl, struct roffcol *col,
240                 const struct tbl_dat *dp)
241 {
242         size_t           sz;
243         const char      *str;

245         str = dp->string ? dp->string : "";
246         sz = (*tbl->slen)(str, tbl->arg);

248         if (col->width < sz)
249                 col->width = sz;
250 }

252 static void
253 tblcalc_number(struct rofftbl *tbl, struct roffcol *col,
254                 const struct tbl *tp, const struct tbl_dat *dp)
255 {
256         int              i;
257         size_t           sz, psz, ssz, d;
258         const char      *str;
259         char            *cp;
```

```
260         char             buf[2];

262         /*
263          * First calculate number width and decimal place (last + 1 for
264          * non-decimal numbers).  If the stored decimal is subsequent to
265          * ours, make our size longer by that difference
266          * (right-"shifting"); similarly, if ours is subsequent the
267          * stored, then extend the stored size by the difference.
268          * Finally, re-assign the stored values.
269          */

271         str = dp->string ? dp->string : "";
272         sz = (*tbl->slen)(str, tbl->arg);

274         /* FIXME: TBL_DATA_HORIZ et al.? */

276         buf[0] = tp->decimal;
277         buf[1] = '\0';

279         psz = (*tbl->slen)(buf, tbl->arg);

281         if (NULL != (cp = strrchr(str, tp->decimal))) {
282                 buf[1] = '\0';
283                 for (ssz = 0, i = 0; cp != &str[i]; i++) {
284                         buf[0] = str[i];
285                         ssz += (*tbl->slen)(buf, tbl->arg);
286                 }
287                 d = ssz + psz;
288         } else
289                 d = sz + psz;

291         /* Adjust the settings for this column. */

293         if (col->decimal > d) {
294                 sz += col->decimal - d;
295                 d = col->decimal;
296         } else
297                 col->width += d - col->decimal;

299         if (sz > col->width)
300                 col->width = sz;
301         if (d > col->decimal)
302                 col->decimal = d;
303 }
```

```
**********************************************************
    2124 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/out.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: out.h,v 1.21 2011/07/17 15:24:25 kristaps Exp $ */
/*
 * 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 OUT_H
#define OUT_H

enum    roffscale {
        SCALE_CM, /* centimeters (c) */
        SCALE_IN, /* inches (i) */
        SCALE_PC, /* pica (P) */
        SCALE_PT, /* points (p) */
        SCALE_EM, /* ems (m) */
        SCALE_MM, /* mini-ems (M) */
        SCALE_EN, /* ens (n) */
        SCALE_BU, /* default horizontal (u) */
        SCALE_VS, /* default vertical (v) */
        SCALE_FS, /* syn. for u (f) */
        SCALE_MAX
};

struct  roffcol {
        size_t           width; /* width of cell */
        size_t           decimal; /* decimal position in cell */
};

struct  roffsu {
        enum roffscale    unit;
        double            scale;
};

typedef size_t  (*tbl_strlen)(const char *, void *);
typedef size_t  (*tbl_len)(size_t, void *);

struct  rofftbl {
        tbl_strlen       slen; /* calculate string length */
        tbl_len          len; /* produce width of empty space */
        struct roffcol  *cols; /* master column specifiers */
        void            *arg; /* passed to slen and len */
};

__BEGIN_DECLS

#define SCALE_VS_INIT(p, v) \
        do { (p)->unit = SCALE_VS; \
             (p)->scale = (v); } \
        while (/* CONSTCOND */ 0)

#define SCALE_HS_INIT(p, v) \
```



```
        do { (p)->unit = SCALE_BU; \
             (p)->scale = (v); } \
        while (/* CONSTCOND */ 0)

int              a2roffsu(const char *, struct roffsu *, enum roffscale);
void             tblcalc(struct rofftbl *tbl, const struct tbl_span *);

__END_DECLS

#endif /*!OUT_H*/
```

```
**********************************************************
   10389 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/preconv.c
Finish integration.  Use mandoc_preconv, etc.
**********************************************************
```

```
  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

 42 enum    enc {
 43         ENC_UTF_8, /* UTF-8 */
 44         ENC_US_ASCII, /* US-ASCII */
 45         ENC_LATIN_1, /* Latin-1 */
 46         ENC__MAX
 47 };

 49 struct  buf {
 50         char            *buf; /* binary input buffer */
 51         size_t           sz; /* size of binary buffer */
 52         size_t           offs; /* starting buffer offset */
 53 };

 55 struct  encode {
 56         const char      *name;
 57         int             (*conv)(const struct buf *);
 58 };

 60 static  int      cue_enc(const struct buf *, size_t *, enum enc *);
 61 static  int      conv_latin_1(const struct buf *);
```



```
 62 static  int      conv_us_ascii(const struct buf *);
 63 static  int      conv_utf_8(const struct buf *);
 64 static  int      read_whole_file(const char *, int,
 65                         struct buf *, int *);
 66 static  void     resize_buf(struct buf *, size_t);
 67 static  void     usage(void);

 69 static  const struct encode encs[ENC__MAX] = {
 70         { "utf-8", conv_utf_8 }, /* ENC_UTF_8 */
 71         { "us-ascii", conv_us_ascii }, /* ENC_US_ASCII */
 72         { "latin-1", conv_latin_1 }, /* ENC_LATIN_1 */
 73 };

 75 static  const char       *progname;

 77 static void
 78 usage(void)
 79 {

 81         fprintf(stderr, "usage: %s "
 82                         "[-D enc] "
 83                         "[-e ENC] "
 84                         "[file]\n", progname);
 85 }

 87 static int
 88 conv_latin_1(const struct buf *b)
 89 {
 90         size_t           i;
 91         unsigned char    cu;
 92         const char      *cp;

 94         cp = b->buf + (int)b->offs;

 96         /*
 97          * Latin-1 falls into the first 256 code-points of Unicode, so
 98          * there's no need for any sort of translation.  Just make the
 99          * 8-bit characters use the Unicode escape.
100          * Note that binary values 128 < v < 160 are passed through
101          * unmodified to mandoc.
102          */

104         for (i = b->offs; i < b->sz; i++) {
105                 cu = (unsigned char)*cp++;
106                 cu < 160U ? putchar(cu) : printf("\\[u%.4X]", cu);
107         }

109         return(1);
110 }

112 static int
113 conv_us_ascii(const struct buf *b)
114 {

116         /*
117          * US-ASCII has no conversion since it falls into the first 128
118          * bytes of Unicode.
119          */

121         fwrite(b->buf, 1, b->sz, stdout);
122         return(1);
123 }

125 static int
126 conv_utf_8(const struct buf *b)
127 {
```

```
128         int              state, be;
129         unsigned int     accum;
130         size_t           i;
131         unsigned char    cu;
132         const char      *cp;
133         const long       one = 1L;

135         cp = b->buf + (int)b->offs;
136         state = 0;
137         accum = 0U;
138         be = 0;

140         /* Quick test for big-endian value. */

142         if ( ! (*((const char *)(&one))))
143                 be = 1;

145         for (i = b->offs; i < b->sz; i++) {
146                 cu = (unsigned char)*cp++;
147                 if (state) {
148                         if ( ! (cu & 128) || (cu & 64)) {
149                                 /* Bad sequence header. */
150                                 return(0);
151                         }

153                         /* Accept only legitimate bit patterns. */

155                         if (cu > 191 || cu < 128) {
156                                 /* Bad in-sequence bits. */
157                                 return(0);
158                         }

160                         accum |= (cu & 63) << --state * 6;

162                         /*
163                          * Accum is held in little-endian order as
164                          * stipulated by the UTF-8 sequence coding.  We
165                          * need to convert to a native big-endian if our
166                          * architecture requires it.
167                          */

169                         if (0 == state && be)
170                                 accum = (accum >> 24) |
171                                         ((accum << 8) & 0x00FF0000) |
172                                         ((accum >> 8) & 0x0000FF00) |
173                                         (accum << 24);

175                         if (0 == state) {
176                                 accum < 128U ? putchar(accum) :
177                                         printf("\\[u%.4X]", accum);
178                                 accum = 0U;
179                         }
180                 } else if (cu & (1 << 7)) {
181                         /*
182                          * Entering a UTF-8 state:  if we encounter a
183                          * UTF-8 bitmask, calculate the expected UTF-8
184                          * state from it.
185                          */
186                         for (state = 0; state < 7; state++)
187                                 if ( ! (cu & (1 << (7 - state))))
188                                         break;

190                         /* Accept only legitimate bit patterns. */

192                         switch (state) {
193                         case (4):
```



```
194                                 if (cu <= 244 && cu >= 240) {
195                                         accum = (cu & 7) << 18;
196                                         break;
197                                 }
198                                 /* Bad 4-sequence start bits. */
199                                 return(0);
200                         case (3):
201                                 if (cu <= 239 && cu >= 224) {
202                                         accum = (cu & 15) << 12;
203                                         break;
204                                 }
205                                 /* Bad 3-sequence start bits. */
206                                 return(0);
207                         case (2):
208                                 if (cu <= 223 && cu >= 194) {
209                                         accum = (cu & 31) << 6;
210                                         break;
211                                 }
212                                 /* Bad 2-sequence start bits. */
213                                 return(0);
214                         default:
215                                 /* Bad sequence bit mask. */
216                                 return(0);
217                         }
218                         state--;
219                 } else
220                         putchar(cu);
221         }

223         if (0 != state) {
224                 /* Bad trailing bits. */
225                 return(0);
226         }

228         return(1);
229 }

231 static void
232 resize_buf(struct buf *buf, size_t initial)
233 {

235         buf->sz = buf->sz > initial / 2 ?
236                 2 * buf->sz : initial;

238         buf->buf = realloc(buf->buf, buf->sz);
239         if (NULL == buf->buf) {
240                 perror(NULL);
241                 exit(EXIT_FAILURE);
242         }
243 }

245 static int
246 read_whole_file(const char *f, int fd,
247                 struct buf *fb, int *with_mmap)
248 {
249         size_t           off;
250         ssize_t          ssz;

252 #ifdef  HAVE_MMAP
253         struct stat      st;
254         if (-1 == fstat(fd, &st)) {
255                 perror(f);
256                 return(0);
257         }

259         /*
```

```
260          * If we're a regular file, try just reading in the whole entry
261          * via mmap().  This is faster than reading it into blocks, and
262          * since each file is only a few bytes to begin with, I'm not
263          * concerned that this is going to tank any machines.
264          */
265
266         if (S_ISREG(st.st_mode) && st.st_size >= (1U << 31)) {
267                 fprintf(stderr, "%s: input too large\n", f);
268                 return(0);
269         }
270
271         if (S_ISREG(st.st_mode)) {
272                 *with_mmap = 1;
273                 fb->sz = (size_t)st.st_size;
274                 fb->buf = mmap(NULL, fb->sz, PROT_READ,
275                                 MAP_FILE|MAP_SHARED, fd, 0);
276                 if (fb->buf != MAP_FAILED)
277                         return(1);
278         }
279 #endif
280
281         /*
282          * If this isn't a regular file (like, say, stdin), then we must
283          * go the old way and just read things in bit by bit.
284          */
285
286         *with_mmap = 0;
287         off = 0;
288         fb->sz = 0;
289         fb->buf = NULL;
290         for (;;) {
291                 if (off == fb->sz && fb->sz == (1U << 31)) {
292                         fprintf(stderr, "%s: input too large\n", f);
293                         break;
294                 }
295
296                 if (off == fb->sz)
297                         resize_buf(fb, 65536);
298
299                 ssz = read(fd, fb->buf + (int)off, fb->sz - off);
300                 if (ssz == 0) {
301                         fb->sz = off;
302                         return(1);
303                 }
304                 if (ssz == -1) {
305                         perror(f);
306                         break;
307                 }
308                 off += (size_t)ssz;
309         }
310
311         free(fb->buf);
312         fb->buf = NULL;
313         return(0);
314 }
315
316 static int
317 cue_enc(const struct buf *b, size_t *offs, enum enc *enc)
318 {
319         const char      *ln, *eoln, *eoph;
320         size_t           sz, phsz, nsz;
321         int              i;
322
323         ln = b->buf + (int)*offs;
324         sz = b->sz - *offs;
```



```
325
326         /* Look for the end-of-line. */
327
328         if (NULL == (eoln = memchr(ln, '\n', sz)))
329                 return(-1);
330
331         /* Set next-line marker. */
332
333         *offs = (size_t)((eoln + 1) - b->buf);
334
335         /* Check if we have the correct header/trailer. */
336
337         if ((sz = (size_t)(eoln - ln)) < 10 ||
338                         memcmp(ln, ".\\\" -*-", 7) ||
339                         memcmp(eoln - 3, "-*-", 3))
340                 return(0);
341
342         /* Move after the header and adjust for the trailer. */
343
344         ln += 7;
345         sz -= 10;
346
347         while (sz > 0) {
348                 while (sz > 0 && ' ' == *ln) {
349                         ln++;
350                         sz--;
351                 }
352                 if (0 == sz)
353                         break;
354
355                 /* Find the end-of-phrase marker (or eoln). */
356
357                 if (NULL == (eoph = memchr(ln, ';', sz)))
358                         eoph = eoln - 3;
359                 else
360                         eoph++;
361
362                 /* Only account for the "coding" phrase. */
363
364                 if ((phsz = (size_t)(eoph - ln)) < 7 ||
365                                 strncasecmp(ln, "coding:", 7)) {
366                         sz -= phsz;
367                         ln += phsz;
368                         continue;
369                 }
370
371                 sz -= 7;
372                 ln += 7;
373
374                 while (sz > 0 && ' ' == *ln) {
375                         ln++;
376                         sz--;
377                 }
378                 if (0 == sz)
379                         break;
380
381                 /* Check us against known encodings. */
382
383                 for (i = 0; i < (int)ENC__MAX; i++) {
384                         nsz = strlen(encs[i].name);
385                         if (phsz < nsz)
386                                 continue;
387                         if (strncasecmp(ln, encs[i].name, nsz))
388                                 continue;
389
390                         *enc = (enum enc)i;
391                         return(1);
```

```
392                 }
394                 /* Unknown encoding. */
396                 *enc = ENC__MAX;
397                 return(1);
398         }
400         return(0);
401 }
403 int
404 main(int argc, char *argv[])
405 {
406         int              i, ch, map, fd, rc;
407         struct buf       b;
408         const char      *fn;
409         enum enc         enc, def;
410         unsigned char    bom[3] = { 0xEF, 0xBB, 0xBF };
411         size_t           offs;
412         extern int       optind;
413         extern char     *optarg;
415         progname = strrchr(argv[0], '/');
416         if (progname == NULL)
417                 progname = argv[0];
418         else
419                 ++progname;
421         fn = "<stdin>";
422         fd = STDIN_FILENO;
423         rc = EXIT_FAILURE;
424         enc = def = ENC__MAX;
425         map = 0;
427         memset(&b, 0, sizeof(struct buf));
429         while (-1 != (ch = getopt(argc, argv, "D:e:rdvh")))
430                 switch (ch) {
431                 case ('D'):
432                         /* FALLTHROUGH */
433                 case ('e'):
434                         for (i = 0; i < (int)ENC__MAX; i++) {
435                                 if (strcasecmp(optarg, encs[i].name))
436                                         continue;
437                                 break;
438                         }
439                         if (i < (int)ENC__MAX) {
440                                 if ('D' == ch)
441                                         def = (enum enc)i;
442                                 else
443                                         enc = (enum enc)i;
444                                 break;
445                         }
447                         fprintf(stderr, "%s: Bad encoding\n", optarg);
448                         return(EXIT_FAILURE);
449                 case ('r'):
450                         /* FALLTHROUGH */
451                 case ('d'):
452                         /* FALLTHROUGH */
453                 case ('v'):
454                         /* Compatibility with GNU preconv. */
455                         break;
456                 case ('h'):
457                         /* Compatibility with GNU preconv. */
```



```
458                         /* FALLTHROUGH */
459                 default:
460                         usage();
461                         return(EXIT_FAILURE);
462                 }
464         argc -= optind;
465         argv += optind;
466
467         /*
468          * Open and read the first argument on the command-line.
469          * If we don't have one, we default to stdin.
470          */
472         if (argc > 0) {
473                 fn = *argv;
474                 fd = open(fn, O_RDONLY, 0);
475                 if (-1 == fd) {
476                         perror(fn);
477                         return(EXIT_FAILURE);
478                 }
479         }
481         if ( ! read_whole_file(fn, fd, &b, &map))
482                 goto out;
484         /* Try to read the UTF-8 BOM. */
486         if (ENC__MAX == enc)
487                 if (b.sz > 3 && 0 == memcmp(b.buf, bom, 3)) {
488                         b.offs = 3;
489                         enc = ENC_UTF_8;
490                 }
492         /* Try reading from the "-*-" cue. */
494         if (ENC__MAX == enc) {
495                 offs = b.offs;
496                 ch = cue_enc(&b, &offs, &enc);
497                 if (0 == ch)
498                         ch = cue_enc(&b, &offs, &enc);
499         }
501         /*
502          * No encoding has been detected.
503          * Thus, we either fall into our default encoder, if specified,
504          * or use Latin-1 if all else fails.
505          */
507         if (ENC__MAX == enc)
508                 enc = ENC__MAX == def ? ENC_LATIN_1 : def;
510         if ( ! (*encs[(int)enc].conv)(&b)) {
511                 fprintf(stderr, "%s: Bad encoding\n", fn);
512                 goto out;
513         }
515         rc = EXIT_SUCCESS;
516 out:
517 #ifdef  HAVE_MMAP
518         if (map)
519                 munmap(b.buf, b.sz);
520         else
521 #endif
522                 free(b.buf);
```

```
524         if (fd > STDIN_FILENO)
525                 close(fd);

527         return(rc);
528 }
```

```
**********************************************************
    2091 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/predefs.in
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: predefs.in,v 1.3 2011/07/31 11:36:49 schwarze Exp $ */
/*
 * Copyright (c) 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.
 */

/*
 * The predefined-string translation tables.  Each corresponds to a
 * predefined strings from (e.g.) tmac/mdoc/doc-nroff.  The left-hand
 * side corresponds to the input sequence (\*x, \*(xx and so on).  The
 * right-hand side is what's produced by libroff.
 *
 * XXX - C-escape strings!
 * XXX - update PREDEF_MAX in roff.c if adding more!
 */

PREDEF("Am", "&")
PREDEF("Ba", "|")
PREDEF("Ge", "\\(>=")
PREDEF("Gt", ">")
PREDEF("If", "infinity")
PREDEF("Le", "\\(<=")
PREDEF("Lq", "\\(lq")
PREDEF("Lt", "<")
PREDEF("Na", "NaN")
PREDEF("Ne", "\\(!=")
PREDEF("Pi", "pi")
PREDEF("Pm", "\\(+-")
PREDEF("Rq", "\\(rq")
PREDEF("left-bracket", "[")
PREDEF("left-parenthesis", "(")
PREDEF("lp", "(")
PREDEF("left-singlequote", "\\(oq")
PREDEF("q", "\\(dq")
PREDEF("quote-left", "\\(oq")
PREDEF("quote-right", "\\(cq")
PREDEF("R", "\\(rg")
PREDEF("right-bracket", "]")
PREDEF("right-parenthesis", ")")
PREDEF("rp", ")")
PREDEF("right-singlequote", "\\(cq")
PREDEF("Tm", "(Tm)")
PREDEF("Px", "POSIX")
PREDEF("Ai", "ANSI")
PREDEF("\'", "\\\'")
PREDEF("aa", "\\(aa")
PREDEF("ga", "\\(ga")
PREDEF("'", "\\'")
PREDEF("lq", "\\(lq")
PREDEF("rq", "\\(rq")
```



```
PREDEF("ua", "\\(ua")
PREDEF("va", "\\(va")
PREDEF("<=", "\\(<=")
PREDEF(">=", "\\(>=")
```

```
**********************************************************
   19263 Tue Jul 15 13:48:07 2014
new/usr/src/cmd/mandoc/read.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: read.c,v 1.28 2012/02/16 20:51:31 joerg Exp $ */
/*
 * 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

#ifdef HAVE_MMAP
# include <sys/stat.h>
# include <sys/mman.h>
#endif

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

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

#ifndef MAP_FILE
#define MAP_FILE        0
#endif

#define REPARSE_LIMIT   1000

struct  buf {
        char            *buf; /* binary input buffer */
        size_t           sz; /* size of binary buffer */
};

struct  mparse {
        enum mandoclevel  file_status; /* status of current parse */
        enum mandoclevel  wlevel; /* ignore messages below this */
        int               line; /* line number in the file */
        enum mparset      inttype; /* which parser to use */
        struct man       *pman; /* persistent man parser */
        struct mdoc      *pmdoc; /* persistent mdoc parser */
        struct man       *man; /* man parser */
```



```
        struct mdoc      *mdoc; /* mdoc parser */
        struct roff      *roff; /* roff parser (!NULL) */
        int               reparse_count; /* finite interp. stack */
        mandocmsg         mmsg; /* warning/error message handler */
        void             *arg; /* argument to mmsg */
        const char       *file;
        struct buf       *secondary;
};

static  void      resize_buf(struct buf *, size_t);
static  void      mparse_buf_r(struct mparse *, struct buf, int);
static  void      mparse_readfd_r(struct mparse *, int, const char *, int);
static  void      pset(const char *, int, struct mparse *);
static  int       read_whole_file(const char *, int, struct buf *, int *);
static  void      mparse_end(struct mparse *);

static  const enum mandoclevel    mandoclimits[MANDOCLEVEL_MAX] = {
        MANDOCERR_OK,
        MANDOCERR_WARNING,
        MANDOCERR_WARNING,
        MANDOCERR_ERROR,
        MANDOCERR_FATAL,
        MANDOCERR_MAX,
        MANDOCERR_MAX
};

static  const char * const      mandocerrs[MANDOCERR_MAX] = {
        "ok",

        "generic warning",

        /* related to the prologue */
        "no title in document",
        "document title should be all caps",
        "unknown manual section",
        "date missing, using today's date",
        "cannot parse date, using it verbatim",
        "prologue macros out of order",
        "duplicate prologue macro",
        "macro not allowed in prologue",
        "macro not allowed in body",

        /* related to document structure */
        ".so is fragile, better use ln(1)",
        "NAME section must come first",
        "bad NAME section contents",
        "manual name not yet set",
        "sections out of conventional order",
        "duplicate section name",
        "section not in conventional manual section",

        /* related to macros and nesting */
        "skipping obsolete macro",
        "skipping paragraph macro",
        "skipping no-space macro",
        "blocks badly nested",
        "child violates parent syntax",
        "nested displays are not portable",
        "already in literal mode",
        "line scope broken",

        /* related to missing macro arguments */
        "skipping empty macro",
        "argument count wrong",
        "missing display type",
        "list type must come first",
```

```
128         "tag lists require a width argument",
129         "missing font type",
130         "skipping end of block that is not open",

132         /* related to bad macro arguments */
133         "skipping argument",
134         "duplicate argument",
135         "duplicate display type",
136         "duplicate list type",
137         "unknown AT&T UNIX version",
138         "bad Boolean value",
139         "unknown font",
140         "unknown standard specifier",
141         "bad width argument",

143         /* related to plain text */
144         "blank line in non-literal context",
145         "tab in non-literal context",
146         "end of line whitespace",
147         "bad comment style",
148         "bad escape sequence",
149         "unterminated quoted string",

151         /* related to equations */
152         "unexpected literal in equation",
153
154         "generic error",

156         /* related to equations */
157         "unexpected equation scope closure",
158         "equation scope open on exit",
159         "overlapping equation scopes",
160         "unexpected end of equation",
161         "equation syntax error",

163         /* related to tables */
164         "bad table syntax",
165         "bad table option",
166         "bad table layout",
167         "no table layout cells specified",
168         "no table data cells specified",
169         "ignore data in cell",
170         "data block still open",
171         "ignoring extra data cells",

173         "input stack limit exceeded, infinite loop?",
174         "skipping bad character",
175         "escaped character not allowed in a name",
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 end of block that is not open",
181         "missing end of block",
182         "scope open on exit",
183         "uname(3) system call failed",
184         "macro requires line argument(s)",
185         "macro requires body argument(s)",
186         "macro requires argument(s)",
187         "missing list type",
188         "line argument(s) will be lost",
189         "body argument(s) will be lost",

191         "generic fatal error",

193         "not a manual",
```



```
194         "column syntax is inconsistent",
195         "NOT IMPLEMENTED: .Bd -file",
196         "argument count wrong, violates syntax",
197         "child violates parent syntax",
198         "argument count wrong, violates syntax",
199         "NOT IMPLEMENTED: .so with absolute path or \"..\"",
200         "no document body",
201         "no document prologue",
202         "static buffer exhausted",
203 };

205 static  const char * const      mandoclevels[MANDOCLEVEL_MAX] = {
206         "SUCCESS",
207         "RESERVED",
208         "WARNING",
209         "ERROR",
210         "FATAL",
211         "BADARG",
212         "SYSERR"
213 };

215 static void
216 resize_buf(struct buf *buf, size_t initial)
217 {

219         buf->sz = buf->sz > initial/2 ? 2 * buf->sz : initial;
220         buf->buf = mandoc_realloc(buf->buf, buf->sz);
221 }

223 static void
224 pset(const char *buf, int pos, struct mparse *curp)
225 {
226         int              i;

228         /*
229          * Try to intuit which kind of manual parser should be used.  If
230          * passed in by command-line (-man, -mdoc), then use that
231          * explicitly.  If passed as -mandoc, then try to guess from the
232          * line: either skip dot-lines, use -mdoc when finding '.Dt', or
233          * default to -man, which is more lenient.
234          *
235          * Separate out pmdoc/pman from mdoc/man: the first persists
236          * through all parsers, while the latter is used per-parse.
237          */

239         if ('.' == buf[0] || '\'' == buf[0]) {
240                 for (i = 1; buf[i]; i++)
241                         if (' ' != buf[i] && '\t' != buf[i])
242                                 break;
243                 if ('\0' == buf[i])
244                         return;
245         }

247         switch (curp->inttype) {
248         case (MPARSE_MDOC):
249                 if (NULL == curp->pmdoc)
250                         curp->pmdoc = mdoc_alloc(curp->roff, curp);
251                 assert(curp->pmdoc);
252                 curp->mdoc = curp->pmdoc;
253                 return;
254         case (MPARSE_MAN):
255                 if (NULL == curp->pman)
256                         curp->pman = man_alloc(curp->roff, curp);
257                 assert(curp->pman);
258                 curp->man = curp->pman;
259                 return;
```

```
260         default:
261                 break;
262         }

264         if (pos >= 3 && 0 == memcmp(buf, ".Dd", 3))  {
265                 if (NULL == curp->pmdoc)
266                         curp->pmdoc = mdoc_alloc(curp->roff, curp);
267                 assert(curp->pmdoc);
268                 curp->mdoc = curp->pmdoc;
269                 return;
270         }

272         if (NULL == curp->pman)
273                 curp->pman = man_alloc(curp->roff, curp);
274         assert(curp->pman);
275         curp->man = curp->pman;
276 }

278 /*
279  * Main parse routine for an opened file.  This is called for each
280  * opened file and simply loops around the full input file, possibly
281  * nesting (i.e., with 'so').
282  */
283 static void
284 mparse_buf_r(struct mparse *curp, struct buf blk, int start)
285 {
286         const struct tbl_span   *span;
287         struct buf       ln;
288         enum rofferr     rr;
289         int              i, of, rc;
290         int              pos; /* byte number in the ln buffer */
291         int              lnn; /* line number in the real file */
292         unsigned char    c;

294         memset(&ln, 0, sizeof(struct buf));

296         lnn = curp->line;
297         pos = 0;

299         for (i = 0; i < (int)blk.sz; ) {
300                 if (0 == pos && '\0' == blk.buf[i])
301                         break;

303                 if (start) {
304                         curp->line = lnn;
305                         curp->reparse_count = 0;
306                 }

308                 while (i < (int)blk.sz && (start || '\0' != blk.buf[i])) {

310                         /*
311                          * When finding an unescaped newline character,
312                          * leave the character loop to process the line.
313                          * Skip a preceding carriage return, if any.
314                          */

316                         if ('\r' == blk.buf[i] && i + 1 < (int)blk.sz &&
317                             '\n' == blk.buf[i + 1])
318                                 ++i;
319                         if ('\n' == blk.buf[i]) {
320                                 ++i;
321                                 ++lnn;
322                                 break;
323                         }

325                         /*
```



```
326                          * Warn about bogus characters.  If you're using
327                          * non-ASCII encoding, you're screwing your
328                          * readers.  Since I'd rather this not happen,
329                          * I'll be helpful and replace these characters
330                          * with "?", so we don't display gibberish.
331                          * Note to manual writers: use special characters.
332                          */

334                         c = (unsigned char) blk.buf[i];

336                         if ( ! (isascii(c) &&
337                                         (isgraph(c) || isblank(c)))) {
338                                 mandoc_msg(MANDOCERR_BADCHAR, curp,
339                                                 curp->line, pos, NULL);
340                                 i++;
341                                 if (pos >= (int)ln.sz)
342                                         resize_buf(&ln, 256);
343                                 ln.buf[pos++] = '?';
344                                 continue;
345                         }

347                         /* Trailing backslash = a plain char. */

349                         if ('\\' != blk.buf[i] || i + 1 == (int)blk.sz) {
350                                 if (pos >= (int)ln.sz)
351                                         resize_buf(&ln, 256);
352                                 ln.buf[pos++] = blk.buf[i++];
353                                 continue;
354                         }

356                         /*
357                          * Found escape and at least one other character.
358                          * When it's a newline character, skip it.
359                          * When there is a carriage return in between,
360                          * skip that one as well.
361                          */

363                         if ('\r' == blk.buf[i + 1] && i + 2 < (int)blk.sz &&
364                             '\n' == blk.buf[i + 2])
365                                 ++i;
366                         if ('\n' == blk.buf[i + 1]) {
367                                 i += 2;
368                                 ++lnn;
369                                 continue;
370                         }

372                         if ('"' == blk.buf[i + 1] || '#' == blk.buf[i + 1]) {
373                                 i += 2;
374                                 /* Comment, skip to end of line */
375                                 for (; i < (int)blk.sz; ++i) {
376                                         if ('\n' == blk.buf[i]) {
377                                                 ++i;
378                                                 ++lnn;
379                                                 break;
380                                         }
381                                 }

383                                 /* Backout trailing whitespaces */
384                                 for (; pos > 0; --pos) {
385                                         if (ln.buf[pos - 1] != ' ')
386                                                 break;
387                                         if (pos > 2 && ln.buf[pos - 2] == '\\')
388                                                 break;
389                                 }
390                                 break;
391                         }
```

```
393                         /* Some other escape sequence, copy & cont. */
395                         if (pos + 1 >= (int)ln.sz)
396                                 resize_buf(&ln, 256);
398                         ln.buf[pos++] = blk.buf[i++];
399                         ln.buf[pos++] = blk.buf[i++];
400                 }
402                 if (pos >= (int)ln.sz)
403                         resize_buf(&ln, 256);
405                 ln.buf[pos] = '\0';
407                 /*
408                  * A significant amount of complexity is contained by
409                  * the roff preprocessor.  It's line-oriented but can be
410                  * expressed on one line, so we need at times to
411                  * readjust our starting point and re-run it.  The roff
412                  * preprocessor can also readjust the buffers with new
413                  * data, so we pass them in wholesale.
414                  */
416                 of = 0;
418                 /*
419                  * Maintain a lookaside buffer of all parsed lines.  We
420                  * only do this if mparse_keep() has been invoked (the
421                  * buffer may be accessed with mparse_getkeep()).
422                  */
424                 if (curp->secondary) {
425                         curp->secondary->buf =
426                                 mandoc_realloc
427                                 (curp->secondary->buf,
428                                  curp->secondary->sz + pos + 2);
429                         memcpy(curp->secondary->buf +
430                                         curp->secondary->sz,
431                                         ln.buf, pos);
432                         curp->secondary->sz += pos;
433                         curp->secondary->buf
434                                 [curp->secondary->sz] = '\n';
435                         curp->secondary->sz++;
436                         curp->secondary->buf
437                                 [curp->secondary->sz] = '\0';
438                 }
439 rerun:
440                 rr = roff_parseln
441                         (curp->roff, curp->line,
442                          &ln.buf, &ln.sz, of, &of);
444                 switch (rr) {
445                 case (ROFF_REPARSE):
446                         if (REPARSE_LIMIT >= ++curp->reparse_count)
447                                 mparse_buf_r(curp, ln, 0);
448                         else
449                                 mandoc_msg(MANDOCERR_ROFFLOOP, curp,
450                                         curp->line, pos, NULL);
451                         pos = 0;
452                         continue;
453                 case (ROFF_APPEND):
454                         pos = (int)strlen(ln.buf);
455                         continue;
456                 case (ROFF_RERUN):
457                         goto rerun;
```



```
458                 case (ROFF_IGN):
459                         pos = 0;
460                         continue;
461                 case (ROFF_ERR):
462                         assert(MANDOCLEVEL_FATAL <= curp->file_status);
463                         break;
464                 case (ROFF_SO):
465                         /*
466                          * We remove 'so' clauses from our lookaside
467                          * buffer because we're going to descend into
468                          * the file recursively.
469                          */
470                         if (curp->secondary)
471                                 curp->secondary->sz -= pos + 1;
472                         mparse_readfd_r(curp, -1, ln.buf + of, 1);
473                         if (MANDOCLEVEL_FATAL <= curp->file_status)
474                                 break;
475                         pos = 0;
476                         continue;
477                 default:
478                         break;
479                 }
481                 /*
482                  * If we encounter errors in the recursive parse, make
483                  * sure we don't continue parsing.
484                  */
486                 if (MANDOCLEVEL_FATAL <= curp->file_status)
487                         break;
489                 /*
490                  * If input parsers have not been allocated, do so now.
491                  * We keep these instanced between parsers, but set them
492                  * locally per parse routine since we can use different
493                  * parsers with each one.
494                  */
496                 if ( ! (curp->man || curp->mdoc))
497                         pset(ln.buf + of, pos - of, curp);
499                 /*
500                  * Lastly, push down into the parsers themselves.  One
501                  * of these will have already been set in the pset()
502                  * routine.
503                  * If libroff returns ROFF_TBL, then add it to the
504                  * currently open parse.  Since we only get here if
505                  * there does exist data (see tbl_data.c), we're
506                  * guaranteed that something's been allocated.
507                  * Do the same for ROFF_EQN.
508                  */
510                 rc = -1;
512                 if (ROFF_TBL == rr)
513                         while (NULL != (span = roff_span(curp->roff))) {
514                                 rc = curp->man ?
515                                         man_addspan(curp->man, span) :
516                                         mdoc_addspan(curp->mdoc, span);
517                                 if (0 == rc)
518                                         break;
519                         }
520                 else if (ROFF_EQN == rr)
521                         rc = curp->mdoc ?
522                                 mdoc_addeqn(curp->mdoc,
523                                         roff_eqn(curp->roff)) :
```

```
524                                 man_addeqn(curp->man,
525                                         roff_eqn(curp->roff));
526                 else if (curp->man || curp->mdoc)
527                         rc = curp->man ?
528                                 man_parseln(curp->man,
529                                         curp->line, ln.buf, of) :
530                                 mdoc_parseln(curp->mdoc,
531                                         curp->line, ln.buf, of);

533                 if (0 == rc) {
534                         assert(MANDOCLEVEL_FATAL <= curp->file_status);
535                         break;
536                 }

538                 /* Temporary buffers typically are not full. */

540                 if (0 == start && '\0' == blk.buf[i])
541                         break;

543                 /* Start the next input line. */

545                 pos = 0;
546         }

548         free(ln.buf);
549 }

551 static int
552 read_whole_file(const char *file, int fd, struct buf *fb, int *with_mmap)
553 {
554         size_t           off;
555         ssize_t          ssz;

557 #ifdef  HAVE_MMAP
558         struct stat      st;
559         if (-1 == fstat(fd, &st)) {
560                 perror(file);
561                 return(0);
562         }

564         /*
565          * If we're a regular file, try just reading in the whole entry
566          * via mmap().  This is faster than reading it into blocks, and
567          * since each file is only a few bytes to begin with, I'm not
568          * concerned that this is going to tank any machines.
569          */

571         if (S_ISREG(st.st_mode)) {
572                 if (st.st_size >= (1U << 31)) {
573                         fprintf(stderr, "%s: input too large\n", file);
574                         return(0);
575                 }
576                 *with_mmap = 1;
577                 fb->sz = (size_t)st.st_size;
578                 fb->buf = mmap(NULL, fb->sz, PROT_READ,
579                                 MAP_FILE|MAP_SHARED, fd, 0);
580                 if (fb->buf != MAP_FAILED)
581                         return(1);
582         }
583 #endif

585         /*
586          * If this isn't a regular file (like, say, stdin), then we must
587          * go the old way and just read things in bit by bit.
588          */
```



```
590         *with_mmap = 0;
591         off = 0;
592         fb->sz = 0;
593         fb->buf = NULL;
594         for (;;) {
595                 if (off == fb->sz) {
596                         if (fb->sz == (1U << 31)) {
597                                 fprintf(stderr, "%s: input too large\n", file);
598                                 break;
599                         }
600                         resize_buf(fb, 65536);
601                 }
602                 ssz = read(fd, fb->buf + (int)off, fb->sz - off);
603                 if (ssz == 0) {
604                         fb->sz = off;
605                         return(1);
606                 }
607                 if (ssz == -1) {
608                         perror(file);
609                         break;
610                 }
611                 off += (size_t)ssz;
612         }

614         free(fb->buf);
615         fb->buf = NULL;
616         return(0);
617 }

619 static void
620 mparse_end(struct mparse *curp)
621 {

623         if (MANDOCLEVEL_FATAL <= curp->file_status)
624                 return;

626         if (curp->mdoc && ! mdoc_endparse(curp->mdoc)) {
627                 assert(MANDOCLEVEL_FATAL <= curp->file_status);
628                 return;
629         }

631         if (curp->man && ! man_endparse(curp->man)) {
632                 assert(MANDOCLEVEL_FATAL <= curp->file_status);
633                 return;
634         }

636         if ( ! (curp->man || curp->mdoc)) {
637                 mandoc_msg(MANDOCERR_NOTMANUAL, curp, 1, 0, NULL);
638                 curp->file_status = MANDOCLEVEL_FATAL;
639                 return;
640         }

642         roff_endparse(curp->roff);
643 }

645 static void
646 mparse_parse_buffer(struct mparse *curp, struct buf blk, const char *file,
647                 int re)
648 {
649         const char      *svfile;

651         /* Line number is per-file. */
652         svfile = curp->file;
653         curp->file = file;
654         curp->line = 1;
```

```
656         mparse_buf_r(curp, blk, 1);

658         if (0 == re && MANDOCLEVEL_FATAL > curp->file_status)
659                 mparse_end(curp);

661         curp->file = svfile;
662 }

664 enum mandoclevel
665 mparse_readmem(struct mparse *curp, const void *buf, size_t len,
666                 const char *file)
667 {
668         struct buf blk;

670         blk.buf = UNCONST(buf);
671         blk.sz = len;

673         mparse_parse_buffer(curp, blk, file, 0);
674         return(curp->file_status);
675 }

677 static void
678 mparse_readfd_r(struct mparse *curp, int fd, const char *file, int re)
679 {
680         struct buf       blk;
681         int              with_mmap;

683         if (-1 == fd)
684                 if (-1 == (fd = open(file, O_RDONLY, 0))) {
685                         perror(file);
686                         curp->file_status = MANDOCLEVEL_SYSERR;
687                         return;
688                 }
689         /*
690          * Run for each opened file; may be called more than once for
691          * each full parse sequence if the opened file is nested (i.e.,
692          * from ‘so’).  Simply sucks in the whole file and moves into
693          * the parse phase for the file.
694          */

696         if ( ! read_whole_file(file, fd, &blk, &with_mmap)) {
697                 curp->file_status = MANDOCLEVEL_SYSERR;
698                 return;
699         }

701         mparse_parse_buffer(curp, blk, file, re);

703 #ifdef  HAVE_MMAP
704         if (with_mmap)
705                 munmap(blk.buf, blk.sz);
706         else
707 #endif
708                 free(blk.buf);

710         if (STDIN_FILENO != fd && -1 == close(fd))
711                 perror(file);
712 }

714 enum mandoclevel
715 mparse_readfd(struct mparse *curp, int fd, const char *file)
716 {

718         mparse_readfd_r(curp, fd, file, 0);
719         return(curp->file_status);
720 }
```



```
722 struct mparse *
723 mparse_alloc(enum mparset inttype, enum mandoclevel wlevel, mandocmsg mmsg, void
724 {
725         struct mparse   *curp;

727         assert(wlevel <= MANDOCLEVEL_FATAL);

729         curp = mandoc_calloc(1, sizeof(struct mparse));

731         curp->wlevel = wlevel;
732         curp->mmsg = mmsg;
733         curp->arg = arg;
734         curp->inttype = inttype;

736         curp->roff = roff_alloc(curp);
737         return(curp);
738 }

740 void
741 mparse_reset(struct mparse *curp)
742 {

744         roff_reset(curp->roff);

746         if (curp->mdoc)
747                 mdoc_reset(curp->mdoc);
748         if (curp->man)
749                 man_reset(curp->man);
750         if (curp->secondary)
751                 curp->secondary->sz = 0;

753         curp->file_status = MANDOCLEVEL_OK;
754         curp->mdoc = NULL;
755         curp->man = NULL;
756 }

758 void
759 mparse_free(struct mparse *curp)
760 {

762         if (curp->pmdoc)
763                 mdoc_free(curp->pmdoc);
764         if (curp->pman)
765                 man_free(curp->pman);
766         if (curp->roff)
767                 roff_free(curp->roff);
768         if (curp->secondary)
769                 free(curp->secondary->buf);

771         free(curp->secondary);
772         free(curp);
773 }

775 void
776 mparse_result(struct mparse *curp, struct mdoc **mdoc, struct man **man)
777 {

779         if (mdoc)
780                 *mdoc = curp->mdoc;
781         if (man)
782                 *man = curp->man;
783 }

785 void
786 mandoc_vmsg(enum mandocerr t, struct mparse *m,
787                 int ln, int pos, const char *fmt, ...)
```

```
788 {
789         char             buf[256];
790         va_list          ap;
791
792         va_start(ap, fmt);
793         vsnprintf(buf, sizeof(buf) - 1, fmt, ap);
794         va_end(ap);
795
796         mandoc_msg(t, m, ln, pos, buf);
797 }
798
799 void
800 mandoc_msg(enum mandocerr er, struct mparse *m,
801                 int ln, int col, const char *msg)
802 {
803         enum mandoclevel level;
804
805         level = MANDOCLEVEL_FATAL;
806         while (er < mandoclimits[level])
807                 level--;
808
809         if (level < m->wlevel)
810                 return;
811
812         if (m->mmsg)
813                 (*m->mmsg)(er, level, m->file, ln, col, msg);
814
815         if (m->file_status < level)
816                 m->file_status = level;
817 }
818
819 const char *
820 mparse_strerror(enum mandocerr er)
821 {
822
823         return(mandocerrs[er]);
824 }
825
826 const char *
827 mparse_strlevel(enum mandoclevel lvl)
828 {
829         return(mandoclevels[lvl]);
830 }
831
832 void
833 mparse_keep(struct mparse *p)
834 {
835
836         assert(NULL == p->secondary);
837         p->secondary = mandoc_calloc(1, sizeof(struct buf));
838 }
839
840 const char *
841 mparse_getkeep(const struct mparse *p)
842 {
843
844         assert(p->secondary);
845         return(p->secondary->sz ? p->secondary->buf : NULL);
846 }
```

```
**********************************************************
   37963 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/roff.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: roff.c,v 1.172 2011/10/24 21:41:45 schwarze Exp $ */
/*
 * Copyright (c) 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 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 <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

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

/* Maximum number of nested if-else conditionals. */
#define RSTACK_MAX      128

/* Maximum number of string expansions per line, to break infinite loops. */
#define EXPAND_LIMIT    1000

enum    rofft {
        ROFF_ad,
        ROFF_am,
        ROFF_ami,
        ROFF_am1,
        ROFF_de,
        ROFF_dei,
        ROFF_de1,
        ROFF_ds,
        ROFF_el,
        ROFF_hy,
        ROFF_ie,
        ROFF_if,
        ROFF_ig,
        ROFF_it,
        ROFF_ne,
        ROFF_nh,
        ROFF_nr,
        ROFF_ns,
        ROFF_ps,
        ROFF_rm,
        ROFF_so,
        ROFF_ta,
        ROFF_tr,
        ROFF_TS,
```



```
        ROFF_TE,
        ROFF_T_,
        ROFF_EQ,
        ROFF_EN,
        ROFF_cblock,
        ROFF_ccond,
        ROFF_USERDEF,
        ROFF_MAX
};

enum    roffrule {
        ROFFRULE_ALLOW,
        ROFFRULE_DENY
};

/*
 * A single register entity.  If "set" is zero, the value of the
 * register should be the default one, which is per-register.
 * Registers are assumed to be unsigned ints for now.
 */
struct  reg {
        int              set; /* whether set or not */
        unsigned int     u; /* unsigned integer */
};

/*
 * An incredibly-simple string buffer.
 */
struct  roffstr {
        char            *p; /* nil-terminated buffer */
        size_t           sz; /* saved strlen(p) */
};

/*
 * A key-value roffstr pair as part of a singly-linked list.
 */
struct  roffkv {
        struct roffstr   key;
        struct roffstr   val;
        struct roffkv   *next; /* next in list */
};

struct  roff {
        struct mparse   *parse; /* parse point */
        struct roffnode *last; /* leaf of stack */
        enum roffrule    rstack[RSTACK_MAX]; /* stack of !‘ie’ rules */
        int              rstackpos; /* position in rstack */
        struct reg       regs[REG__MAX];
        struct roffkv   *strtab; /* user-defined strings & macros */
        struct roffkv   *xmbtab; /* multi-byte trans table (‘tr’) */
        struct roffstr  *xtab; /* single-byte trans table (‘tr’) */
        const char      *current_string; /* value of last called user macro */
        struct tbl_node *first_tbl; /* first table parsed */
        struct tbl_node *last_tbl; /* last table parsed */
        struct tbl_node *tbl; /* current table being parsed */
        struct eqn_node *last_eqn; /* last equation parsed */
        struct eqn_node *first_eqn; /* first equation parsed */
        struct eqn_node *eqn; /* current equation being parsed */
};

struct  roffnode {
        enum rofft       tok; /* type of node */
        struct roffnode *parent; /* up one in stack */
        int              line; /* parse line */
        int              col; /* parse col */
        char            *name; /* node name, e.g. macro name */
```

```
128         char            *end; /* end-rules: custom token */
129         int              endspan; /* end-rules: next-line or infty */
130         enum roffrule    rule; /* current evaluation rule */
131 };

133 #define ROFF_ARGS        struct roff *r, /* parse ctx */ \
134                          enum rofft tok, /* tok of macro */ \
135                          char **bufp, /* input buffer */ \
136                          size_t *szp, /* size of input buffer */ \
137                          int ln, /* parse line */ \
138                          int ppos, /* original pos in buffer */ \
139                          int pos, /* current pos in buffer */ \
140                          int *offs /* reset offset of buffer data */

142 typedef enum rofferr (*roffproc)(ROFF_ARGS);

144 struct  roffmac {
145         const char      *name; /* macro name */
146         roffproc         proc; /* process new macro */
147         roffproc         text; /* process as child text of macro */
148         roffproc         sub; /* process as child of macro */
149         int              flags;
150 #define ROFFMAC_STRUCT  (1 << 0) /* always interpret */
151         struct roffmac  *next;
152 };

154 struct  predef {
155         const char      *name; /* predefined input name */
156         const char      *str; /* replacement symbol */
157 };

159 #define PREDEF(__name, __str) \
160         { (__name), (__str) },

162 static  enum rofft       roffhash_find(const char *, size_t);
163 static  void             roffhash_init(void);
164 static  void             roffnode_cleanscope(struct roff *);
165 static  void             roffnode_pop(struct roff *);
166 static  void             roffnode_push(struct roff *, enum rofft,
167                                 const char *, int, int);
168 static  enum rofferr     roff_block(ROFF_ARGS);
169 static  enum rofferr     roff_block_text(ROFF_ARGS);
170 static  enum rofferr     roff_block_sub(ROFF_ARGS);
171 static  enum rofferr     roff_cblock(ROFF_ARGS);
172 static  enum rofferr     roff_ccond(ROFF_ARGS);
173 static  enum rofferr     roff_cond(ROFF_ARGS);
174 static  enum rofferr     roff_cond_text(ROFF_ARGS);
175 static  enum rofferr     roff_cond_sub(ROFF_ARGS);
176 static  enum rofferr     roff_ds(ROFF_ARGS);
177 static  enum roffrule    roff_evalcond(const char *, int *);
178 static  void             roff_free1(struct roff *);
179 static  void             roff_freestr(struct roffkv *);
180 static  char            *roff_getname(struct roff *, char **, int, int);
181 static  const char      *roff_getstrn(const struct roff *,
182                                 const char *, size_t);
183 static  enum rofferr     roff_line_ignore(ROFF_ARGS);
184 static  enum rofferr     roff_nr(ROFF_ARGS);
185 static  void             roff_openeqn(struct roff *, const char *,
186                                 int, int, const char *);
187 static  enum rofft       roff_parse(struct roff *, const char *, int *);
188 static  enum rofferr     roff_parsetext(char *);
189 static  enum rofferr     roff_res(struct roff *,
190                                 char **, size_t *, int, int);
191 static  enum rofferr     roff_rm(ROFF_ARGS);
192 static  void             roff_setstr(struct roff *,
193                                 const char *, const char *, int);
```



```
194 static  void             roff_setstrn(struct roffkv **, const char *,
195                                 size_t, const char *, size_t, int);
196 static  enum rofferr     roff_so(ROFF_ARGS);
197 static  enum rofferr     roff_tr(ROFF_ARGS);
198 static  enum rofferr     roff_TE(ROFF_ARGS);
199 static  enum rofferr     roff_TS(ROFF_ARGS);
200 static  enum rofferr     roff_EQ(ROFF_ARGS);
201 static  enum rofferr     roff_EN(ROFF_ARGS);
202 static  enum rofferr     roff_T_(ROFF_ARGS);
203 static  enum rofferr     roff_userdef(ROFF_ARGS);

205 /* See roffhash_find() */

207 #define ASCII_HI         126
208 #define ASCII_LO         33
209 #define HASHWIDTH       (ASCII_HI - ASCII_LO + 1)

211 static  struct roffmac  *hash[HASHWIDTH];

213 static  struct roffmac   roffs[ROFF_MAX] = {
214         { "ad", roff_line_ignore, NULL, NULL, 0, NULL },
215         { "am", roff_block, roff_block_text, roff_block_sub, 0, NULL },
216         { "ami", roff_block, roff_block_text, roff_block_sub, 0, NULL },
217         { "am1", roff_block, roff_block_text, roff_block_sub, 0, NULL },
218         { "de", roff_block, roff_block_text, roff_block_sub, 0, NULL },
219         { "dei", roff_block, roff_block_text, roff_block_sub, 0, NULL },
220         { "de1", roff_block, roff_block_text, roff_block_sub, 0, NULL },
221         { "ds", roff_ds, NULL, NULL, 0, NULL },
222         { "el", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
223         { "hy", roff_line_ignore, NULL, NULL, 0, NULL },
224         { "ie", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
225         { "if", roff_cond, roff_cond_text, roff_cond_sub, ROFFMAC_STRUCT, NULL }
226         { "ig", roff_block, roff_block_text, roff_block_sub, 0, NULL },
227         { "it", roff_line_ignore, NULL, NULL, 0, NULL },
228         { "ne", roff_line_ignore, NULL, NULL, 0, NULL },
229         { "nh", roff_line_ignore, NULL, NULL, 0, NULL },
230         { "nr", roff_nr, NULL, NULL, 0, NULL },
231         { "ns", roff_line_ignore, NULL, NULL, 0, NULL },
232         { "ps", roff_line_ignore, NULL, NULL, 0, NULL },
233         { "rm", roff_rm, NULL, NULL, 0, NULL },
234         { "so", roff_so, NULL, NULL, 0, NULL },
235         { "ta", roff_line_ignore, NULL, NULL, 0, NULL },
236         { "tr", roff_tr, NULL, NULL, 0, NULL },
237         { "TS", roff_TS, NULL, NULL, 0, NULL },
238         { "TE", roff_TE, NULL, NULL, 0, NULL },
239         { "T&", roff_T_, NULL, NULL, 0, NULL },
240         { "EQ", roff_EQ, NULL, NULL, 0, NULL },
241         { "EN", roff_EN, NULL, NULL, 0, NULL },
242         { ".", roff_cblock, NULL, NULL, 0, NULL },
243         { "\\}", roff_ccond, NULL, NULL, 0, NULL },
244         { NULL, roff_userdef, NULL, NULL, 0, NULL },
245 };

247 /* Array of injected predefined strings. */
248 #define PREDEFS_MAX      38
249 static  const struct predef predefs[PREDEFS_MAX] = {
250 #include "predefs.in"
251 };

253 /* See roffhash_find() */
254 #define ROFF_HASH(p)    (p[0] - ASCII_LO)

256 static void
257 roffhash_init(void)
258 {
259         struct roffmac   *n;
```

```
260         int               buc, i;

262         for (i = 0; i < (int)ROFF_USERDEF; i++) {
263                 assert(roffs[i].name[0] >= ASCII_LO);
264                 assert(roffs[i].name[0] <= ASCII_HI);

266                 buc = ROFF_HASH(roffs[i].name);

268                 if (NULL != (n = hash[buc])) {
269                         for ( ; n->next; n = n->next)
270                                 /* Do nothing. */ ;
271                         n->next = &roffs[i];
272                 } else
273                         hash[buc] = &roffs[i];
274         }
275 }

277 /*
278  * Look up a roff token by its name.  Returns ROFF_MAX if no macro by
279  * the nil-terminated string name could be found.
280  */
281 static enum rofft
282 roffhash_find(const char *p, size_t s)
283 {
284         int              buc;
285         struct roffmac  *n;

287         /*
288          * libroff has an extremely simple hashtable, for the time
289          * being, which simply keys on the first character, which must
290          * be printable, then walks a chain.  It works well enough until
291          * optimised.
292          */

294         if (p[0] < ASCII_LO || p[0] > ASCII_HI)
295                 return(ROFF_MAX);

297         buc = ROFF_HASH(p);

299         if (NULL == (n = hash[buc]))
300                 return(ROFF_MAX);
301         for ( ; n; n = n->next)
302                 if (0 == strncmp(n->name, p, s) && '\0' == n->name[(int)s])
303                         return((enum rofft)(n - roffs));

305         return(ROFF_MAX);
306 }


309 /*
310  * Pop the current node off of the stack of roff instructions currently
311  * pending.
312  */
313 static void
314 roffnode_pop(struct roff *r)
315 {
316         struct roffnode *p;

318         assert(r->last);
319         p = r->last;

321         r->last = r->last->parent;
322         free(p->name);
323         free(p->end);
324         free(p);
325 }
```



```
328 /*
329  * Push a roff node onto the instruction stack.  This must later be
330  * removed with roffnode_pop().
331  */
332 static void
333 roffnode_push(struct roff *r, enum rofft tok, const char *name,
334                 int line, int col)
335 {
336         struct roffnode *p;

338         p = mandoc_calloc(1, sizeof(struct roffnode));
339         p->tok = tok;
340         if (name)
341                 p->name = mandoc_strdup(name);
342         p->parent = r->last;
343         p->line = line;
344         p->col = col;
345         p->rule = p->parent ? p->parent->rule : ROFFRULE_DENY;

347         r->last = p;
348 }


351 static void
352 roff_free1(struct roff *r)
353 {
354         struct tbl_node *t;
355         struct eqn_node *e;
356         int              i;

358         while (NULL != (t = r->first_tbl)) {
359                 r->first_tbl = t->next;
360                 tbl_free(t);
361         }

363         r->first_tbl = r->last_tbl = r->tbl = NULL;

365         while (NULL != (e = r->first_eqn)) {
366                 r->first_eqn = e->next;
367                 eqn_free(e);
368         }

370         r->first_eqn = r->last_eqn = r->eqn = NULL;

372         while (r->last)
373                 roffnode_pop(r);

375         roff_freestr(r->strtab);
376         roff_freestr(r->xmbtab);

378         r->strtab = r->xmbtab = NULL;

380         if (r->xtab)
381                 for (i = 0; i < 128; i++)
382                         free(r->xtab[i].p);

384         free(r->xtab);
385         r->xtab = NULL;
386 }

388 void
389 roff_reset(struct roff *r)
390 {
391         int              i;
```

```
393         roff_free1(r);
395         memset(&r->regs, 0, sizeof(struct reg) * REG__MAX);
397         for (i = 0; i < PREDEFS_MAX; i++)
398                 roff_setstr(r, predefs[i].name, predefs[i].str, 0);
399 }


402 void
403 roff_free(struct roff *r)
404 {
406         roff_free1(r);
407         free(r);
408 }


411 struct roff *
412 roff_alloc(struct mparse *parse)
413 {
414         struct roff     *r;
415         int              i;
417         r = mandoc_calloc(1, sizeof(struct roff));
418         r->parse = parse;
419         r->rstackpos = -1;
420
421         roffhash_init();
423         for (i = 0; i < PREDEFS_MAX; i++)
424                 roff_setstr(r, predefs[i].name, predefs[i].str, 0);
426         return(r);
427 }
429 /*
430  * Pre-filter each and every line for reserved words (one beginning with
431  * '\*', e.g., '\*(ab').  These must be handled before the actual line
432  * is processed.
433  * This also checks the syntax of regular escapes.
434  */
435 static enum rofferr
436 roff_res(struct roff *r, char **bufp, size_t *szp, int ln, int pos)
437 {
438         enum mandoc_esc  esc;
439         const char      *stesc; /* start of an escape sequence ('\\') */
440         const char      *stnam; /* start of the name, after "[(*" */
441         const char      *cp;    /* end of the name, e.g. before ']' */
442         const char      *res;   /* the string to be substituted */
443         int              i, maxl, expand_count;
444         size_t           nsz;
445         char            *n;
447         expand_count = 0;
449 again:
450         cp = *bufp + pos;
451         while (NULL != (cp = strchr(cp, '\\'))) {
452                 stesc = cp++;
454                 /*
455                  * The second character must be an asterisk.
456                  * If it isn't, skip it anyway:  It is escaped,
457                  * so it can't start another escape sequence.
```



```
458                  */
460                 if ('\0' == *cp)
461                         return(ROFF_CONT);
463                 if ('*' != *cp) {
464                         res = cp;
465                         esc = mandoc_escape(&cp, NULL, NULL);
466                         if (ESCAPE_ERROR != esc)
467                                 continue;
468                         cp = res;
469                         mandoc_msg
470                                 (MANDOCERR_BADESCAPE, r->parse,
471                                  ln, (int)(stesc - *bufp), NULL);
472                         return(ROFF_CONT);
473                 }
475                 cp++;
477                 /*
478                  * The third character decides the length
479                  * of the name of the string.
480                  * Save a pointer to the name.
481                  */
483                 switch (*cp) {
484                 case ('\0'):
485                         return(ROFF_CONT);
486                 case ('('):
487                         cp++;
488                         maxl = 2;
489                         break;
490                 case ('['):
491                         cp++;
492                         maxl = 0;
493                         break;
494                 default:
495                         maxl = 1;
496                         break;
497                 }
498                 stnam = cp;
500                 /* Advance to the end of the name. */
502                 for (i = 0; 0 == maxl || i < maxl; i++, cp++) {
503                         if ('\0' == *cp) {
504                                 mandoc_msg
505                                         (MANDOCERR_BADESCAPE,
506                                          r->parse, ln,
507                                          (int)(stesc - *bufp), NULL);
508                                 return(ROFF_CONT);
509                         }
510                         if (0 == maxl && ']' == *cp)
511                                 break;
512                 }
514                 /*
515                  * Retrieve the replacement string; if it is
516                  * undefined, resume searching for escapes.
517                  */
519                 res = roff_getstrn(r, stnam, (size_t)i);
521                 if (NULL == res) {
522                         mandoc_msg
523                                 (MANDOCERR_BADESCAPE, r->parse,
```

```
524                                 ln, (int)(stesc - *bufp), NULL);
525                         res = "";
526                 }
527
528                 /* Replace the escape sequence by the string. */
529
530                 pos = stesc - *bufp;
531
532                 nsz = *szp + strlen(res) + 1;
533                 n = mandoc_malloc(nsz);
534
535                 strlcpy(n, *bufp, (size_t)(stesc - *bufp + 1));
536                 strlcat(n, res, nsz);
537                 strlcat(n, cp + (maxl ? 0 : 1), nsz);
538
539                 free(*bufp);
540
541                 *bufp = n;
542                 *szp = nsz;
543
544                 if (EXPAND_LIMIT >= ++expand_count)
545                         goto again;
546
547                 /* Just leave the string unexpanded. */
548                 mandoc_msg(MANDOCERR_ROFFLOOP, r->parse, ln, pos, NULL);
549                 return(ROFF_IGN);
550         }
551         return(ROFF_CONT);
552 }
553
554 /*
555  * Process text streams: convert all breakable hyphens into ASCII_HYPH.
556  */
557 static enum rofferr
558 roff_parsetext(char *p)
559 {
560         size_t           sz;
561         const char      *start;
562         enum mandoc_esc  esc;
563
564         start = p;
565
566         while ('\0' != *p) {
567                 sz = strcspn(p, "-\\");
568                 p += sz;
569
570                 if ('\0' == *p)
571                         break;
572
573                 if ('\\' == *p) {
574                         /* Skip over escapes. */
575                         p++;
576                         esc = mandoc_escape
577                                 ((const char **)&p, NULL, NULL);
578                         if (ESCAPE_ERROR == esc)
579                                 break;
580                         continue;
581                 } else if (p == start) {
582                         p++;
583                         continue;
584                 }
585
586                 if (isalpha((unsigned char)p[-1]) &&
587                     isalpha((unsigned char)p[1]))
588                         *p = ASCII_HYPH;
589                 p++;
```



```
590         }
591
592         return(ROFF_CONT);
593 }
594
595 enum rofferr
596 roff_parseln(struct roff *r, int ln, char **bufp,
597                 size_t *szp, int pos, int *offs)
598 {
599         enum rofft       t;
600         enum rofferr     e;
601         int              ppos, ctl;
602
603         /*
604          * Run the reserved-word filter only if we have some reserved
605          * words to fill in.
606          */
607
608         e = roff_res(r, bufp, szp, ln, pos);
609         if (ROFF_IGN == e)
610                 return(e);
611         assert(ROFF_CONT == e);
612
613         ppos = pos;
614         ctl = mandoc_getcontrol(*bufp, &pos);
615
616         /*
617          * First, if a scope is open and we're not a macro, pass the
618          * text through the macro's filter.  If a scope isn't open and
619          * we're not a macro, just let it through.
620          * Finally, if there's an equation scope open, divert it into it
621          * no matter our state.
622          */
623
624         if (r->last && ! ctl) {
625                 t = r->last->tok;
626                 assert(roffs[t].text);
627                 e = (*roffs[t].text)
628                         (r, t, bufp, szp, ln, pos, pos, offs);
629                 assert(ROFF_IGN == e || ROFF_CONT == e);
630                 if (ROFF_CONT != e)
631                         return(e);
632                 if (r->eqn)
633                         return(eqn_read(&r->eqn, ln, *bufp, pos, offs));
634                 if (r->tbl)
635                         return(tbl_read(r->tbl, ln, *bufp, pos));
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));
645
646         /*
647          * If a scope is open, go to the child handler for that macro,
648          * as it may want to preprocess before doing anything with it.
649          * Don't do so if an equation is open.
650          */
651
652         if (r->last) {
653                 t = r->last->tok;
654                 assert(roffs[t].sub);
655                 return((*roffs[t].sub)
```

```
656                                 (r, t, bufp, szp,
657                                  ln, ppos, pos, offs));
658         }
659
660         /*
661          * Lastly, as we've no scope open, try to look up and execute
662          * the new macro.  If no macro is found, simply return and let
663          * the compilers handle it.
664          */
665
666         if (ROFF_MAX == (t = roff_parse(r, *bufp, &pos)))
667                 return(ROFF_CONT);
668
669         assert(roffs[t].proc);
670         return((*roffs[t].proc)
671                         (r, t, bufp, szp,
672                          ln, ppos, pos, offs));
673 }
674
675
676 void
677 roff_endparse(struct roff *r)
678 {
679
680         if (r->last)
681                 mandoc_msg(MANDOCERR_SCOPEEXIT, r->parse,
682                                 r->last->line, r->last->col, NULL);
683
684         if (r->eqn) {
685                 mandoc_msg(MANDOCERR_SCOPEEXIT, r->parse,
686                                 r->eqn->eqn.ln, r->eqn->eqn.pos, NULL);
687                 eqn_end(&r->eqn);
688         }
689
690         if (r->tbl) {
691                 mandoc_msg(MANDOCERR_SCOPEEXIT, r->parse,
692                                 r->tbl->line, r->tbl->pos, NULL);
693                 tbl_end(&r->tbl);
694         }
695 }
696
697 /*
698  * Parse a roff node's type from the input buffer.  This must be in the
699  * form of ".foo xxx" in the usual way.
700  */
701 static enum rofft
702 roff_parse(struct roff *r, const char *buf, int *pos)
703 {
704         const char      *mac;
705         size_t           maclen;
706         enum rofft       t;
707
708         if ('\0' == buf[*pos] || '"' == buf[*pos] ||
709                         '\t' == buf[*pos] || ' ' == buf[*pos])
710                 return(ROFF_MAX);
711
712         /*
713          * We stop the macro parse at an escape, tab, space, or nil.
714          * However, '\}' is also a valid macro, so make sure we don't
715          * clobber it by seeing the '\' as the end of token.
716          */
717
718         mac = buf + *pos;
719         maclen = strcspn(mac + 1, " \\\t\0") + 1;
720
721         t = (r->current_string = roff_getstrn(r, mac, maclen))
```



```
722             ? ROFF_USERDEF : roffhash_find(mac, maclen);
723
724         *pos += (int)maclen;
725
726         while (buf[*pos] && ' ' == buf[*pos])
727                 (*pos)++;
728
729         return(t);
730 }
731
732 /* ARGSUSED */
733 static enum rofferr
734 roff_cblock(ROFF_ARGS)
735 {
736
737         /*
738          * A block-close '..' should only be invoked as a child of an
739          * ignore macro, otherwise raise a warning and just ignore it.
740          */
741
742         if (NULL == r->last) {
743                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
744                 return(ROFF_IGN);
745         }
746
747         switch (r->last->tok) {
748         case (ROFF_am):
749                 /* FALLTHROUGH */
750         case (ROFF_ami):
751                 /* FALLTHROUGH */
752         case (ROFF_am1):
753                 /* FALLTHROUGH */
754         case (ROFF_de):
755                 /* ROFF_de1 is remapped to ROFF_de in roff_block(). */
756                 /* FALLTHROUGH */
757         case (ROFF_dei):
758                 /* FALLTHROUGH */
759         case (ROFF_ig):
760                 break;
761         default:
762                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
763                 return(ROFF_IGN);
764         }
765
766         if ((*bufp)[pos])
767                 mandoc_msg(MANDOCERR_ARGSLOST, r->parse, ln, pos, NULL);
768
769         roffnode_pop(r);
770         roffnode_cleanscope(r);
771         return(ROFF_IGN);
772
773 }
774
775
776 static void
777 roffnode_cleanscope(struct roff *r)
778 {
779
780         while (r->last) {
781                 if (--r->last->endspan < 0)
782                         break;
783                 roffnode_pop(r);
784         }
785 }
```

```
788 /* ARGSUSED */
789 static enum rofferr
790 roff_ccond(ROFF_ARGS)
791 {
793         if (NULL == r->last) {
794                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
795                 return(ROFF_IGN);
796         }
798         switch (r->last->tok) {
799         case (ROFF_el):
800                 /* FALLTHROUGH */
801         case (ROFF_ie):
802                 /* FALLTHROUGH */
803         case (ROFF_if):
804                 break;
805         default:
806                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
807                 return(ROFF_IGN);
808         }
810         if (r->last->endspan > -1) {
811                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
812                 return(ROFF_IGN);
813         }
815         if ((*bufp)[pos])
816                 mandoc_msg(MANDOCERR_ARGSLOST, r->parse, ln, pos, NULL);
818         roffnode_pop(r);
819         roffnode_cleanscope(r);
820         return(ROFF_IGN);
821 }
824 /* ARGSUSED */
825 static enum rofferr
826 roff_block(ROFF_ARGS)
827 {
828         int             sv;
829         size_t          sz;
830         char            *name;
832         name = NULL;
834         if (ROFF_ig != tok) {
835                 if ('\0' == (*bufp)[pos]) {
836                         mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
837                         return(ROFF_IGN);
838                 }
840                 /*
841                  * Re-write `de1', since we don't really care about
842                  * groff's strange compatibility mode, into `de'.
843                  */
845                 if (ROFF_de1 == tok)
846                         tok = ROFF_de;
847                 if (ROFF_de == tok)
848                         name = *bufp + pos;
849                 else
850                         mandoc_msg(MANDOCERR_REQUEST, r->parse, ln, ppos,
851                             roffs[tok].name);
853                 while ((*bufp)[pos] && ! isspace((unsigned char)(*bufp)[pos]))
```



```
854                         pos++;
856                 while (isspace((unsigned char)(*bufp)[pos]))
857                         (*bufp)[pos++] = '\0';
858         }
860         roffnode_push(r, tok, name, ln, ppos);
862         /*
863          * At the beginning of a `de' macro, clear the existing string
864          * with the same name, if there is one.  New content will be
865          * added from roff_block_text() in multiline mode.
866          */
868         if (ROFF_de == tok)
869                 roff_setstr(r, name, "", 0);
871         if ('\0' == (*bufp)[pos])
872                 return(ROFF_IGN);
874         /* If present, process the custom end-of-line marker. */
876         sv = pos;
877         while ((*bufp)[pos] && ! isspace((unsigned char)(*bufp)[pos]))
878                 pos++;
880         /*
881          * Note: groff does NOT like escape characters in the input.
882          * Instead of detecting this, we're just going to let it fly and
883          * to hell with it.
884          */
886         assert(pos > sv);
887         sz = (size_t)(pos - sv);
889         if (1 == sz && '.' == (*bufp)[sv])
890                 return(ROFF_IGN);
892         r->last->end = mandoc_malloc(sz + 1);
894         memcpy(r->last->end, *bufp + sv, sz);
895         r->last->end[(int)sz] = '\0';
897         if ((*bufp)[pos])
898                 mandoc_msg(MANDOCERR_ARGSLOST, r->parse, ln, pos, NULL);
900         return(ROFF_IGN);
901 }
904 /* ARGSUSED */
905 static enum rofferr
906 roff_block_sub(ROFF_ARGS)
907 {
908         enum rofft      t;
909         int             i, j;
911         /*
912          * First check whether a custom macro exists at this level.  If
913          * it does, then check against it.  This is some of groff's
914          * stranger behaviours.  If we encountered a custom end-scope
915          * tag and that tag also happens to be a "real" macro, then we
916          * need to try interpreting it again as a real macro.  If it's
917          * not, then return ignore.  Else continue.
918          */
```

```
920         if (r->last->end) {
921                 for (i = pos, j = 0; r->last->end[j]; j++, i++)
922                         if ((*bufp)[i] != r->last->end[j])
923                                 break;
924
925                 if ('\0' == r->last->end[j] &&
926                                 ('\0' == (*bufp)[i] ||
927                                  ' ' == (*bufp)[i] ||
928                                  '\t' == (*bufp)[i])) {
929                         roffnode_pop(r);
930                         roffnode_cleanscope(r);
931
932                         while (' ' == (*bufp)[i] || '\t' == (*bufp)[i])
933                                 i++;
934
935                         pos = i;
936                         if (ROFF_MAX != roff_parse(r, *bufp, &pos))
937                                 return(ROFF_RERUN);
938                         return(ROFF_IGN);
939                 }
940         }
941
942         /*
943          * If we have no custom end-query or lookup failed, then try
944          * pulling it out of the hashtable.
945          */
946
947         t = roff_parse(r, *bufp, &pos);
948
949         /*
950          * Macros other than block-end are only significant
951          * in 'de' blocks; elsewhere, simply throw them away.
952          */
953         if (ROFF_cblock != t) {
954                 if (ROFF_de == tok)
955                         roff_setstr(r, r->last->name, *bufp + ppos, 1);
956                 return(ROFF_IGN);
957         }
958
959         assert(roffs[t].proc);
960         return((*roffs[t].proc)(r, t, bufp, szp,
961                                 ln, ppos, pos, offs));
962 }
963
964
965 /* ARGSUSED */
966 static enum rofferr
967 roff_block_text(ROFF_ARGS)
968 {
969
970         if (ROFF_de == tok)
971                 roff_setstr(r, r->last->name, *bufp + pos, 1);
972
973         return(ROFF_IGN);
974 }
975
976
977 /* ARGSUSED */
978 static enum rofferr
979 roff_cond_sub(ROFF_ARGS)
980 {
981         enum rofft       t;
982         enum roffrule    rr;
983         char            *ep;
984
985         rr = r->last->rule;
```



```
986         roffnode_cleanscope(r);
987
988         /*
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.
993          */
994
995         if (ROFF_MAX == (t = roff_parse(r, *bufp, &pos))) {
996                 ep = &(*bufp)[pos];
997                 for ( ; NULL != (ep = strchr(ep, '\\')); ep++) {
998                         ep++;
999                         if ('}' != *ep)
1000                                continue;
1001
1002                        /*
1003                         * Make the \} go away.
1004                         * This is a little haphazard, as it's not quite
1005                         * clear how nroff does this.
1006                         * If we're at the end of line, then just chop
1007                         * off the \} and resize the buffer.
1008                         * If we aren't, then conver it to spaces.
1009                         */
1010
1011                        if ('\0' == *(ep + 1)) {
1012                                *--ep = '\0';
1013                                *szp -= 2;
1014                        } else
1015                                *(ep - 1) = *ep = ' ';
1016
1017                        roff_ccond(r, ROFF_ccond, bufp, szp,
1018                                        ln, pos, pos + 2, offs);
1019                        break;
1020                }
1021                return(ROFFRULE_DENY == rr ? ROFF_IGN : ROFF_CONT);
1022        }
1023
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         */
1029
1030        if (ROFFRULE_DENY == rr)
1031                if ( ! (ROFFMAC_STRUCT & roffs[t].flags))
1032                        if (ROFF_ccond != t)
1033                                return(ROFF_IGN);
1034
1035        assert(roffs[t].proc);
1036        return((*roffs[t].proc)(r, t, bufp, szp,
1037                                ln, ppos, pos, offs));
1038 }
1039
1040 /* ARGSUSED */
1041 static enum rofferr
1042 roff_cond_text(ROFF_ARGS)
1043 {
1044         char            *ep;
1045         enum roffrule    rr;
1046
1047         rr = r->last->rule;
1048         roffnode_cleanscope(r);
1049
1050         ep = &(*bufp)[pos];
1051         for ( ; NULL != (ep = strchr(ep, '\\')); ep++) {
```

```
1052                ep++;
1053                if ('}' != *ep)
1054                        continue;
1055                *ep = '&';
1056                roff_ccond(r, ROFF_ccond, bufp, szp,
1057                                ln, pos, pos + 2, offs);
1058        }
1059        return(ROFFRULE_DENY == rr ? ROFF_IGN : ROFF_CONT);
1060 }

1062 static enum roffrule
1063 roff_evalcond(const char *v, int *pos)
1064 {

1066        switch (v[*pos]) {
1067        case ('n'):
1068                (*pos)++;
1069                return(ROFFRULE_ALLOW);
1070        case ('e'):
1071                /* FALLTHROUGH */
1072        case ('o'):
1073                /* FALLTHROUGH */
1074        case ('t'):
1075                (*pos)++;
1076                return(ROFFRULE_DENY);
1077        default:
1078                break;
1079        }

1081        while (v[*pos] && ' ' != v[*pos])
1082                (*pos)++;
1083        return(ROFFRULE_DENY);
1084 }

1086 /* ARGSUSED */
1087 static enum rofferr
1088 roff_line_ignore(ROFF_ARGS)
1089 {

1091        if (ROFF_it == tok)
1092                mandoc_msg(MANDOCERR_REQUEST, r->parse, ln, ppos, "it");

1094        return(ROFF_IGN);
1095 }

1097 /* ARGSUSED */
1098 static enum rofferr
1099 roff_cond(ROFF_ARGS)
1100 {
1101        int              sv;
1102        enum roffrule    rule;

1104        /*
1105         * An '.el' has no conditional body: it will consume the value
1106         * of the current rstack entry set in prior 'ie' calls or
1107         * defaults to DENY.
1108         *
1109         * If we're not an 'el', however, then evaluate the conditional.
1110         */

1112        rule = ROFF_el == tok ?
1113                (r->rstackpos < 0 ?
1114                 ROFFRULE_DENY : r->rstack[r->rstackpos--]) :
1115                roff_evalcond(*bufp, &pos);

1117        sv = pos;
```



```
1118        while (' ' == (*bufp)[pos])
1119                pos++;

1121        /*
1122         * Roff is weird.  If we have just white-space after the
1123         * conditional, it's considered the BODY and we exit without
1124         * really doing anything.  Warn about this.  It's probably
1125         * wrong.
1126         */

1128        if ('\0' == (*bufp)[pos] && sv != pos) {
1129                mandoc_msg(MANDOCERR_NOARGS, r->parse, ln, ppos, NULL);
1130                return(ROFF_IGN);
1131        }

1133        roffnode_push(r, tok, NULL, ln, ppos);

1135        r->last->rule = rule;

1137        /*
1138         * An if-else will put the NEGATION of the current evaluated
1139         * conditional into the stack of rules.
1140         */

1142        if (ROFF_ie == tok) {
1143                if (r->rstackpos == RSTACK_MAX - 1) {
1144                        mandoc_msg(MANDOCERR_MEM,
1145                                        r->parse, ln, ppos, NULL);
1146                        return(ROFF_ERR);
1147                }
1148                r->rstack[++r->rstackpos] =
1149                        ROFFRULE_DENY == r->last->rule ?
1150                        ROFFRULE_ALLOW : ROFFRULE_DENY;
1151        }

1153        /* If the parent has false as its rule, then so do we. */

1155        if (r->last->parent && ROFFRULE_DENY == r->last->parent->rule)
1156                r->last->rule = ROFFRULE_DENY;

1158        /*
1159         * Determine scope.  If we're invoked with "\{" trailing the
1160         * conditional, then we're in a multiline scope.  Else our scope
1161         * expires on the next line.
1162         */

1164        r->last->endspan = 1;

1166        if ('\\' == (*bufp)[pos] && '{' == (*bufp)[pos + 1]) {
1167                r->last->endspan = -1;
1168                pos += 2;
1169        }

1171        /*
1172         * If there are no arguments on the line, the next-line scope is
1173         * assumed.
1174         */

1176        if ('\0' == (*bufp)[pos])
1177                return(ROFF_IGN);

1179        /* Otherwise re-run the roff parser after recalculating. */

1181        *offs = pos;
1182        return(ROFF_RERUN);
1183 }
```

```
1186 /* ARGSUSED */
1187 static enum rofferr
1188 roff_ds(ROFF_ARGS)
1189 {
1190         char            *name, *string;

1192         /*
1193          * A symbol is named by the first word following the macro
1194          * invocation up to a space.  Its value is anything after the
1195          * name's trailing whitespace and optional double-quote.  Thus,
1196          *
1197          *  [.ds foo "bar  "     ]
1198          *
1199          * will have 'bar  "     ' as its value.
1200          */

1202         string = *bufp + pos;
1203         name = roff_getname(r, &string, ln, pos);
1204         if ('\0' == *name)
1205                 return(ROFF_IGN);

1207         /* Read past initial double-quote. */
1208         if ('"' == *string)
1209                 string++;

1211         /* The rest is the value. */
1212         roff_setstr(r, name, string, 0);
1213         return(ROFF_IGN);
1214 }

1216 int
1217 roff_regisset(const struct roff *r, enum regs reg)
1218 {

1220         return(r->regs[(int)reg].set);
1221 }

1223 unsigned int
1224 roff_regget(const struct roff *r, enum regs reg)
1225 {

1227         return(r->regs[(int)reg].u);
1228 }

1230 void
1231 roff_regunset(struct roff *r, enum regs reg)
1232 {

1234         r->regs[(int)reg].set = 0;
1235 }

1237 /* ARGSUSED */
1238 static enum rofferr
1239 roff_nr(ROFF_ARGS)
1240 {
1241         const char      *key;
1242         char            *val;
1243         int              iv;

1245         val = *bufp + pos;
1246         key = roff_getname(r, &val, ln, pos);

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         }

1256         return(ROFF_IGN);
1257 }

1259 /* ARGSUSED */
1260 static enum rofferr
1261 roff_rm(ROFF_ARGS)
1262 {
1263         const char       *name;
1264         char             *cp;

1266         cp = *bufp + pos;
1267         while ('\0' != *cp) {
1268                 name = roff_getname(r, &cp, ln, (int)(cp - *bufp));
1269                 if ('\0' != *name)
1270                         roff_setstr(r, name, NULL, 0);
1271         }
1272         return(ROFF_IGN);
1273 }

1275 /* ARGSUSED */
1276 static enum rofferr
1277 roff_TE(ROFF_ARGS)
1278 {

1280         if (NULL == r->tbl)
1281                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
1282         else
1283                 tbl_end(&r->tbl);

1285         return(ROFF_IGN);
1286 }

1288 /* ARGSUSED */
1289 static enum rofferr
1290 roff_T_(ROFF_ARGS)
1291 {

1293         if (NULL == r->tbl)
1294                 mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
1295         else
1296                 tbl_restart(ppos, ln, r->tbl);

1298         return(ROFF_IGN);
1299 }

1301 #if 0
1302 static int
1303 roff_closeeqn(struct roff *r)
1304 {

1306         return(r->eqn && ROFF_EQN == eqn_end(&r->eqn) ? 1 : 0);
1307 }
1308 #endif

1310 static void
1311 roff_openeqn(struct roff *r, const char *name, int line,
1312                 int offs, const char *buf)
1313 {
1314         struct eqn_node *e;
1315         int              poff;
```

```
1317         assert(NULL == r->eqn);
1318         e = eqn_alloc(name, offs, line, r->parse);

1320         if (r->last_eqn)
1321                 r->last_eqn->next = e;
1322         else
1323                 r->first_eqn = r->last_eqn = e;

1325         r->eqn = r->last_eqn = e;

1327         if (buf) {
1328                 poff = 0;
1329                 eqn_read(&r->eqn, line, buf, offs, &poff);
1330         }
1331 }

1333 /* ARGSUSED */
1334 static enum rofferr
1335 roff_EQ(ROFF_ARGS)
1336 {

1338         roff_openeqn(r, *bufp + pos, ln, ppos, NULL);
1339         return(ROFF_IGN);
1340 }

1342 /* ARGSUSED */
1343 static enum rofferr
1344 roff_EN(ROFF_ARGS)
1345 {

1347         mandoc_msg(MANDOCERR_NOSCOPE, r->parse, ln, ppos, NULL);
1348         return(ROFF_IGN);
1349 }

1351 /* ARGSUSED */
1352 static enum rofferr
1353 roff_TS(ROFF_ARGS)
1354 {
1355         struct tbl_node *t;

1357         if (r->tbl) {
1358                 mandoc_msg(MANDOCERR_SCOPEBROKEN, r->parse, ln, ppos, NULL);
1359                 tbl_end(&r->tbl);
1360         }

1362         t = tbl_alloc(ppos, ln, r->parse);

1364         if (r->last_tbl)
1365                 r->last_tbl->next = t;
1366         else
1367                 r->first_tbl = r->last_tbl = t;

1369         r->tbl = r->last_tbl = t;
1370         return(ROFF_IGN);
1371 }

1373 /* ARGSUSED */
1374 static enum rofferr
1375 roff_tr(ROFF_ARGS)
1376 {
1377         const char      *p, *first, *second;
1378         size_t           fsz, ssz;
1379         enum mandoc_esc  esc;

1381         p = *bufp + pos;
```



```
1383         if ('\0' == *p) {
1384                 mandoc_msg(MANDOCERR_ARGCOUNT, r->parse, ln, ppos, NULL);
1385                 return(ROFF_IGN);
1386         }

1388         while ('\0' != *p) {
1389                 fsz = ssz = 1;

1391                 first = p++;
1392                 if ('\\' == *first) {
1393                         esc = mandoc_escape(&p, NULL, NULL);
1394                         if (ESCAPE_ERROR == esc) {
1395                                 mandoc_msg
1396                                         (MANDOCERR_BADESCAPE, r->parse,
1397                                          ln, (int)(p - *bufp), NULL);
1398                                 return(ROFF_IGN);
1399                         }
1400                         fsz = (size_t)(p - first);
1401                 }

1403                 second = p++;
1404                 if ('\\' == *second) {
1405                         esc = mandoc_escape(&p, NULL, NULL);
1406                         if (ESCAPE_ERROR == esc) {
1407                                 mandoc_msg
1408                                         (MANDOCERR_BADESCAPE, r->parse,
1409                                          ln, (int)(p - *bufp), NULL);
1410                                 return(ROFF_IGN);
1411                         }
1412                         ssz = (size_t)(p - second);
1413                 } else if ('\0' == *second) {
1414                         mandoc_msg(MANDOCERR_ARGCOUNT, r->parse,
1415                                         ln, (int)(p - *bufp), NULL);
1416                         second = " ";
1417                         p--;
1418                 }

1420                 if (fsz > 1) {
1421                         roff_setstrn(&r->xmbtab, first,
1422                                         fsz, second, ssz, 0);
1423                         continue;
1424                 }

1426                 if (NULL == r->xtab)
1427                         r->xtab = mandoc_calloc
1428                                 (128, sizeof(struct roffstr));

1430                 free(r->xtab[(int)*first].p);
1431                 r->xtab[(int)*first].p = mandoc_strndup(second, ssz);
1432                 r->xtab[(int)*first].sz = ssz;
1433         }

1435         return(ROFF_IGN);
1436 }

1438 /* ARGSUSED */
1439 static enum rofferr
1440 roff_so(ROFF_ARGS)
1441 {
1442         char *name;

1444         mandoc_msg(MANDOCERR_SO, r->parse, ln, ppos, NULL);

1446         /*
1447          * Handle 'so'.  Be EXTREMELY careful, as we shouldn't be
```

```
1448          * opening anything that's not in our cwd or anything beneath
1449          * it.  Thus, explicitly disallow traversing up the file-system
1450          * or using absolute paths.
1451          */

1453         name = *bufp + pos;
1454         if ('/' == *name || strstr(name, "../") || strstr(name, "/..")) {
1455                 mandoc_msg(MANDOCERR_SOPATH, r->parse, ln, pos, NULL);
1456                 return(ROFF_ERR);
1457         }

1459         *offs = pos;
1460         return(ROFF_SO);
1461 }

1463 /* ARGSUSED */
1464 static enum rofferr
1465 roff_userdef(ROFF_ARGS)
1466 {
1467         const char       *arg[9];
1468         char             *cp, *n1, *n2;
1469         int               i;

1471         /*
1472          * Collect pointers to macro argument strings
1473          * and null-terminate them.
1474          */
1475         cp = *bufp + pos;
1476         for (i = 0; i < 9; i++)
1477                 arg[i] = '\0' == *cp ? "" :
1478                     mandoc_getarg(r->parse, &cp, ln, &pos);

1480         /*
1481          * Expand macro arguments.
1482          */
1483         *szp = 0;
1484         n1 = cp = mandoc_strdup(r->current_string);
1485         while (NULL != (cp = strstr(cp, "\\$"))) {
1486                 i = cp[2] - '1';
1487                 if (0 > i || 8 < i) {
1488                         /* Not an argument invocation. */
1489                         cp += 2;
1490                         continue;
1491                 }

1493                 *szp = strlen(n1) - 3 + strlen(arg[i]) + 1;
1494                 n2 = mandoc_malloc(*szp);

1496                 strlcpy(n2, n1, (size_t)(cp - n1 + 1));
1497                 strlcat(n2, arg[i], *szp);
1498                 strlcat(n2, cp + 3, *szp);

1500                 cp = n2 + (cp - n1);
1501                 free(n1);
1502                 n1 = n2;
1503         }

1505         /*
1506          * Replace the macro invocation
1507          * by the expanded macro.
1508          */
1509         free(*bufp);
1510         *bufp = n1;
1511         if (0 == *szp)
1512                 *szp = strlen(*bufp) + 1;
```



```
1514         return(*szp > 1 && '\n' == (*bufp)[(int)*szp - 2] ?
1515            ROFF_REPARSE : ROFF_APPEND);
1516 }

1518 static char *
1519 roff_getname(struct roff *r, char **cpp, int ln, int pos)
1520 {
1521         char     *name, *cp;

1523         name = *cpp;
1524         if ('\0' == *name)
1525                 return(name);

1527         /* Read until end of name. */
1528         for (cp = name; '\0' != *cp && ' ' != *cp; cp++) {
1529                 if ('\\' != *cp)
1530                         continue;
1531                 cp++;
1532                 if ('\\' == *cp)
1533                         continue;
1534                 mandoc_msg(MANDOCERR_NAMESC, r->parse, ln, pos, NULL);
1535                 *cp = '\0';
1536                 name = cp;
1537         }

1539         /* Nil-terminate name. */
1540         if ('\0' != *cp)
1541                 *(cp++) = '\0';

1543         /* Read past spaces. */
1544         while (' ' == *cp)
1545                 cp++;

1547         *cpp = cp;
1548         return(name);
1549 }

1551 /*
1552  * Store *string into the user-defined string called *name.
1553  * In multiline mode, append to an existing entry and append '\n';
1554  * else replace the existing entry, if there is one.
1555  * To clear an existing entry, call with (*r, *name, NULL, 0).
1556  */
1557 static void
1558 roff_setstr(struct roff *r, const char *name, const char *string,
1559         int multiline)
1560 {

1562         roff_setstrn(&r->strtab, name, strlen(name), string,
1563                         string ? strlen(string) : 0, multiline);
1564 }

1566 static void
1567 roff_setstrn(struct roffkv **r, const char *name, size_t namesz,
1568                 const char *string, size_t stringsz, int multiline)
1569 {
1570         struct roffkv   *n;
1571         char            *c;
1572         int              i;
1573         size_t           oldch, newch;

1575         /* Search for an existing string with the same name. */
1576         n = *r;

1578         while (n && strcmp(name, n->key.p))
1579                 n = n->next;
```

```
1581         if (NULL == n) {
1582                 /* Create a new string table entry. */
1583                 n = mandoc_malloc(sizeof(struct roffkv));
1584                 n->key.p = mandoc_strndup(name, namesz);
1585                 n->key.sz = namesz;
1586                 n->val.p = NULL;
1587                 n->val.sz = 0;
1588                 n->next = *r;
1589                 *r = n;
1590         } else if (0 == multiline) {
1591                 /* In multiline mode, append; else replace. */
1592                 free(n->val.p);
1593                 n->val.p = NULL;
1594                 n->val.sz = 0;
1595         }

1597         if (NULL == string)
1598                 return;

1600         /*
1601          * One additional byte for the '\n' in multiline mode,
1602          * and one for the terminating '\0'.
1603          */
1604         newch = stringsz + (multiline ? 2u : 1u);

1606         if (NULL == n->val.p) {
1607                 n->val.p = mandoc_malloc(newch);
1608                 *n->val.p = '\0';
1609                 oldch = 0;
1610         } else {
1611                 oldch = n->val.sz;
1612                 n->val.p = mandoc_realloc(n->val.p, oldch + newch);
1613         }

1615         /* Skip existing content in the destination buffer. */
1616         c = n->val.p + (int)oldch;

1618         /* Append new content to the destination buffer. */
1619         i = 0;
1620         while (i < (int)stringsz) {
1621                 /*
1622                  * Rudimentary roff copy mode:
1623                  * Handle escaped backslashes.
1624                  */
1625                 if ('\\' == string[i] && '\\' == string[i + 1])
1626                         i++;
1627                 *c++ = string[i++];
1628         }

1630         /* Append terminating bytes. */
1631         if (multiline)
1632                 *c++ = '\n';

1634         *c = '\0';
1635         n->val.sz = (int)(c - n->val.p);
1636 }

1638 static const char *
1639 roff_getstrn(const struct roff *r, const char *name, size_t len)
1640 {
1641         const struct roffkv *n;

1643         for (n = r->strtab; n; n = n->next)
1644                 if (0 == strncmp(name, n->key.p, len) &&
1645                                 '\0' == n->key.p[(int)len])
```



```
1646                         return(n->val.p);

1648         return(NULL);
1649 }

1651 static void
1652 roff_freestr(struct roffkv *r)
1653 {
1654         struct roffkv    *n, *nn;

1656         for (n = r; n; n = nn) {
1657                 free(n->key.p);
1658                 free(n->val.p);
1659                 nn = n->next;
1660                 free(n);
1661         }
1662 }

1664 const struct tbl_span *
1665 roff_span(const struct roff *r)
1666 {
1667
1668         return(r->tbl ? tbl_span(r->tbl) : NULL);
1669 }

1671 const struct eqn *
1672 roff_eqn(const struct roff *r)
1673 {
1674
1675         return(r->last_eqn ? &r->last_eqn->eqn : NULL);
1676 }

1678 /*
1679  * Duplicate an input string, making the appropriate character
1680  * conversations (as stipulated by 'tr') along the way.
1681  * Returns a heap-allocated string with all the replacements made.
1682  */
1683 char *
1684 roff_strdup(const struct roff *r, const char *p)
1685 {
1686         const struct roffkv *cp;
1687         char                *res;
1688         const char      *pp;
1689         size_t           ssz, sz;
1690         enum mandoc_esc  esc;

1692         if (NULL == r->xmbtab && NULL == r->xtab)
1693                 return(mandoc_strdup(p));
1694         else if ('\0' == *p)
1695                 return(mandoc_strdup(""));

1697         /*
1698          * Step through each character looking for term matches
1699          * (remember that a 'tr' can be invoked with an escape, which is
1700          * a glyph but the escape is multi-character).
1701          * We only do this if the character hash has been initialised
1702          * and the string is >0 length.
1703          */

1705         res = NULL;
1706         ssz = 0;

1708         while ('\0' != *p) {
1709                 if ('\\' != *p && r->xtab && r->xtab[(int)*p].p) {
1710                         sz = r->xtab[(int)*p].sz;
1711                         res = mandoc_realloc(res, ssz + sz + 1);
```

```
1712                        memcpy(res + ssz, r->xtab[(int)*p].p, sz);
1713                        ssz += sz;
1714                        p++;
1715                        continue;
1716                } else if ('\\' != *p) {
1717                        res = mandoc_realloc(res, ssz + 2);
1718                        res[ssz++] = *p++;
1719                        continue;
1720                }

1722                /* Search for term matches. */
1723                for (cp = r->xmbtab; cp; cp = cp->next)
1724                        if (0 == strncmp(p, cp->key.p, cp->key.sz))
1725                                break;

1727                if (NULL != cp) {
1728                        /*
1729                         * A match has been found.
1730                         * Append the match to the array and move
1731                         * forward by its keysize.
1732                         */
1733                        res = mandoc_realloc
1734                                (res, ssz + cp->val.sz + 1);
1735                        memcpy(res + ssz, cp->val.p, cp->val.sz);
1736                        ssz += cp->val.sz;
1737                        p += (int)cp->key.sz;
1738                        continue;
1739                }

1741                /*
1742                 * Handle escapes carefully: we need to copy
1743                 * over just the escape itself, or else we might
1744                 * do replacements within the escape itself.
1745                 * Make sure to pass along the bogus string.
1746                 */
1747                pp = p++;
1748                esc = mandoc_escape(&p, NULL, NULL);
1749                if (ESCAPE_ERROR == esc) {
1750                        sz = strlen(pp);
1751                        res = mandoc_realloc(res, ssz + sz + 1);
1752                        memcpy(res + ssz, pp, sz);
1753                        break;
1754                }
1755                /*
1756                 * We bail out on bad escapes.
1757                 * No need to warn: we already did so when
1758                 * roff_res() was called.
1759                 */
1760                sz = (int)(p - pp);
1761                res = mandoc_realloc(res, ssz + sz + 1);
1762                memcpy(res + ssz, pp, sz);
1763                ssz += sz;
1764        }

1766        res[(int)ssz] = '\0';
1767        return(res);
1768 }
```

```
**********************************************************
    1152 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/st.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: st.c,v 1.9 2011/03/22 14:33:05 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 <stdlib.h>
#include <string.h>
#include <time.h>

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

#define LINE(x, y) \
        if (0 == strcmp(p, x)) return(y);

const char *
mdoc_a2st(const char *p)
{

#include "st.in"

        return(NULL);
}
```

```
**********************************************************
    4542 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/st.in
Initial import of man functionality.
**********************************************************
/*      $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.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("-p1387.2-95",     "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("-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")
LINE("-susv3",          "Version\\~3 of the Single UNIX Specification")
LINE("-svid4",          "System\\~V Interface Definition, Fourth Edition (\\(lqS
```

```
**********************************************************
    4049 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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 *p;

        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 *p)
{
        struct tbl_row  *rp;
        struct tbl_cell *cp;
        struct tbl_span *sp;
        struct tbl_dat  *dp;
        struct tbl_head *hp;

        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 = 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 = p->first_head)) {
                p->first_head = hp->next;
                free(hp);
        }

        free(p);
}

void
```

```
128 tbl_restart(int line, int pos, struct tbl_node *tbl)
129 {
130         if (TBL_PART_CDATA == tbl->part)
131                 mandoc_msg(MANDOCERR_TBLBLOCK, tbl->parse,
132                                 tbl->line, tbl->pos, NULL);
133
134         tbl->part = TBL_PART_LAYOUT;
135         tbl->line = line;
136         tbl->pos = pos;
137
138         if (NULL == tbl->first_span || NULL == tbl->first_span->first)
139                 mandoc_msg(MANDOCERR_TBLNODATA, tbl->parse,
140                                 tbl->line, tbl->pos, NULL);
141 }
142
143 const struct tbl_span *
144 tbl_span(struct tbl_node *tbl)
145 {
146         struct tbl_span  *span;
147
148         assert(tbl);
149         span = tbl->current_span ? tbl->current_span->next
150                                  : tbl->first_span;
151         if (span)
152                 tbl->current_span = span;
153         return(span);
154 }
155
156 void
157 tbl_end(struct tbl_node **tblp)
158 {
159         struct tbl_node *tbl;
160
161         tbl = *tblp;
162         *tblp = NULL;
163
164         if (NULL == tbl->first_span || NULL == tbl->first_span->first)
165                 mandoc_msg(MANDOCERR_TBLNODATA, tbl->parse,
166                                 tbl->line, tbl->pos, NULL);
167
168         if (tbl->last_span)
169                 tbl->last_span->flags |= TBL_SPAN_LAST;
170
171         if (TBL_PART_CDATA == tbl->part)
172                 mandoc_msg(MANDOCERR_TBLBLOCK, tbl->parse,
173                                 tbl->line, tbl->pos, NULL);
174 }
```

```
**********************************************************
    6446 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl_data.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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 and vertical lines to data formats, since
         * we want to match data with data layout cells in the header.
         */

        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(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, "="))
                dat->pos = TBL_DATA_DHORIZ;
        else if ( ! strcmp(dat->string, "\\_"))
                dat->pos = TBL_DATA_NHORIZ;
        else if ( ! strcmp(dat->string, "\\="))
                dat->pos = TBL_DATA_NDHORIZ;
        else
```

```
128                 dat->pos = TBL_DATA_DATA;

130         if (TBL_CELL_HORIZ == dat->layout->pos ||
131                         TBL_CELL_DHORIZ == dat->layout->pos ||
132                         TBL_CELL_DOWN == dat->layout->pos)
133                 if (TBL_DATA_DATA == dat->pos && '\0' != *dat->string)
134                         mandoc_msg(MANDOCERR_TBLIGNDATA,
135                                         tbl->parse, ln, sv, NULL);

137         return(1);
138 }

140 /* ARGSUSED */
141 int
142 tbl_cdata(struct tbl_node *tbl, int ln, const char *p)
143 {
144         struct tbl_dat  *dat;
145         size_t           sz;
146         int              pos;

148         pos = 0;

150         dat = tbl->last_span->last;

152         if (p[pos] == 'T' && p[pos + 1] == '}') {
153                 pos += 2;
154                 if (p[pos] == tbl->opts.tab) {
155                         tbl->part = TBL_PART_DATA;
156                         pos++;
157                         return(data(tbl, tbl->last_span, ln, p, &pos));
158                 } else if ('\0' == p[pos]) {
159                         tbl->part = TBL_PART_DATA;
160                         return(1);
161                 }

163                 /* Fallthrough: T} is part of a word. */
164         }

166         dat->pos = TBL_DATA_DATA;

168         if (dat->string) {
169                 sz = strlen(p) + strlen(dat->string) + 2;
170                 dat->string = mandoc_realloc(dat->string, sz);
171                 strlcat(dat->string, " ", sz);
172                 strlcat(dat->string, p, sz);
173         } else
174                 dat->string = mandoc_strdup(p);

176         if (TBL_CELL_DOWN == dat->layout->pos)
177                 mandoc_msg(MANDOCERR_TBLIGNDATA,
178                                 tbl->parse, ln, pos, NULL);

180         return(0);
181 }

183 static struct tbl_span *
184 newspan(struct tbl_node *tbl, int line, struct tbl_row *rp)
185 {
186         struct tbl_span *dp;

188         dp = mandoc_calloc(1, sizeof(struct tbl_span));
189         dp->line = line;
190         dp->tbl = &tbl->opts;
191         dp->layout = rp;
192         dp->head = tbl->first_head;
```



```
194         if (tbl->last_span) {
195                 tbl->last_span->next = dp;
196                 tbl->last_span = dp;
197         } else {
198                 tbl->last_span = tbl->first_span = dp;
199                 tbl->current_span = NULL;
200                 dp->flags |= TBL_SPAN_FIRST;
201         }

203         return(dp);
204 }

206 int
207 tbl_data(struct tbl_node *tbl, int ln, const char *p)
208 {
209         struct tbl_span *dp;
210         struct tbl_row  *rp;
211         int              pos;

213         pos = 0;

215         if ('\0' == p[pos]) {
216                 mandoc_msg(MANDOCERR_TBL, tbl->parse, ln, pos, NULL);
217                 return(0);
218         }

220         /*
221          * Choose a layout row: take the one following the last parsed
222          * span's.  If that doesn't exist, use the last parsed span's.
223          * If there's no last parsed span, use the first row.  Lastly,
224          * if the last span was a horizontal line, use the same layout
225          * (it doesn't "consume" the layout).
226          */

228         if (tbl->last_span) {
229                 assert(tbl->last_span->layout);
230                 if (tbl->last_span->pos == TBL_SPAN_DATA) {
231                         for (rp = tbl->last_span->layout->next;
232                                         rp && rp->first; rp = rp->next) {
233                                 switch (rp->first->pos) {
234                                 case (TBL_CELL_HORIZ):
235                                         dp = newspan(tbl, ln, rp);
236                                         dp->pos = TBL_SPAN_HORIZ;
237                                         continue;
238                                 case (TBL_CELL_DHORIZ):
239                                         dp = newspan(tbl, ln, rp);
240                                         dp->pos = TBL_SPAN_DHORIZ;
241                                         continue;
242                                 default:
243                                         break;
244                                 }
245                                 break;
246                         }
247                 } else
248                         rp = tbl->last_span->layout;

250                 if (NULL == rp)
251                         rp = tbl->last_span->layout;
252         } else
253                 rp = tbl->first_row;

255         assert(rp);

257         dp = newspan(tbl, ln, rp);

259         if ( ! strcmp(p, "_")) {
```

```
260                 dp->pos = TBL_SPAN_HORIZ;
261                 return(1);
262         } else if ( ! strcmp(p, "=")) {
263                 dp->pos = TBL_SPAN_DHORIZ;
264                 return(1);
265         }

267         dp->pos = TBL_SPAN_DATA;

269         /* This returns 0 when TBL_PART_CDATA is entered. */

271         while ('\0' != p[pos])
272                 if ( ! data(tbl, dp, ln, p, &pos))
273                         return(0);

275         return(1);
276 }
```

```
**********************************************************
    3315 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl_html.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: tbl_html.c,v 1.9 2011/09/18 14:14:15 schwarze Exp $ */
/*
 * Copyright (c) 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 <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mandoc.h"
#include "out.h"
#include "html.h"

static  void     html_tblopen(struct html *, const struct tbl_span *);
static  size_t   html_tbl_len(size_t, void *);
static  size_t   html_tbl_strlen(const char *, void *);

/* ARGSUSED */
static size_t
html_tbl_len(size_t sz, void *arg)
{

        return(sz);
}

/* ARGSUSED */
static size_t
html_tbl_strlen(const char *p, void *arg)
{

        return(strlen(p));
}

static void
html_tblopen(struct html *h, const struct tbl_span *sp)
{
        const struct tbl_head *hp;
        struct htmlpair  tag;
        struct roffsu    su;
        struct roffcol  *col;

        if (TBL_SPAN_FIRST & sp->flags) {
                h->tbl.len = html_tbl_len;
                h->tbl.slen = html_tbl_strlen;
                tblcalc(&h->tbl, sp);
```



```
        }

        assert(NULL == h->tblt);
        PAIR_CLASS_INIT(&tag, "tbl");
        h->tblt = print_otag(h, TAG_TABLE, 1, &tag);

        for (hp = sp->head; hp; hp = hp->next) {
                bufinit(h);
                col = &h->tbl.cols[hp->ident];
                SCALE_HS_INIT(&su, col->width);
                bufcat_su(h, "width", &su);
                PAIR_STYLE_INIT(&tag, h);
                print_otag(h, TAG_COL, 1, &tag);
        }

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

void
print_tblclose(struct html *h)
{

        assert(h->tblt);
        print_tagq(h, h->tblt);
        h->tblt = NULL;
}

void
print_tbl(struct html *h, const struct tbl_span *sp)
{
        const struct tbl_head *hp;
        const struct tbl_dat *dp;
        struct htmlpair  tag;
        struct tag      *tt;

        /* Inhibit printing of spaces: we do padding ourselves. */

        if (NULL == h->tblt)
                html_tblopen(h, sp);

        assert(h->tblt);

        h->flags |= HTML_NONOSPACE;
        h->flags |= HTML_NOSPACE;

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

        switch (sp->pos) {
        case (TBL_SPAN_HORIZ):
                /* FALLTHROUGH */
        case (TBL_SPAN_DHORIZ):
                PAIR_INIT(&tag, ATTR_COLSPAN, "0");
                print_otag(h, TAG_TD, 1, &tag);
                break;
        default:
                dp = sp->first;
                for (hp = sp->head; hp; hp = hp->next) {
                        print_stagq(h, tt);
                        print_otag(h, TAG_TD, 0, NULL);

                        switch (hp->pos) {
                        case (TBL_HEAD_VERT):
                                /* FALLTHROUGH */
                        case (TBL_HEAD_DVERT):
                                continue;
                        case (TBL_HEAD_DATA):
```

```
128                                 if (NULL == dp)
129                                         break;
130                                 if (TBL_CELL_DOWN != dp->layout->pos)
131                                         if (dp->string)
132                                                 print_text(h, dp->string);
133                                 dp = dp->next;
134                                 break;
135                         }
136                 }
137                 break;
138         }

140         print_tagq(h, tt);

142         h->flags &= ~HTML_NONOSPACE;

144         if (TBL_SPAN_LAST & sp->flags) {
145                 assert(h->tbl.cols);
146                 free(h->tbl.cols);
147                 h->tbl.cols = NULL;
148                 print_tblclose(h);
149         }

151 }
```

```
**********************************************************
   10207 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl_layout.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: tbl_layout.c,v 1.22 2011/09/18 14:14:15 schwarze Exp $ */
/*
 * 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.
 */
#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"

struct  tbl_phrase {
        char             name;
        enum tbl_cellt   key;
};

/*
 * FIXME: we can make this parse a lot nicer by, when an error is
 * encountered in a layout key, bailing to the next key (i.e. to the
 * next whitespace then continuing).
 */

#define KEYS_MAX         11

static  const struct tbl_phrase keys[KEYS_MAX] = {
        { 'c',           TBL_CELL_CENTRE },
        { 'r',           TBL_CELL_RIGHT },
        { 'l',           TBL_CELL_LEFT },
        { 'n',           TBL_CELL_NUMBER },
        { 's',           TBL_CELL_SPAN },
        { 'a',           TBL_CELL_LONG },
        { '^',           TBL_CELL_DOWN },
        { '-',           TBL_CELL_HORIZ },
        { '_',           TBL_CELL_HORIZ },
        { '=',           TBL_CELL_DHORIZ },
        { '|',           TBL_CELL_VERT }
};

static  int              mods(struct tbl_node *, struct tbl_cell *,
                               int, const char *, int *);
static  int              cell(struct tbl_node *, struct tbl_row *,
                               int, const char *, int *);
```



```
static  void             row(struct tbl_node *, int, const char *, int *);
static  struct tbl_cell *cell_alloc(struct tbl_node *,
                               struct tbl_row *, enum tbl_cellt);
static  void             head_adjust(const struct tbl_cell *,
                               struct tbl_head *);

static int
mods(struct tbl_node *tbl, struct tbl_cell *cp,
                int ln, const char *p, int *pos)
{
        char             buf[5];
        int              i;

        /* Not all types accept modifiers. */

        switch (cp->pos) {
        case (TBL_CELL_DOWN):
                /* FALLTHROUGH */
        case (TBL_CELL_HORIZ):
                /* FALLTHROUGH */
        case (TBL_CELL_DHORIZ):
                /* FALLTHROUGH */
        case (TBL_CELL_VERT):
                /* FALLTHROUGH */
        case (TBL_CELL_DVERT):
                return(1);
        default:
                break;
        }

mod:
        /*
         * XXX: since, at least for now, modifiers are non-conflicting
         * (are separable by value, regardless of position), we let
         * modifiers come in any order.  The existing tbl doesn't let
         * this happen.
         */
        switch (p[*pos]) {
        case ('\0'):
                /* FALLTHROUGH */
        case (' '):
                /* FALLTHROUGH */
        case ('\t'):
                /* FALLTHROUGH */
        case (','):
                /* FALLTHROUGH */
        case ('.'):
                return(1);
        default:
                break;
        }

        /* Throw away parenthesised expression. */

        if ('(' == p[*pos]) {
                (*pos)++;
                while (p[*pos] && ')' != p[*pos])
                        (*pos)++;
                if (')' == p[*pos]) {
                        (*pos)++;
                        goto mod;
                }
                mandoc_msg(MANDOCERR_TBLLAYOUT,
                                tbl->parse, ln, *pos, NULL);
                return(0);
        }
```

```
129         /* Parse numerical spacing from modifier string. */

131         if (isdigit((unsigned char)p[*pos])) {
132                 for (i = 0; i < 4; i++) {
133                         if ( ! isdigit((unsigned char)p[*pos + i]))
134                                 break;
135                         buf[i] = p[*pos + i];
136                 }
137                 buf[i] = '\0';

139                 /* No greater than 4 digits. */

141                 if (4 == i) {
142                         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
143                                         ln, *pos, NULL);
144                         return(0);
145                 }

147                 *pos += i;
148                 cp->spacing = (size_t)atoi(buf);

150                 goto mod;
151                 /* NOTREACHED */
152         }

154         /* TODO: GNU has many more extensions. */

156         switch (tolower((unsigned char)p[(*pos)++])) {
157         case ('z'):
158                 cp->flags |= TBL_CELL_WIGN;
159                 goto mod;
160         case ('u'):
161                 cp->flags |= TBL_CELL_UP;
162                 goto mod;
163         case ('e'):
164                 cp->flags |= TBL_CELL_EQUAL;
165                 goto mod;
166         case ('t'):
167                 cp->flags |= TBL_CELL_TALIGN;
168                 goto mod;
169         case ('d'):
170                 cp->flags |= TBL_CELL_BALIGN;
171                 goto mod;
172         case ('w'):  /* XXX for now, ignore minimal column width */
173                 goto mod;
174         case ('f'):
175                 break;
176         case ('r'):
177                 /* FALLTHROUGH */
178         case ('b'):
179                 /* FALLTHROUGH */
180         case ('i'):
181                 (*pos)--;
182                 break;
183         default:
184                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
185                                 ln, *pos - 1, NULL);
186                 return(0);
187         }

189         switch (tolower((unsigned char)p[(*pos)++])) {
190         case ('3'):
191                 /* FALLTHROUGH */
192         case ('b'):
193                 cp->flags |= TBL_CELL_BOLD;
```



```
194                 goto mod;
195         case ('2'):
196                 /* FALLTHROUGH */
197         case ('i'):
198                 cp->flags |= TBL_CELL_ITALIC;
199                 goto mod;
200         case ('1'):
201                 /* FALLTHROUGH */
202         case ('r'):
203                 goto mod;
204         default:
205                 break;
206         }

208         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
209                         ln, *pos - 1, NULL);
210         return(0);
211 }

213 static int
214 cell(struct tbl_node *tbl, struct tbl_row *rp,
215                 int ln, const char *p, int *pos)
216 {
217         int              i;
218         enum tbl_cellt   c;

220         /* Parse the column position ('r', 'R', '|', ...). */

222         for (i = 0; i < KEYS_MAX; i++)
223                 if (tolower((unsigned char)p[*pos]) == keys[i].name)
224                         break;

226         if (KEYS_MAX == i) {
227                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
228                                 ln, *pos, NULL);
229                 return(0);
230         }

232         c = keys[i].key;

234         /*
235          * If a span cell is found first, raise a warning and abort the
236          * parse.  If a span cell is found and the last layout element
237          * isn't a "normal" layout, bail.
238          *
239          * FIXME: recover from this somehow?
240          */

242         if (TBL_CELL_SPAN == c) {
243                 if (NULL == rp->first) {
244                         mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
245                                         ln, *pos, NULL);
246                         return(0);
247                 } else if (rp->last)
248                         switch (rp->last->pos) {
249                         case (TBL_CELL_VERT):
250                         case (TBL_CELL_DVERT):
251                         case (TBL_CELL_HORIZ):
252                         case (TBL_CELL_DHORIZ):
253                                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse,
254                                                 ln, *pos, NULL);
255                                 return(0);
256                         default:
257                                 break;
258                         }
259         }
```

```
261         /*
262          * If a vertical spanner is found, we may not be in the first
263          * row.
264          */

266         if (TBL_CELL_DOWN == c && rp == tbl->first_row) {
267                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse, ln, *pos, NULL);
268                 return(0);
269         }

271         (*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
280         /* Disallow adjacent spacers. */

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)) {
285                 mandoc_msg(MANDOCERR_TBLLAYOUT, tbl->parse, ln, *pos - 1, NULL);
286                 return(0);
287         }

289         /* Allocate cell then parse its modifiers. */

291         return(mods(tbl, cell_alloc(tbl, rp, c), ln, p, pos));
292 }


295 static void
296 row(struct tbl_node *tbl, int ln, const char *p, int *pos)
297 {
298         struct tbl_row  *rp;

300 row:    /*
301          * EBNF describing this section:
302          *
303          * row          ::= row_list [:space:]* [.]?[\n]
304          * row_list     ::= [:space:]* row_elem row_tail
305          * row_tail     ::= [:space:]*[,] row_list |
306          *                  epsilon
307          * row_elem     ::= [\t\ ]*[:alpha:]+
308          */

310         rp = mandoc_calloc(1, sizeof(struct tbl_row));
311         if (tbl->last_row) {
312                 tbl->last_row->next = rp;
313                 tbl->last_row = rp;
314         } else
315                 tbl->last_row = tbl->first_row = rp;

317 cell:
318         while (isspace((unsigned char)p[*pos]))
319                 (*pos)++;

321         /* Safely exit layout context. */

323         if ('.' == p[*pos]) {
324                 tbl->part = TBL_PART_DATA;
325                 if (NULL == tbl->first_row)
```



```
326                         mandoc_msg(MANDOCERR_TBLNOLAYOUT, tbl->parse,
327                                         ln, *pos, NULL);
328                 (*pos)++;
329                 return;
330         }

332         /* End (and possibly restart) a row. */

334         if (',' == p[*pos]) {
335                 (*pos)++;
336                 goto row;
337         } else if ('\0' == p[*pos])
338                 return;

340         if ( ! cell(tbl, rp, ln, p, pos))
341                 return;

343         goto cell;
344         /* NOTREACHED */
345 }

347 int
348 tbl_layout(struct tbl_node *tbl, int ln, const char *p)
349 {
350         int              pos;

352         pos = 0;
353         row(tbl, ln, p, &pos);

355         /* Always succeed. */
356         return(1);
357 }

359 static struct tbl_cell *
360 cell_alloc(struct tbl_node *tbl, struct tbl_row *rp, enum tbl_cellt pos)
361 {
362         struct tbl_cell *p, *pp;
363         struct tbl_head *h, *hp;

365         p = mandoc_calloc(1, sizeof(struct tbl_cell));

367         if (NULL != (pp = rp->last)) {
368                 rp->last->next = p;
369                 rp->last = p;
370         } else
371                 rp->last = rp->first = p;

373         p->pos = pos;

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;
396         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)) {
401                         p->head = h;
402                         return(p);
403                 }
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)) {
418                         hp = mandoc_calloc(1, sizeof(struct tbl_head));
419                         hp->ident = tbl->opts.cols++;
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++;
444         if (tbl->last_head) {
445                 hp->prev = tbl->last_head;
446                 tbl->last_head->next = hp;
447                 tbl->last_head = hp;
448         } else
449                 tbl->last_head = tbl->first_head = hp;
451         head_adjust(p, hp);
452         p->head = hp;
453         return(p);
454 }
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 }
```

```
**********************************************************
    6142 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl_opts.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: tbl_opts.c,v 1.12 2011/09/18 14:14:15 schwarze Exp $ */
/*
 * 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.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

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

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

enum    tbl_ident {
        KEY_CENTRE = 0,
        KEY_DELIM,
        KEY_EXPAND,
        KEY_BOX,
        KEY_DBOX,
        KEY_ALLBOX,
        KEY_TAB,
        KEY_LINESIZE,
        KEY_NOKEEP,
        KEY_DPOINT,
        KEY_NOSPACE,
        KEY_FRAME,
        KEY_DFRAME,
        KEY_MAX
};

struct  tbl_phrase {
        const char      *name;
        int              key;
        enum tbl_ident   ident;
};

/* Handle Commonwealth/American spellings. */
#define KEY_MAXKEYS      14

/* Maximum length of key name string. */
#define KEY_MAXNAME      13

/* Maximum length of key number size. */
#define KEY_MAXNUMSZ     10
```



```
static  const struct tbl_phrase keys[KEY_MAXKEYS] = {
        { "center",      TBL_OPT_CENTRE,        KEY_CENTRE},
        { "centre",      TBL_OPT_CENTRE,        KEY_CENTRE},
        { "delim",       0,                     KEY_DELIM},
        { "expand",      TBL_OPT_EXPAND,        KEY_EXPAND},
        { "box",         TBL_OPT_BOX,           KEY_BOX},
        { "doublebox",   TBL_OPT_DBOX,          KEY_DBOX},
        { "allbox",      TBL_OPT_ALLBOX,        KEY_ALLBOX},
        { "frame",       TBL_OPT_BOX,           KEY_FRAME},
        { "doubleframe", TBL_OPT_DBOX,          KEY_DFRAME},
        { "tab",         0,                     KEY_TAB},
        { "linesize",    0,                     KEY_LINESIZE},
        { "nokeep",      TBL_OPT_NOKEEP,        KEY_NOKEEP},
        { "decimalpoint", 0,                    KEY_DPOINT},
        { "nospaces",    TBL_OPT_NOSPACE,       KEY_NOSPACE},
};

static  int              arg(struct tbl_node *, int,
                             const char *, int *, enum tbl_ident);
static  void             opt(struct tbl_node *, int,
                             const char *, int *);

static int
arg(struct tbl_node *tbl, int ln, const char *p, int *pos, enum tbl_ident key)
{
        int              i;
        char             buf[KEY_MAXNUMSZ];

        while (isspace((unsigned char)p[*pos]))
                (*pos)++;

        /* Arguments always begin with a parenthesis. */

        if ('(' != p[*pos]) {
                mandoc_msg(MANDOCERR_TBL, tbl->parse,
                                ln, *pos, NULL);
                return(0);
        }

        (*pos)++;

        /*
         * The arguments can be ANY value, so we can't just stop at the
         * next close parenthesis (the argument can be a closed
         * parenthesis itself).
         */

        switch (key) {
        case (KEY_DELIM):
                if ('\0' == p[(*pos)++]) {
                        mandoc_msg(MANDOCERR_TBL, tbl->parse,
                                        ln, *pos - 1, NULL);
                        return(0);
                }

                if ('\0' == p[(*pos)++]) {
                        mandoc_msg(MANDOCERR_TBL, tbl->parse,
                                        ln, *pos - 1, NULL);
                        return(0);
                }
                break;
        case (KEY_TAB):
                if ('\0' != (tbl->opts.tab = p[(*pos)++]))
                        break;

                mandoc_msg(MANDOCERR_TBL, tbl->parse,
```

```
128                                 ln, *pos - 1, NULL);
129                 return(0);
130         case (KEY_LINESIZE):
131                 for (i = 0; i < KEY_MAXNUMSZ && p[*pos]; i++, (*pos)++) {
132                         buf[i] = p[*pos];
133                         if ( ! isdigit((unsigned char)buf[i]))
134                                 break;
135                 }

137                 if (i < KEY_MAXNUMSZ) {
138                         buf[i] = '\0';
139                         tbl->opts.linesize = atoi(buf);
140                         break;
141                 }

143                 mandoc_msg(MANDOCERR_TBL, tbl->parse, ln, *pos, NULL);
144                 return(0);
145         case (KEY_DPOINT):
146                 if ('\0' != (tbl->opts.decimal = p[(*pos)++]))
147                         break;

149                 mandoc_msg(MANDOCERR_TBL, tbl->parse,
150                                 ln, *pos - 1, NULL);
151                 return(0);
152         default:
153                 abort();
154                 /* NOTREACHED */
155         }

157         /* End with a close parenthesis. */

159         if (')' == p[(*pos)++])
160                 return(1);

162         mandoc_msg(MANDOCERR_TBL, tbl->parse, ln, *pos - 1, NULL);
163         return(0);
164 }

166 static void
167 opt(struct tbl_node *tbl, int ln, const char *p, int *pos)
168 {
169         int              i, sv;
170         char             buf[KEY_MAXNAME];

172         /*
173          * Parse individual options from the stream as surrounded by
174          * this goto.  Each pass through the routine parses out a single
175          * option and registers it.  Option arguments are processed in
176          * the arg() function.
177          */

179 again:  /*
180          * EBNF describing this section:
181          *
182          * options      ::= option_list [:space:]* [;][\n]
183          * option_list  ::= option option_tail
184          * option_tail  ::= [:space:]+ option_list |
185          *                  ::= epsilon
186          * option       ::= [:alpha:]+ args
187          * args         ::= [:space:]* [(] [:alpha:]+ [)]
188          */

190         while (isspace((unsigned char)p[*pos]))
191                 (*pos)++;

193         /* Safe exit point. */
```



```
195         if (';' == p[*pos])
196                 return;

198         /* Copy up to first non-alpha character. */

200         for (sv = *pos, i = 0; i < KEY_MAXNAME; i++, (*pos)++) {
201                 buf[i] = (char)tolower((unsigned char)p[*pos]);
202                 if ( ! isalpha((unsigned char)buf[i]))
203                         break;
204         }

206         /* Exit if buffer is empty (or overrun). */

208         if (KEY_MAXNAME == i || 0 == i) {
209                 mandoc_msg(MANDOCERR_TBL, tbl->parse, ln, *pos, NULL);
210                 return;
211         }

213         buf[i] = '\0';

215         while (isspace((unsigned char)p[*pos]))
216                 (*pos)++;

218         /*
219          * Look through all of the available keys to find one that
220          * matches the input.  FIXME: hashtable this.
221          */

223         for (i = 0; i < KEY_MAXKEYS; i++) {
224                 if (strcmp(buf, keys[i].name))
225                         continue;

227                 /*
228                  * Note: this is more difficult to recover from, as we
229                  * can be anywhere in the option sequence and it's
230                  * harder to jump to the next.  Meanwhile, just bail out
231                  * of the sequence altogether.
232                  */

234                 if (keys[i].key)
235                         tbl->opts.opts |= keys[i].key;
236                 else if ( ! arg(tbl, ln, p, pos, keys[i].ident))
237                         return;

239                 break;
240         }

242         /*
243          * Allow us to recover from bad options by continuing to another
244          * parse sequence.
245          */

247         if (KEY_MAXKEYS == i)
248                 mandoc_msg(MANDOCERR_TBLOPT, tbl->parse, ln, sv, NULL);

250         goto again;
251         /* NOTREACHED */
252 }

254 int
255 tbl_option(struct tbl_node *tbl, int ln, const char *p)
256 {
257         int              pos;

259         /*
```

```
260          * Table options are always on just one line, so automatically
261          * switch into the next input mode here.
262          */
263         tbl->part = TBL_PART_LAYOUT;

265         pos = 0;
266         opt(tbl, ln, p, &pos);

268         /* Always succeed. */
269         return(1);
270 }
```

```
**********************************************************
   10029 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/tbl_term.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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 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 *,
                        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 *,
                        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));
}

static size_t
term_tbl_len(size_t sz, void *arg)
{

        return(term_len((const struct termp *)arg, sz));
}
```



```
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->tbl->opts)
                        tbl_hframe(tp, sp, 1);
                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->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;
        case (TBL_SPAN_DATA):
                /* Iterate over template headers. */
                dp = sp->first;
                spans = 0;
```

```
128                 for (hp = sp->head; hp; hp = hp->next) {
129                         /*
130                          * If the current data header is invoked during
131                          * a spanner ("spans" > 0), don't emit anything
132                          * at all.
133                          */
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                         }
144
145                         if (--spans >= 0)
146                                 continue;
147
148                         /*
149                          * All cells get a leading blank, except the
150                          * first one and those after double rulers.
151                          */
152
153                         if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
154                                 tbl_char(tp, ASCII_NBRSP, 1);
155
156                         col = &tp->tbl.cols[hp->ident];
157                         tbl_data(tp, sp->tbl, dp, col);
158
159                         /* No trailing blanks. */
160
161                         if (NULL == hp->next)
162                                 break;
163
164                         /*
165                          * Add another blank between cells,
166                          * or two when there is no vertical ruler.
167                          */
168
169                         tbl_char(tp, ASCII_NBRSP,
170                             TBL_HEAD_VERT  == hp->next->pos ||
171                             TBL_HEAD_DVERT == hp->next->pos ? 1 : 2);
172
173                         /*
174                          * Go to the next data cell and assign the
175                          * number of subsequent spans, if applicable.
176                          */
177
178                         if (dp) {
179                                 spans = dp->spans;
180                                 dp = dp->next;
181                         }
182                 }
183                 break;
184         }
185
186         /* Vertical frame at the end of each row. */
187
188         if (TBL_OPT_BOX & sp->tbl->opts || TBL_OPT_DBOX & sp->tbl->opts)
189                 term_word(tp, TBL_SPAN_HORIZ == sp->pos ||
190                     TBL_SPAN_DHORIZ == sp->pos ? "+" : " |");
191         term_flushln(tp);
192
193         /*
```



```
194          * If we're the last row, clean up after ourselves: clear the
195          * existing table configuration and set it to NULL.
196          */
197
198         if (TBL_SPAN_LAST & sp->flags) {
199                 if (TBL_OPT_DBOX & sp->tbl->opts ||
200                     TBL_OPT_BOX  & sp->tbl->opts)
201                         tbl_hframe(tp, sp, 0);
202                 if (TBL_OPT_DBOX & sp->tbl->opts)
203                         tbl_hframe(tp, sp, 1);
204                 assert(tp->tbl.cols);
205                 free(tp->tbl.cols);
206                 tp->tbl.cols = NULL;
207         }
208
209         tp->flags &= ~TERMP_NONOSPACE;
210         tp->rmargin = rmargin;
211         tp->maxrmargin = maxrmargin;
212
213 }
214
215 /*
216  * Horizontal rules extend across the entire table.
217  * Calculate the width by iterating over columns.
218  */
219 static size_t
220 tbl_rulewidth(struct termp *tp, const struct tbl_head *hp)
221 {
222         size_t           width;
223
224         width = tp->tbl.cols[hp->ident].width;
225         if (TBL_HEAD_DATA == hp->pos) {
226                 /* Account for leading blanks. */
227                 if (hp->prev && TBL_HEAD_DVERT != hp->prev->pos)
228                         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         }
236         return(width);
237 }
238
239 /*
240  * Rules inside the table can be single or double
241  * and have crossings with vertical rules marked with pluses.
242  */
243 static void
244 tbl_hrule(struct termp *tp, const struct tbl_span *sp)
245 {
246         const struct tbl_head *hp;
247         char             c;
248
249         c = '-';
250         if (TBL_SPAN_DHORIZ == sp->pos)
251                 c = '=';
252
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));
257 }
258
259 /*
```

```
 * Rules above and below the table are always single
 * and have an additional plus at the beginning and end.
 * For double frames, this function is called twice,
 * and the outer one does not have crossings.
 */
static void
tbl_hframe(struct termp *tp, const struct tbl_span *sp, int outer)
{
        const struct tbl_head *hp;

        term_word(tp, "+");
        for (hp = sp->head; hp; hp = hp->next)
                tbl_char(tp,
                    outer || TBL_HEAD_DATA == hp->pos ? '-' : '+',
                    tbl_rulewidth(tp, hp));
        term_word(tp, "+");
        term_flushln(tp);
}

static void
tbl_data(struct termp *tp, const struct tbl *tbl,
                const struct tbl_dat *dp,
                const struct roffcol *col)
{

        if (NULL == dp) {
                tbl_char(tp, ASCII_NBRSP, col->width);
                return;
        }
        assert(dp->layout);

        switch (dp->pos) {
        case (TBL_DATA_NONE):
                tbl_char(tp, ASCII_NBRSP, col->width);
                return;
        case (TBL_DATA_HORIZ):
                /* FALLTHROUGH */
        case (TBL_DATA_NHORIZ):
                tbl_char(tp, '-', col->width);
                return;
        case (TBL_DATA_NDHORIZ):
                /* FALLTHROUGH */
        case (TBL_DATA_DHORIZ):
                tbl_char(tp, '=', col->width);
                return;
        default:
                break;
        }

        switch (dp->layout->pos) {
        case (TBL_CELL_HORIZ):
                tbl_char(tp, '-', col->width);
                break;
        case (TBL_CELL_DHORIZ):
                tbl_char(tp, '=', col->width);
                break;
        case (TBL_CELL_LONG):
                /* FALLTHROUGH */
        case (TBL_CELL_CENTRE):
                /* FALLTHROUGH */
        case (TBL_CELL_LEFT):
                /* FALLTHROUGH */
        case (TBL_CELL_RIGHT):
                tbl_literal(tp, dp, col);
                break;
        case (TBL_CELL_NUMBER):
```



```
                tbl_number(tp, tbl, dp, col);
                break;
        case (TBL_CELL_DOWN):
                tbl_char(tp, ASCII_NBRSP, col->width);
                break;
        default:
                abort();
                /* NOTREACHED */
        }
}

static void
tbl_vrule(struct termp *tp, const struct tbl_head *hp)
{

        switch (hp->pos) {
        case (TBL_HEAD_VERT):
                term_word(tp, "|");
                break;
        case (TBL_HEAD_DVERT):
                term_word(tp, "||");
                break;
        default:
                break;
        }
}

static void
tbl_char(struct termp *tp, char c, size_t len)
{
        size_t          i, sz;
        char            cp[2];

        cp[0] = c;
        cp[1] = '\0';

        sz = term_strlen(tp, cp);

        for (i = 0; i < len; i += sz)
                term_word(tp, cp);
}

static void
tbl_literal(struct termp *tp, const struct tbl_dat *dp,
                const struct roffcol *col)
{
        size_t           len, padl, padr;

        assert(dp->string);
        len = term_strlen(tp, dp->string);
        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;
```

```
392                 padr = 0;
393                 break;
394         default:
395                 break;
396         }

398         tbl_char(tp, ASCII_NBRSP, padl);
399         term_word(tp, dp->string);
400         tbl_char(tp, ASCII_NBRSP, padr);
401 }

403 static void
404 tbl_number(struct termp *tp, const struct tbl *tbl,
405                 const struct tbl_dat *dp,
406                 const struct roffcol *col)
407 {
408         char            *cp;
409         char             buf[2];
410         size_t           sz, psz, ssz, d, padl;
411         int              i;

413         /*
414          * See calc_data_number().  Left-pad by taking the offset of our
415          * and the maximum decimal; right-pad by the remaining amount.
416          */

418         assert(dp->string);

420         sz = term_strlen(tp, dp->string);

422         buf[0] = tbl->decimal;
423         buf[1] = '\0';

425         psz = term_strlen(tp, buf);

427         if (NULL != (cp = strrchr(dp->string, tbl->decimal))) {
428                 buf[1] = '\0';
429                 for (ssz = 0, i = 0; cp != &dp->string[i]; i++) {
430                         buf[0] = dp->string[i];
431                         ssz += term_strlen(tp, buf);
432                 }
433                 d = ssz + psz;
434         } else
435                 d = sz + psz;

437         padl = col->decimal - d;

439         tbl_char(tp, ASCII_NBRSP, padl);
440         term_word(tp, dp->string);
441         if (col->width > sz + padl)
442                 tbl_char(tp, ASCII_NBRSP, col->width - sz - padl);
443 }
```

```
**********************************************************
   15531 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/term.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: term.c,v 1.201 2011/09/21 09:57:13 schwarze Exp $ */
/*
 * 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 <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

static  void            adjbuf(struct termp *p, int);
static  void            bufferc(struct termp *, char);
static  void            encode(struct termp *, const char *, size_t);
static  void            encode1(struct termp *, int);

void
term_free(struct termp *p)
{

        if (p->buf)
                free(p->buf);
        if (p->symtab)
                mchars_free(p->symtab);

        free(p);
}


void
term_begin(struct termp *p, term_margin head,
                term_margin foot, const void *arg)
{

        p->headf = head;
        p->footf = foot;
        p->argf = arg;
```



```
        (*p->begin)(p);
}


void
term_end(struct termp *p)
{

        (*p->end)(p);
}

/*
 * Flush a line of text.  A "line" is loosely defined as being something
 * that should be followed by a newline, regardless of whether it's
 * broken apart by newlines getting there.  A line can also be a
 * fragment of a columnar list ('Bl -tag' or 'Bl -column'), which does
 * not have a trailing newline.
 *
 * The following flags may be specified:
 *
 *  - TERMP_NOBREAK: this is the most important and is used when making
 *    columns.  In short: don't print a newline and instead expect the
 *    next call to do the padding up to the start of the next column.
 *
 *  - TERMP_TWOSPACE: make sure there is room for at least two space
 *    characters of padding.  Otherwise, rather break the line.
 *
 *  - TERMP_DANGLE: don't newline when TERMP_NOBREAK is specified and
 *    the line is overrun, and don't pad-right if it's underrun.
 *
 *  - TERMP_HANG: like TERMP_DANGLE, but doesn't newline when
 *    overrunning, instead save the position and continue at that point
 *    when the next invocation.
 *
 *  In-line line breaking:
 *
 *  If TERMP_NOBREAK is specified and the line overruns the right
 *  margin, it will break and pad-right to the right margin after
 *  writing.  If maxrmargin is violated, it will break and continue
 *  writing from the right-margin, which will lead to the above scenario
 *  upon exit.  Otherwise, the line will break at the right margin.
 */
void
term_flushln(struct termp *p)
{
        int             i;     /* current input position in p->buf */
        size_t          vis;   /* current visual position on output */
        size_t          vbl;   /* number of blanks to prepend to output */
        size_t          vend;  /* end of word visual position on output */
        size_t          bp;    /* visual right border position */
        size_t          dv;    /* temporary for visual pos calculations */
        int             j;     /* temporary loop index for p->buf */
        int             jhy;   /* last hyph before overflow w/r/t j */
        size_t          maxvis; /* output position of visible boundary */
        size_t          mmax; /* used in calculating bp */

        /*
         * First, establish the maximum columns of "visible" content.
         * This is usually the difference between the right-margin and
         * an indentation, but can be, for tagged lists or columns, a
         * small set of values.
         */
        assert  (p->rmargin >= p->offset);
        dv     = p->rmargin - p->offset;
        maxvis = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;
        dv     = p->maxrmargin - p->offset;
```

```
128         mmax   = (int)dv > p->overstep ? dv - (size_t)p->overstep : 0;

130         bp = TERMP_NOBREAK & p->flags ? mmax : maxvis;

132         /*
133          * Calculate the required amount of padding.
134          */
135         vbl = p->offset + p->overstep > p->viscol ?
136               p->offset + p->overstep - p->viscol : 0;

138         vis = vend = 0;
139         i = 0;

141         while (i < p->col) {
142                 /*
143                  * Handle literal tab characters: collapse all
144                  * subsequent tabs into a single huge set of spaces.
145                  */
146                 while (i < p->col && '\t' == p->buf[i]) {
147                         vend = (vis / p->tabwidth + 1) * p->tabwidth;
148                         vbl += vend - vis;
149                         vis = vend;
150                         i++;
151                 }

153                 /*
154                  * Count up visible word characters.  Control sequences
155                  * (starting with the CSI) aren't counted.  A space
156                  * generates a non-printing word, which is valid (the
157                  * space is printed according to regular spacing rules).
158                  */

160                 for (j = i, jhy = 0; j < p->col; j++) {
161                         if ((j && ' ' == p->buf[j]) || '\t' == p->buf[j])
162                                 break;

164                         /* Back over the the last printed character. */
165                         if (8 == p->buf[j]) {
166                                 assert(j);
167                                 vend -= (*p->width)(p, p->buf[j - 1]);
168                                 continue;
169                         }

171                         /* Regular word. */
172                         /* Break at the hyphen point if we overrun. */
173                         if (vend > vis && vend < bp &&
174                                         ASCII_HYPH == p->buf[j])
175                                 jhy = j;

177                         vend += (*p->width)(p, p->buf[j]);
178                 }

180                 /*
181                  * Find out whether we would exceed the right margin.
182                  * If so, break to the next line.
183                  */
184                 if (vend > bp && 0 == jhy && vis > 0) {
185                         vend -= vis;
186                         (*p->endline)(p);
187                         p->viscol = 0;
188                         if (TERMP_NOBREAK & p->flags) {
189                                 vbl = p->rmargin;
190                                 vend += p->rmargin - p->offset;
191                         } else
192                                 vbl = p->offset;
```



```
194                         /* Remove the p->overstep width. */

196                         bp += (size_t)p->overstep;
197                         p->overstep = 0;
198                 }

200                 /* Write out the [remaining] word. */
201                 for ( ; i < p->col; i++) {
202                         if (vend > bp && jhy > 0 && i > jhy)
203                                 break;
204                         if ('\t' == p->buf[i])
205                                 break;
206                         if (' ' == p->buf[i]) {
207                                 j = i;
208                                 while (' ' == p->buf[i])
209                                         i++;
210                                 dv = (size_t)(i - j) * (*p->width)(p, ' ');
211                                 vbl += dv;
212                                 vend += dv;
213                                 break;
214                         }
215                         if (ASCII_NBRSP == p->buf[i]) {
216                                 vbl += (*p->width)(p, ' ');
217                                 continue;
218                         }

220                         /*
221                          * Now we definitely know there will be
222                          * printable characters to output,
223                          * so write preceding white space now.
224                          */
225                         if (vbl) {
226                                 (*p->advance)(p, vbl);
227                                 p->viscol += vbl;
228                                 vbl = 0;
229                         }

231                         if (ASCII_HYPH == p->buf[i]) {
232                                 (*p->letter)(p, '-');
233                                 p->viscol += (*p->width)(p, '-');
234                                 continue;
235                         }

237                         (*p->letter)(p, p->buf[i]);
238                         if (8 == p->buf[i])
239                                 p->viscol -= (*p->width)(p, p->buf[i-1]);
240                         else
241                                 p->viscol += (*p->width)(p, p->buf[i]);
242                 }
243                 vis = vend;
244         }

246         /*
247          * If there was trailing white space, it was not printed;
248          * so reset the cursor position accordingly.
249          */
250         if (vis)
251                 vis -= vbl;

253         p->col = 0;
254         p->overstep = 0;

256         if ( ! (TERMP_NOBREAK & p->flags)) {
257                 p->viscol = 0;
258                 (*p->endline)(p);
259                 return;
```

```
260         }

262         if (TERMP_HANG & p->flags) {
263                 /* We need one blank after the tag. */
264                 p->overstep = (int)(vis - maxvis + (*p->width)(p, ' '));

266                 /*
267                  * Behave exactly the same way as groff:
268                  * If we have overstepped the margin, temporarily move
269                  * it to the right and flag the rest of the line to be
270                  * shorter.
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.
275                  */
276                 if (p->overstep < -1)
277                         p->overstep = 0;
278                 return;

280         } else if (TERMP_DANGLE & p->flags)
281                 return;

283         /* If the column was overrun, break the line. */
284         if (maxvis <= vis +
285             ((TERMP_TWOSPACE & p->flags) ? (*p->width)(p, ' ') : 0)) {
286                 (*p->endline)(p);
287                 p->viscol = 0;
288         }
289 }


292 /*
293  * A newline only breaks an existing line; it won't assert vertical
294  * space.  All data in the output buffer is flushed prior to the newline
295  * assertion.
296  */
297 void
298 term_newln(struct termp *p)
299 {

301         p->flags |= TERMP_NOSPACE;
302         if (p->col || p->viscol)
303                 term_flushln(p);
304 }


307 /*
308  * Asserts a vertical space (a full, empty line-break between lines).
309  * Note that if used twice, this will cause two blank spaces and so on.
310  * All data in the output buffer is flushed prior to the newline
311  * assertion.
312  */
313 void
314 term_vspace(struct termp *p)
315 {

317         term_newln(p);
318         p->viscol = 0;
319         (*p->endline)(p);
320 }

322 void
323 term_fontlast(struct termp *p)
324 {
325         enum termfont    f;
```



```
327         f = p->fontl;
328         p->fontl = p->fontq[p->fonti];
329         p->fontq[p->fonti] = f;
330 }


333 void
334 term_fontrepl(struct termp *p, enum termfont f)
335 {

337         p->fontl = p->fontq[p->fonti];
338         p->fontq[p->fonti] = f;
339 }


342 void
343 term_fontpush(struct termp *p, enum termfont f)
344 {

346         assert(p->fonti + 1 < 10);
347         p->fontl = p->fontq[p->fonti];
348         p->fontq[++p->fonti] = f;
349 }


352 const void *
353 term_fontq(struct termp *p)
354 {

356         return(&p->fontq[p->fonti]);
357 }


360 enum termfont
361 term_fonttop(struct termp *p)
362 {

364         return(p->fontq[p->fonti]);
365 }


368 void
369 term_fontpopq(struct termp *p, const void *key)
370 {

372         while (p->fonti >= 0 && key != &p->fontq[p->fonti])
373                 p->fonti--;
374         assert(p->fonti >= 0);
375 }


378 void
379 term_fontpop(struct termp *p)
380 {

382         assert(p->fonti);
383         p->fonti--;
384 }

386 /*
387  * Handle pwords, partial words, which may be either a single word or a
388  * phrase that cannot be broken down (such as a literal string).  This
389  * handles word styling.
390  */
391 void
```

```
392 term_word(struct termp *p, const char *word)
393 {
394         const char      *seq, *cp;
395         char             c;
396         int              sz, uc;
397         size_t           ssz;
398         enum mandoc_esc  esc;

400         if ( ! (TERMP_NOSPACE & p->flags)) {
401                 if ( ! (TERMP_KEEP & p->flags)) {
402                         if (TERMP_PREKEEP & p->flags)
403                                 p->flags |= TERMP_KEEP;
404                         bufferc(p, ' ');
405                         if (TERMP_SENTENCE & p->flags)
406                                 bufferc(p, ' ');
407                 } else
408                         bufferc(p, ASCII_NBRSP);
409         }

411         if ( ! (p->flags & TERMP_NONOSPACE))
412                 p->flags &= ~TERMP_NOSPACE;
413         else
414                 p->flags |= TERMP_NOSPACE;

416         p->flags &= ~(TERMP_SENTENCE | TERMP_IGNDELIM);

418         while ('\0' != *word) {
419                 if ((ssz = strcspn(word, "\\")) > 0)
420                         encode(p, word, ssz);

422                 word += (int)ssz;
423                 if ('\\' != *word)
424                         continue;

426                 word++;
427                 esc = mandoc_escape(&word, &seq, &sz);
428                 if (ESCAPE_ERROR == esc)
429                         break;

431                 if (TERMENC_ASCII != p->enc)
432                         switch (esc) {
433                         case (ESCAPE_UNICODE):
434                                 uc = mchars_num2uc(seq + 1, sz - 1);
435                                 if ('\0' == uc)
436                                         break;
437                                 encode1(p, uc);
438                                 continue;
439                         case (ESCAPE_SPECIAL):
440                                 uc = mchars_spec2cp(p->symtab, seq, sz);
441                                 if (uc <= 0)
442                                         break;
443                                 encode1(p, uc);
444                                 continue;
445                         default:
446                                 break;
447                         }

449                 switch (esc) {
450                 case (ESCAPE_UNICODE):
451                         encode1(p, '?');
452                         break;
453                 case (ESCAPE_NUMBERED):
454                         c = mchars_num2char(seq, sz);
455                         if ('\0' != c)
456                                 encode(p, &c, 1);
457                         break;
```



```
458                 case (ESCAPE_SPECIAL):
459                         cp = mchars_spec2str(p->symtab, seq, sz, &ssz);
460                         if (NULL != cp)
461                                 encode(p, cp, ssz);
462                         else if (1 == ssz)
463                                 encode(p, seq, sz);
464                         break;
465                 case (ESCAPE_FONTBOLD):
466                         term_fontrepl(p, TERMFONT_BOLD);
467                         break;
468                 case (ESCAPE_FONTITALIC):
469                         term_fontrepl(p, TERMFONT_UNDER);
470                         break;
471                 case (ESCAPE_FONT):
472                         /* FALLTHROUGH */
473                 case (ESCAPE_FONTROMAN):
474                         term_fontrepl(p, TERMFONT_NONE);
475                         break;
476                 case (ESCAPE_FONTPREV):
477                         term_fontlast(p);
478                         break;
479                 case (ESCAPE_NOSPACE):
480                         if ('\0' == *word)
481                                 p->flags |= TERMP_NOSPACE;
482                         break;
483                 default:
484                         break;
485                 }
486         }
487 }

489 static void
490 adjbuf(struct termp *p, int sz)
491 {

493         if (0 == p->maxcols)
494                 p->maxcols = 1024;
495         while (sz >= p->maxcols)
496                 p->maxcols <<= 2;

498         p->buf = mandoc_realloc
499                 (p->buf, sizeof(int) * (size_t)p->maxcols);
500 }

502 static void
503 bufferc(struct termp *p, char c)
504 {

506         if (p->col + 1 >= p->maxcols)
507                 adjbuf(p, p->col + 1);

509         p->buf[p->col++] = c;
510 }

512 /*
513  * See encode().
514  * Do this for a single (probably unicode) value.
515  * Does not check for non-decorated glyphs.
516  */
517 static void
518 encode1(struct termp *p, int c)
519 {
520         enum termfont     f;

522         if (p->col + 4 >= p->maxcols)
523                 adjbuf(p, p->col + 4);
```

```
525         f = term_fonttop(p);

527         if (TERMFONT_NONE == f) {
528                 p->buf[p->col++] = c;
529                 return;
530         } else if (TERMFONT_UNDER == f) {
531                 p->buf[p->col++] = '_';
532         } else
533                 p->buf[p->col++] = c;

535         p->buf[p->col++] = 8;
536         p->buf[p->col++] = c;
537 }

539 static void
540 encode(struct termp *p, const char *word, size_t sz)
541 {
542         enum termfont     f;
543         int               i, len;

545         /* LINTED */
546         len = sz;

548         /*
549          * Encode and buffer a string of characters.  If the current
550          * font mode is unset, buffer directly, else encode then buffer
551          * character by character.
552          */

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++)
558                         p->buf[p->col++] = word[i];
559                 return;
560         }

562         /* Pre-buffer, assuming worst-case. */

564         if (p->col + 1 + (len * 3) >= p->maxcols)
565                 adjbuf(p, p->col + 1 + (len * 3));

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++] = '-';
578                 else
579                         p->buf[p->col++] = word[i];

581                 p->buf[p->col++] = 8;
582                 p->buf[p->col++] = word[i];
583         }
584 }

586 size_t
587 term_len(const struct termp *p, size_t sz)
588 {
```



```
590         return((*p->width)(p, ' ') * sz);
591 }


594 size_t
595 term_strlen(const struct termp *p, const char *cp)
596 {
597         size_t           sz, rsz, i;
598         int              ssz, c;
599         const char      *seq, *rhs;
600         enum mandoc_esc  esc;
601         static const char rej[] = { '\\', ASCII_HYPH, ASCII_NBRSP, '\0' };

603         /*
604          * Account for escaped sequences within string length
605          * calculations.  This follows the logic in term_word() as we
606          * must calculate the width of produced strings.
607          */

609         sz = 0;
610         while ('\0' != *cp) {
611                 rsz = strcspn(cp, rej);
612                 for (i = 0; i < rsz; i++)
613                         sz += (*p->width)(p, *cp++);

615                 c = 0;
616                 switch (*cp) {
617                 case ('\\'):
618                         cp++;
619                         esc = mandoc_escape(&cp, &seq, &ssz);
620                         if (ESCAPE_ERROR == esc)
621                                 return(sz);

623                         if (TERMENC_ASCII != p->enc)
624                                 switch (esc) {
625                                 case (ESCAPE_UNICODE):
626                                         c = mchars_num2uc
627                                                 (seq + 1, ssz - 1);
628                                         if ('\0' == c)
629                                                 break;
630                                         sz += (*p->width)(p, c);
631                                         continue;
632                                 case (ESCAPE_SPECIAL):
633                                         c = mchars_spec2cp
634                                                 (p->symtab, seq, ssz);
635                                         if (c <= 0)
636                                                 break;
637                                         sz += (*p->width)(p, c);
638                                         continue;
639                                 default:
640                                         break;
641                                 }

643                         rhs = NULL;

645                         switch (esc) {
646                         case (ESCAPE_UNICODE):
647                                 sz += (*p->width)(p, '?');
648                                 break;
649                         case (ESCAPE_NUMBERED):
650                                 c = mchars_num2char(seq, ssz);
651                                 if ('\0' != c)
652                                         sz += (*p->width)(p, c);
653                                 break;
654                         case (ESCAPE_SPECIAL):
655                                 rhs = mchars_spec2str
```

```
656                                     (p->symtab, seq, ssz, &rsz);

658                             if (ssz != 1 || rhs)
659                                     break;

661                             rhs = seq;
662                             rsz = ssz;
663                             break;
664                     default:
665                             break;
666                     }

668                     if (NULL == rhs)
669                             break;

671                     for (i = 0; i < rsz; i++)
672                             sz += (*p->width)(p, *rhs++);
673                     break;
674             case (ASCII_NBRSP):
675                     sz += (*p->width)(p, ' ');
676                     cp++;
677                     break;
678             case (ASCII_HYPH):
679                     sz += (*p->width)(p, '-');
680                     cp++;
681                     break;
682             default:
683                     break;
684             }
685     }

687     return(sz);
688 }

690 /* ARGSUSED */
691 size_t
692 term_vspan(const struct termp *p, const struct roffsu *su)
693 {
694     double           r;

696     switch (su->unit) {
697     case (SCALE_CM):
698             r = su->scale * 2;
699             break;
700     case (SCALE_IN):
701             r = su->scale * 6;
702             break;
703     case (SCALE_PC):
704             r = su->scale;
705             break;
706     case (SCALE_PT):
707             r = su->scale / 8;
708             break;
709     case (SCALE_MM):
710             r = su->scale / 1000;
711             break;
712     case (SCALE_VS):
713             r = su->scale;
714             break;
715     default:
716             r = su->scale - 1;
717             break;
718     }

720     if (r < 0.0)
721             r = 0.0;
```



```
722     return(/* LINTED */(size_t)
723                     r);
724 }

726 size_t
727 term_hspan(const struct termp *p, const struct roffsu *su)
728 {
729     double           v;

731     v = ((*p->hspan)(p, su));
732     if (v < 0.0)
733             v = 0.0;
734     return((size_t) /* LINTED */
735                     v);
736 }
```

```
**********************************************************
    4549 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/term.h
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: term.h,v 1.90 2011/12/04 23:10:52 schwarze 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.
 */
#ifndef TERM_H
#define TERM_H

__BEGIN_DECLS

struct  termp;

enum    termenc {
        TERMENC_ASCII,
        TERMENC_LOCALE,
        TERMENC_UTF8
};

enum    termtype {
        TERMTYPE_CHAR,
        TERMTYPE_PS,
        TERMTYPE_PDF
};

enum    termfont {
        TERMFONT_NONE = 0,
        TERMFONT_BOLD,
        TERMFONT_UNDER,
        TERMFONT__MAX
};

#define TERM_MAXMARGIN    100000 /* FIXME */

typedef void    (*term_margin)(struct termp *, const void *);

struct  termp_tbl {
        int               width;        /* width in fixed chars */
        int               decimal;      /* decimal point position */
};

struct  termp {
        enum termtype     type;
        struct rofftbl    tbl;          /* table configuration */
        int               mdocstyle;    /* imitate mdoc(7) output */
        size_t            defindent;    /* Default indent for text. */
        size_t            defrmargin;   /* Right margin of the device. */
        size_t            rmargin;      /* Current right margin. */
        size_t            maxrmargin;   /* Max right margin. */
        int               maxcols;      /* Max size of buf. */
        size_t            offset;       /* Margin offest. */
```



```
        size_t            tabwidth;     /* Distance of tab positions. */
        int               col;          /* Bytes in buf. */
        size_t            viscol;       /* Chars on current line. */
        int               overstep;     /* See termp_flushln(). */
        int               flags;
#define TERMP_SENTENCE   (1 << 1)       /* Space before a sentence. */
#define TERMP_NOSPACE    (1 << 2)       /* No space before words. */
#define TERMP_NOBREAK    (1 << 4)       /* See term_flushln(). */
#define TERMP_IGNDELIM   (1 << 6)       /* Delims like regulars. */
#define TERMP_NONOSPACE  (1 << 7)       /* No space (no autounset). */
#define TERMP_DANGLE     (1 << 8)       /* See term_flushln(). */
#define TERMP_HANG       (1 << 9)       /* See term_flushln(). */
#define TERMP_TWOSPACE   (1 << 10)      /* See term_flushln(). */
#define TERMP_NOSPLIT    (1 << 11)      /* See termp_an_pre/post(). */
#define TERMP_SPLIT      (1 << 12)      /* See termp_an_pre/post(). */
#define TERMP_ANPREC     (1 << 13)      /* See termp_an_pre(). */
#define TERMP_KEEP       (1 << 14)      /* Keep words together. */
#define TERMP_PREKEEP    (1 << 15)      /* ...starting with the next one. */
        int              *buf;          /* Output buffer. */
        enum termenc      enc;          /* Type of encoding. */
        struct mchars    *symtab;       /* Encoded-symbol table. */
        enum termfont     fontl;        /* Last font set. */
        enum termfont     fontq[10];    /* Symmetric fonts. */
        int               fonti;        /* Index of font stack. */
        term_margin       headf;        /* invoked to print head */
        term_margin       footf;        /* invoked to print foot */
        void             (*letter)(struct termp *, int);
        void             (*begin)(struct termp *);
        void             (*end)(struct termp *);
        void             (*endline)(struct termp *);
        void             (*advance)(struct termp *, size_t);
        size_t           (*width)(const struct termp *, int);
        double           (*hspan)(const struct termp *,
                                  const struct roffsu *);
        const void       *argf;         /* arg for headf/footf */
        struct termp_ps  *ps;
};

void              term_eqn(struct termp *, const struct eqn *);
void              term_tbl(struct termp *, const struct tbl_span *);
void              term_free(struct termp *);
void              term_newln(struct termp *);
void              term_vspace(struct termp *);
void              term_word(struct termp *, const char *);
void              term_flushln(struct termp *);
void              term_begin(struct termp *, term_margin,
                       term_margin, const void *);
void              term_end(struct termp *);

size_t            term_hspan(const struct termp *,
                       const struct roffsu *);
size_t            term_vspan(const struct termp *,
                       const struct roffsu *);
size_t            term_strlen(const struct termp *, const char *);
size_t            term_len(const struct termp *, size_t);

enum termfont     term_fonttop(struct termp *);
const void       *term_fontq(struct termp *);
void              term_fontpush(struct termp *, enum termfont);
void              term_fontpop(struct termp *);
void              term_fontpopq(struct termp *, const void *);
void              term_fontrepl(struct termp *, enum termfont);
void              term_fontlast(struct termp *);

__END_DECLS
```

```
128 #endif /*!TERM_H*/
```

```
**********************************************************
    5403 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/term_ascii.c
Initial import of man functionality.
**********************************************************
```

```
/*      $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, "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.
                         */
                        p->mdocstyle = 1;
                        p->defindent = 5;
                        break;
```

```
128                 default:
129                         break;
130                 }
131
132         /* Enforce a lower boundary. */
133         if (p->defrmargin < 58)
134                 p->defrmargin = 58;
135
136         return(p);
137 }
138
139 void *
140 ascii_alloc(char *outopts)
141 {
142
143         return(ascii_init(TERMENC_ASCII, outopts));
144 }
145
146 void *
147 utf8_alloc(char *outopts)
148 {
149
150         return(ascii_init(TERMENC_UTF8, outopts));
151 }
152
153
154 void *
155 locale_alloc(char *outopts)
156 {
157
158         return(ascii_init(TERMENC_LOCALE, outopts));
159 }
160
161 /* ARGSUSED */
162 static size_t
163 ascii_width(const struct termp *p, int c)
164 {
165
166         return(1);
167 }
168
169 void
170 ascii_free(void *arg)
171 {
172
173         term_free((struct termp *)arg);
174 }
175
176 /* ARGSUSED */
177 static void
178 ascii_letter(struct termp *p, int c)
179 {
180
181         putchar(c);
182 }
183
184 static void
185 ascii_begin(struct termp *p)
186 {
187
188         (*p->headf)(p, p->argf);
189 }
190
191 static void
192 ascii_end(struct termp *p)
193 {
```



```
194
195         (*p->footf)(p, p->argf);
196 }
197
198 /* ARGSUSED */
199 static void
200 ascii_endline(struct termp *p)
201 {
202
203         putchar('\n');
204 }
205
206 /* ARGSUSED */
207 static void
208 ascii_advance(struct termp *p, size_t len)
209 {
210         size_t          i;
211
212         for (i = 0; i < len; i++)
213                 putchar(' ');
214 }
215
216 /* ARGSUSED */
217 static double
218 ascii_hspan(const struct termp *p, const struct roffsu *su)
219 {
220         double           r;
221
222         /*
223          * Approximate based on character width.  These are generated
224          * entirely by eyeballing the screen, but appear to be correct.
225          */
226
227         switch (su->unit) {
228         case (SCALE_CM):
229                 r = 4 * su->scale;
230                 break;
231         case (SCALE_IN):
232                 r = 10 * su->scale;
233                 break;
234         case (SCALE_PC):
235                 r = (10 * su->scale) / 6;
236                 break;
237         case (SCALE_PT):
238                 r = (10 * su->scale) / 72;
239                 break;
240         case (SCALE_MM):
241                 r = su->scale / 1000;
242                 break;
243         case (SCALE_VS):
244                 r = su->scale * 2 - 1;
245                 break;
246         default:
247                 r = su->scale;
248                 break;
249         }
250
251         return(r);
252 }
253
254 #ifdef USE_WCHAR
255 /* ARGSUSED */
256 static size_t
257 locale_width(const struct termp *p, int c)
258 {
259         int             rc;
```

```
261         return((rc = wcwidth(c)) < 0 ? 0 : rc);
262 }

264 /* ARGSUSED */
265 static void
266 locale_advance(struct termp *p, size_t len)
267 {
268         size_t          i;

270         for (i = 0; i < len; i++)
271                 putwchar(L' ');
272 }

274 /* ARGSUSED */
275 static void
276 locale_endline(struct termp *p)
277 {

279         putwchar(L'\n');
280 }

282 /* ARGSUSED */
283 static void
284 locale_letter(struct termp *p, int c)
285 {
286
287         putwchar(c);
288 }
289 #endif
```

```
**********************************************************
   23551 Tue Jul 15 13:48:08 2014
new/usr/src/cmd/mandoc/term_ps.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: term_ps.c,v 1.54 2011/10/16 12:20:34 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>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

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

/* These work the buffer used by the header and footer. */
#define PS_BUFSLOP        128

/* Convert PostScript point "x" to an AFM unit. */
#define PNT2AFM(p, x) /* LINTED */ \
        (size_t)((double)(x) * (1000.0 / (double)(p)->ps->scale))

/* Convert an AFM unit "x" to a PostScript points */
#define AFM2PNT(p, x) /* LINTED */ \
        ((double)(x) / (1000.0 / (double)(p)->ps->scale))

struct  glyph {
        unsigned short    wx; /* WX in AFM */
};

struct  font {
        const char       *name; /* FontName in AFM */
#define MAXCHAR           95 /* total characters we can handle */
        struct glyph      gly[MAXCHAR]; /* glyph metrics */
};

struct  termp_ps {
        int               flags;
#define PS_INLINE        (1 << 0)       /* we're in a word */
#define PS_MARGINS       (1 << 1)       /* we're in the margins */
```



```
#define PS_NEWPAGE       (1 << 2)       /* new page, no words yet */
        size_t            pscol;        /* visible column (AFM units) */
        size_t            psrow;        /* visible row (AFM units) */
        char             *psmarg;       /* margin buf */
        size_t            psmargsz;     /* margin buf size */
        size_t            psmargcur;    /* cur index in margin buf */
        char              last;         /* character buffer */
        enum termfont     lastf;        /* last set font */
        size_t            scale;        /* font scaling factor */
        size_t            pages;        /* number of pages shown */
        size_t            lineheight;   /* line height (AFM units) */
        size_t            top;          /* body top (AFM units) */
        size_t            bottom;       /* body bottom (AFM units) */
        size_t            height;       /* page height (AFM units */
        size_t            width;        /* page width (AFM units) */
        size_t            left;         /* body left (AFM units) */
        size_t            header;       /* header pos (AFM units) */
        size_t            footer;       /* footer pos (AFM units) */
        size_t            pdfbytes;     /* current output byte */
        size_t            pdflastpg;    /* byte of last page mark */
        size_t            pdfbody;      /* start of body object */
        size_t           *pdfobjs;      /* table of object offsets */
        size_t            pdfobjsz;     /* size of pdfobjs */
};

static  double            ps_hspan(const struct termp *,
                                const struct roffsu *);
static  size_t            ps_width(const struct termp *, int);
static  void              ps_advance(struct termp *, size_t);
static  void              ps_begin(struct termp *);
static  void              ps_closepage(struct termp *);
static  void              ps_end(struct termp *);
static  void              ps_endline(struct termp *);
static  void              ps_fclose(struct termp *);
static  void              ps_growbuf(struct termp *, size_t);
static  void              ps_letter(struct termp *, int);
static  void              ps_pclose(struct termp *);
static  void              ps_pletter(struct termp *, int);
static  void              ps_printf(struct termp *, const char *, ...);
static  void              ps_putchar(struct termp *, char);
static  void              ps_setfont(struct termp *, enum termfont);
static  struct termp     *pspdf_alloc(char *);
static  void              pdf_obj(struct termp *, size_t);

/*
 * We define, for the time being, three fonts: bold, oblique/italic, and
 * normal (roman).  The following table hard-codes the font metrics for
 * ASCII, i.e., 32--127.
 */

static  const struct font fonts[TERMFONT__MAX] = {
        { "Times-Roman", {
                { 250 },
                { 333 },
                { 408 },
                { 500 },
                { 500 },
                { 833 },
                { 778 },
                { 333 },
                { 333 },
                { 333 },
                { 500 },
                { 564 },
                { 250 },
                { 333 },
```

```
128                 { 250 },
129                 { 278 },
130                 { 500 },
131                 { 500 },
132                 { 500 },
133                 { 500 },
134                 { 500 },
135                 { 500 },
136                 { 500 },
137                 { 500 },
138                 { 500 },
139                 { 500 },
140                 { 278 },
141                 { 278 },
142                 { 564 },
143                 { 564 },
144                 { 564 },
145                 { 444 },
146                 { 921 },
147                 { 722 },
148                 { 667 },
149                 { 667 },
150                 { 722 },
151                 { 611 },
152                 { 556 },
153                 { 722 },
154                 { 722 },
155                 { 333 },
156                 { 389 },
157                 { 722 },
158                 { 611 },
159                 { 889 },
160                 { 722 },
161                 { 722 },
162                 { 556 },
163                 { 722 },
164                 { 667 },
165                 { 556 },
166                 { 611 },
167                 { 722 },
168                 { 722 },
169                 { 944 },
170                 { 722 },
171                 { 722 },
172                 { 611 },
173                 { 333 },
174                 { 278 },
175                 { 333 },
176                 { 469 },
177                 { 500 },
178                 { 333 },
179                 { 444 },
180                 { 500 },
181                 { 444 },
182                 {  500},
183                 {  444},
184                 {  333},
185                 {  500},
186                 {  500},
187                 {  278},
188                 {  278},
189                 {  500},
190                 {  278},
191                 {  778},
192                 {  500},
193                 {  500},
```



```
194                 {  500},
195                 {  500},
196                 {  333},
197                 {  389},
198                 {  278},
199                 {  500},
200                 {  500},
201                 {  722},
202                 {  500},
203                 {  500},
204                 {  444},
205                 {  480},
206                 {  200},
207                 {  480},
208                 {  541},
209         } },
210         { "Times-Bold", {
211                 { 250  },
212                 { 333  },
213                 { 555  },
214                 { 500  },
215                 { 500  },
216                 { 1000 },
217                 { 833  },
218                 { 333  },
219                 { 333  },
220                 { 333  },
221                 { 500  },
222                 { 570  },
223                 { 250  },
224                 { 333  },
225                 { 250  },
226                 { 278  },
227                 { 500  },
228                 { 500  },
229                 { 500  },
230                 { 500  },
231                 { 500  },
232                 { 500  },
233                 { 500  },
234                 { 500  },
235                 { 500  },
236                 { 500  },
237                 { 333  },
238                 { 333  },
239                 { 570  },
240                 { 570  },
241                 { 570  },
242                 { 500  },
243                 { 930  },
244                 { 722  },
245                 { 667  },
246                 { 722  },
247                 { 722  },
248                 { 667  },
249                 { 611  },
250                 { 778  },
251                 { 778  },
252                 { 389  },
253                 { 500  },
254                 { 778  },
255                 { 667  },
256                 { 944  },
257                 { 722  },
258                 { 778  },
259                 { 611  },
```

```
260                 { 778  },
261                 { 722  },
262                 { 556  },
263                 { 667  },
264                 { 722  },
265                 { 722  },
266                 { 1000 },
267                 { 722  },
268                 { 722  },
269                 { 667  },
270                 { 333  },
271                 { 278  },
272                 { 333  },
273                 { 581  },
274                 { 500  },
275                 { 333  },
276                 { 500  },
277                 { 556  },
278                 { 444  },
279                 {  556 },
280                 {  444 },
281                 {  333 },
282                 {  500 },
283                 {  556 },
284                 {  278 },
285                 {  333 },
286                 {  556 },
287                 {  278 },
288                 {  833 },
289                 {  556 },
290                 {  500 },
291                 {  556 },
292                 {  556 },
293                 {  444 },
294                 {  389 },
295                 {  333 },
296                 {  556 },
297                 {  500 },
298                 {  722 },
299                 {  500 },
300                 {  500 },
301                 {  444 },
302                 {  394 },
303                 {  220 },
304                 {  394 },
305                 {  520 },
306         } },
307         { "Times-Italic", {
308                 { 250  },
309                 { 333  },
310                 { 420  },
311                 { 500  },
312                 { 500  },
313                 { 833  },
314                 { 778  },
315                 { 333  },
316                 { 333  },
317                 { 333  },
318                 { 500  },
319                 { 675  },
320                 { 250  },
321                 { 333  },
322                 { 250  },
323                 { 278  },
324                 { 500  },
325                 { 500  },
```



```
326                 { 500  },
327                 { 500  },
328                 { 500  },
329                 { 500  },
330                 { 500  },
331                 { 500  },
332                 { 500  },
333                 { 500  },
334                 { 333  },
335                 { 333  },
336                 { 675  },
337                 { 675  },
338                 { 675  },
339                 { 500  },
340                 { 920  },
341                 { 611  },
342                 { 611  },
343                 { 667  },
344                 { 722  },
345                 { 611  },
346                 { 611  },
347                 { 722  },
348                 { 722  },
349                 { 333  },
350                 { 444  },
351                 { 667  },
352                 { 556  },
353                 { 833  },
354                 { 667  },
355                 { 722  },
356                 { 611  },
357                 { 722  },
358                 { 611  },
359                 { 500  },
360                 { 556  },
361                 { 722  },
362                 { 611  },
363                 { 833  },
364                 { 611  },
365                 { 556  },
366                 { 556  },
367                 { 389  },
368                 { 278  },
369                 { 389  },
370                 { 422  },
371                 { 500  },
372                 { 333  },
373                 { 500  },
374                 { 500  },
375                 { 444  },
376                 {  500 },
377                 {  444 },
378                 {  278 },
379                 {  500 },
380                 {  500 },
381                 {  278 },
382                 {  278 },
383                 {  444 },
384                 {  278 },
385                 {  722 },
386                 {  500 },
387                 {  500 },
388                 {  500 },
389                 {  500 },
390                 {  389 },
391                 {  389 },
```

```
392                 {  278 },
393                 {  500 },
394                 {  444 },
395                 {  667 },
396                 {  444 },
397                 {  444 },
398                 {  389 },
399                 {  400 },
400                 {  275 },
401                 {  400 },
402                 {  541 },
403         } },
404 };

406 void *
407 pdf_alloc(char *outopts)
408 {
409         struct termp    *p;

411         if (NULL != (p = pspdf_alloc(outopts)))
412                 p->type = TERMTYPE_PDF;

414         return(p);
415 }

417 void *
418 ps_alloc(char *outopts)
419 {
420         struct termp    *p;

422         if (NULL != (p = pspdf_alloc(outopts)))
423                 p->type = TERMTYPE_PS;

425         return(p);
426 }

428 static struct termp *
429 pspdf_alloc(char *outopts)
430 {
431         struct termp    *p;
432         unsigned int     pagex, pagey;
433         size_t           marginx, marginy, lineheight;
434         const char      *toks[2];
435         const char      *pp;
436         char            *v;

438         p = mandoc_calloc(1, sizeof(struct termp));
439         p->enc = TERMENC_ASCII;
440         p->ps = mandoc_calloc(1, sizeof(struct termp_ps));

442         p->advance = ps_advance;
443         p->begin = ps_begin;
444         p->end = ps_end;
445         p->endline = ps_endline;
446         p->hspan = ps_hspan;
447         p->letter = ps_letter;
448         p->width = ps_width;
449
450         toks[0] = "paper";
451         toks[1] = NULL;

453         pp = NULL;

455         while (outopts && *outopts)
456                 switch (getsubopt(&outopts, UNCONST(toks), &v)) {
457                 case (0):
```



```
458                         pp = v;
459                         break;
460                 default:
461                         break;
462                 }

464         /* Default to US letter (millimetres). */

466         pagex = 216;
467         pagey = 279;

469         /*
470          * The ISO-269 paper sizes can be calculated automatically, but
471          * it would require bringing in -lm for pow() and I'd rather not
472          * do that.  So just do it the easy way for now.  Since this
473          * only happens once, I'm not terribly concerned.
474          */

476         if (pp && strcasecmp(pp, "letter")) {
477                 if (0 == strcasecmp(pp, "a3")) {
478                         pagex = 297;
479                         pagey = 420;
480                 } else if (0 == strcasecmp(pp, "a4")) {
481                         pagex = 210;
482                         pagey = 297;
483                 } else if (0 == strcasecmp(pp, "a5")) {
484                         pagex = 148;
485                         pagey = 210;
486                 } else if (0 == strcasecmp(pp, "legal")) {
487                         pagex = 216;
488                         pagey = 356;
489                 } else if (2 != sscanf(pp, "%ux%u", &pagex, &pagey))
490                         fprintf(stderr, "%s: Unknown paper\n", pp);
491         }

493         /*
494          * This MUST be defined before any PNT2AFM or AFM2PNT
495          * calculations occur.
496          */

498         p->ps->scale = 11;

500         /* Remember millimetres -> AFM units. */

502         pagex = PNT2AFM(p, ((double)pagex * 2.834));
503         pagey = PNT2AFM(p, ((double)pagey * 2.834));

505         /* Margins are 1/9 the page x and y. */

507         marginx = /* LINTED */
508                 (size_t)((double)pagex / 9.0);
509         marginy = /* LINTED */
510                 (size_t)((double)pagey / 9.0);

512         /* Line-height is 1.4em. */

514         lineheight = PNT2AFM(p, ((double)p->ps->scale * 1.4));

516         p->ps->width = (size_t)pagex;
517         p->ps->height = (size_t)pagey;
518         p->ps->header = pagey - (marginy / 2) - (lineheight / 2);
519         p->ps->top = pagey - marginy;
520         p->ps->footer = (marginy / 2) - (lineheight / 2);
521         p->ps->bottom = marginy;
522         p->ps->left = marginx;
523         p->ps->lineheight = lineheight;
```

```
525         p->defrmargin = pagex - (marginx * 2);
526         return(p);
527 }


530 void
531 pspdf_free(void *arg)
532 {
533         struct termp    *p;

535         p = (struct termp *)arg;

537         if (p->ps->psmarg)
538                 free(p->ps->psmarg);
539         if (p->ps->pdfobjs)
540                 free(p->ps->pdfobjs);

542         free(p->ps);
543         term_free(p);
544 }


547 static void
548 ps_printf(struct termp *p, const char *fmt, ...)
549 {
550         va_list          ap;
551         int              pos, len;

553         va_start(ap, fmt);

555         /*
556          * If we're running in regular mode, then pipe directly into
557          * vprintf().  If we're processing margins, then push the data
558          * into our growable margin buffer.
559          */

561         if ( ! (PS_MARGINS & p->ps->flags)) {
562                 len = vprintf(fmt, ap);
563                 va_end(ap);
564                 p->ps->pdfbytes += /* LINTED */
565                         len < 0 ? 0 : (size_t)len;
566                 return;
567         }

569         /*
570          * XXX: I assume that the in-margin print won't exceed
571          * PS_BUFSLOP (128 bytes), which is reasonable but still an
572          * assumption that will cause pukeage if it's not the case.
573          */

575         ps_growbuf(p, PS_BUFSLOP);

577         pos = (int)p->ps->psmargcur;
578         vsnprintf(&p->ps->psmarg[pos], PS_BUFSLOP, fmt, ap);

580         va_end(ap);

582         p->ps->psmargcur = strlen(p->ps->psmarg);
583 }


586 static void
587 ps_putchar(struct termp *p, char c)
588 {
589         int              pos;
```



```
591         /* See ps_printf(). */

593         if ( ! (PS_MARGINS & p->ps->flags)) {
594                 /* LINTED */
595                 putchar(c);
596                 p->ps->pdfbytes++;
597                 return;
598         }

600         ps_growbuf(p, 2);

602         pos = (int)p->ps->psmargcur++;
603         p->ps->psmarg[pos++] = c;
604         p->ps->psmarg[pos] = '\0';
605 }


608 static void
609 pdf_obj(struct termp *p, size_t obj)
610 {

612         assert(obj > 0);

614         if ((obj - 1) >= p->ps->pdfobjsz) {
615                 p->ps->pdfobjsz = obj + 128;
616                 p->ps->pdfobjs = realloc
617                         (p->ps->pdfobjs,
618                          p->ps->pdfobjsz * sizeof(size_t));
619                 if (NULL == p->ps->pdfobjs) {
620                         perror(NULL);
621                         exit((int)MANDOCLEVEL_SYSERR);
622                 }
623         }

625         p->ps->pdfobjs[(int)obj - 1] = p->ps->pdfbytes;
626         ps_printf(p, "%zu 0 obj\n", obj);
627 }


630 static void
631 ps_closepage(struct termp *p)
632 {
633         int              i;
634         size_t           len, base;

636         /*
637          * Close out a page that we've already flushed to output.  In
638          * PostScript, we simply note that the page must be showed.  In
639          * PDF, we must now create the Length, Resource, and Page node
640          * for the page contents.
641          */

643         assert(p->ps->psmarg && p->ps->psmarg[0]);
644         ps_printf(p, "%s", p->ps->psmarg);

646         if (TERMTYPE_PS != p->type) {
647                 ps_printf(p, "ET\n");

649                 len = p->ps->pdfbytes - p->ps->pdflastpg;
650                 base = p->ps->pages * 4 + p->ps->pdfbody;

652                 ps_printf(p, "endstream\nendobj\n");

654                 /* Length of content. */
655                 pdf_obj(p, base + 1);
```

```
656                 ps_printf(p, "%zu\nendobj\n", len);

658                 /* Resource for content. */
659                 pdf_obj(p, base + 2);
660                 ps_printf(p, "<<\n/ProcSet [/PDF /Text]\n");
661                 ps_printf(p, "/Font <<\n");
662                 for (i = 0; i < (int)TERMFONT__MAX; i++)
663                         ps_printf(p, "/F%d %d 0 R\n", i, 3 + i);
664                 ps_printf(p, ">>\n>>\n");

666                 /* Page node. */
667                 pdf_obj(p, base + 3);
668                 ps_printf(p, "<<\n");
669                 ps_printf(p, "/Type /Page\n");
670                 ps_printf(p, "/Parent 2 0 R\n");
671                 ps_printf(p, "/Resources %zu 0 R\n", base + 2);
672                 ps_printf(p, "/Contents %zu 0 R\n", base);
673                 ps_printf(p, ">>\nendobj\n");
674         } else
675                 ps_printf(p, "showpage\n");

677         p->ps->pages++;
678         p->ps->psrow = p->ps->top;
679         assert( ! (PS_NEWPAGE & p->ps->flags));
680         p->ps->flags |= PS_NEWPAGE;
681 }


684 /* ARGSUSED */
685 static void
686 ps_end(struct termp *p)
687 {
688         size_t           i, xref, base;

690         /*
691          * At the end of the file, do one last showpage.  This is the
692          * same behaviour as groff(1) and works for multiple pages as
693          * well as just one.
694          */

696         if ( ! (PS_NEWPAGE & p->ps->flags)) {
697                 assert(0 == p->ps->flags);
698                 assert('\0' == p->ps->last);
699                 ps_closepage(p);
700         }

702         if (TERMTYPE_PS == p->type) {
703                 ps_printf(p, "%%%%Trailer\n");
704                 ps_printf(p, "%%%%Pages: %zu\n", p->ps->pages);
705                 ps_printf(p, "%%%%EOF\n");
706                 return;
707         }

709         pdf_obj(p, 2);
710         ps_printf(p, "<<\n/Type /Pages\n");
711         ps_printf(p, "/MediaBox [0 0 %zu %zu]\n",
712                         (size_t)AFM2PNT(p, p->ps->width),
713                         (size_t)AFM2PNT(p, p->ps->height));

715         ps_printf(p, "/Count %zu\n", p->ps->pages);
716         ps_printf(p, "/Kids [");

718         for (i = 0; i < p->ps->pages; i++)
719                 ps_printf(p, " %zu 0 R", i * 4 +
720                                 p->ps->pdfbody + 3);
```



```
722         base = (p->ps->pages - 1) * 4 +
723                 p->ps->pdfbody + 4;

725         ps_printf(p, "]\n>>\nendobj\n");
726         pdf_obj(p, base);
727         ps_printf(p, "<<\n");
728         ps_printf(p, "/Type /Catalog\n");
729         ps_printf(p, "/Pages 2 0 R\n");
730         ps_printf(p, ">>\n");
731         xref = p->ps->pdfbytes;
732         ps_printf(p, "xref\n");
733         ps_printf(p, "0 %zu\n", base + 1);
734         ps_printf(p, "0000000000 65535 f \n");

736         for (i = 0; i < base; i++)
737                 ps_printf(p, "%.10zu 00000 n \n",
738                                 p->ps->pdfobjs[(int)i]);

740         ps_printf(p, "trailer\n");
741         ps_printf(p, "<<\n");
742         ps_printf(p, "/Size %zu\n", base + 1);
743         ps_printf(p, "/Root %zu 0 R\n", base);
744         ps_printf(p, "/Info 1 0 R\n");
745         ps_printf(p, ">>\n");
746         ps_printf(p, "startxref\n");
747         ps_printf(p, "%zu\n", xref);
748         ps_printf(p, "%%%%EOF\n");
749 }


752 static void
753 ps_begin(struct termp *p)
754 {
755         time_t           t;
756         int              i;

758         /*
759          * Print margins into margin buffer.  Nothing gets output to the
760          * screen yet, so we don't need to initialise the primary state.
761          */

763         if (p->ps->psmarg) {
764                 assert(p->ps->psmargsz);
765                 p->ps->psmarg[0] = '\0';
766         }

768         /*p->ps->pdfbytes = 0;*/
769         p->ps->psmargcur = 0;
770         p->ps->flags = PS_MARGINS;
771         p->ps->pscol = p->ps->left;
772         p->ps->psrow = p->ps->header;

774         ps_setfont(p, TERMFONT_NONE);

776         (*p->headf)(p, p->argf);
777         (*p->endline)(p);

779         p->ps->pscol = p->ps->left;
780         p->ps->psrow = p->ps->footer;

782         (*p->footf)(p, p->argf);
783         (*p->endline)(p);

785         p->ps->flags &= ~PS_MARGINS;

787         assert(0 == p->ps->flags);
```

```
788         assert(p->ps->psmarg);
789         assert('\0' != p->ps->psmarg[0]);
790
791         /*
792          * Print header and initialise page state.  Following this,
793          * stuff gets printed to the screen, so make sure we're sane.
794          */
795
796         t = time(NULL);
797
798         if (TERMTYPE_PS == p->type) {
799                 ps_printf(p, "%%!PS-Adobe-3.0\n");
800                 ps_printf(p, "%%%%CreationDate: %s", ctime(&t));
801                 ps_printf(p, "%%%%DocumentData: Clean7Bit\n");
802                 ps_printf(p, "%%%%Orientation: Portrait\n");
803                 ps_printf(p, "%%%%Pages: (atend)\n");
804                 ps_printf(p, "%%%%PageOrder: Ascend\n");
805                 ps_printf(p, "%%%%DocumentMedia: "
806                                 "Default %zu %zu 0 () ()\n",
807                                 (size_t)AFM2PNT(p, p->ps->width),
808                                 (size_t)AFM2PNT(p, p->ps->height));
809                 ps_printf(p, "%%%%DocumentNeededResources: font");
810
811                 for (i = 0; i < (int)TERMFONT__MAX; i++)
812                         ps_printf(p, " %s", fonts[i].name);
813
814                 ps_printf(p, "\n%%%%EndComments\n");
815         } else {
816                 ps_printf(p, "%%PDF-1.1\n");
817                 pdf_obj(p, 1);
818                 ps_printf(p, "<<\n");
819                 ps_printf(p, ">>\n");
820                 ps_printf(p, "endobj\n");
821
822                 for (i = 0; i < (int)TERMFONT__MAX; i++) {
823                         pdf_obj(p, (size_t)i + 3);
824                         ps_printf(p, "<<\n");
825                         ps_printf(p, "/Type /Font\n");
826                         ps_printf(p, "/Subtype /Type1\n");
827                         ps_printf(p, "/Name /F%zu\n", i);
828                         ps_printf(p, "/BaseFont /%s\n", fonts[i].name);
829                         ps_printf(p, ">>\n");
830                 }
831         }
832
833         p->ps->pdfbody = (size_t)TERMFONT__MAX + 3;
834         p->ps->pscol = p->ps->left;
835         p->ps->psrow = p->ps->top;
836         p->ps->flags |= PS_NEWPAGE;
837         ps_setfont(p, TERMFONT_NONE);
838 }
839
840
841 static void
842 ps_pletter(struct termp *p, int c)
843 {
844         int              f;
845
846         /*
847          * If we haven't opened a page context, then output that we're
848          * in a new page and make sure the font is correctly set.
849          */
850
851         if (PS_NEWPAGE & p->ps->flags) {
852                 if (TERMTYPE_PS == p->type) {
853                         ps_printf(p, "%%%%Page: %zu %zu\n",
```



```
854                                         p->ps->pages + 1,
855                                         p->ps->pages + 1);
856                         ps_printf(p, "/%s %zu selectfont\n",
857                                         fonts[(int)p->ps->lastf].name,
858                                         p->ps->scale);
859                 } else {
860                         pdf_obj(p, p->ps->pdfbody +
861                                         p->ps->pages * 4);
862                         ps_printf(p, "<<\n");
863                         ps_printf(p, "/Length %zu 0 R\n",
864                                         p->ps->pdfbody + 1 +
865                                         p->ps->pages * 4);
866                         ps_printf(p, ">>\nstream\n");
867                 }
868                 p->ps->pdflastpg = p->ps->pdfbytes;
869                 p->ps->flags &= ~PS_NEWPAGE;
870         }
871
872         /*
873          * If we're not in a PostScript "word" context, then open one
874          * now at the current cursor.
875          */
876
877         if ( ! (PS_INLINE & p->ps->flags)) {
878                 if (TERMTYPE_PS != p->type) {
879                         ps_printf(p, "BT\n/F%d %zu Tf\n",
880                                         (int)p->ps->lastf,
881                                         p->ps->scale);
882                         ps_printf(p, "%.3f %.3f Td\n(",
883                                         AFM2PNT(p, p->ps->pscol),
884                                         AFM2PNT(p, p->ps->psrow));
885                 } else
886                         ps_printf(p, "%.3f %.3f moveto\n(",
887                                         AFM2PNT(p, p->ps->pscol),
888                                         AFM2PNT(p, p->ps->psrow));
889                 p->ps->flags |= PS_INLINE;
890         }
891
892         assert( ! (PS_NEWPAGE & p->ps->flags));
893
894         /*
895          * We need to escape these characters as per the PostScript
896          * specification.  We would also escape non-graphable characters
897          * (like tabs), but none of them would get to this point and
898          * it's superfluous to abort() on them.
899          */
900
901         switch (c) {
902         case ('('):
903                 /* FALLTHROUGH */
904         case (')'):
905                 /* FALLTHROUGH */
906         case ('\\'):
907                 ps_putchar(p, '\\');
908                 break;
909         default:
910                 break;
911         }
912
913         /* Write the character and adjust where we are on the page. */
914
915         f = (int)p->ps->lastf;
916
917         if (c <= 32 || (c - 32 >= MAXCHAR)) {
918                 ps_putchar(p, ' ');
919                 p->ps->pscol += (size_t)fonts[f].gly[0].wx;
```

```
920                 return;
921         }

923         ps_putchar(p, (char)c);
924         c -= 32;
925         p->ps->pscol += (size_t)fonts[f].gly[c].wx;
926 }


929 static void
930 ps_pclose(struct termp *p)
931 {

933         /*
934          * Spit out that we're exiting a word context (this is a
935          * "partial close" because we don't check the last-char buffer
936          * or anything).
937          */

939         if ( ! (PS_INLINE & p->ps->flags))
940                 return;
941
942         if (TERMTYPE_PS != p->type) {
943                 ps_printf(p, ") Tj\nET\n");
944         } else
945                 ps_printf(p, ") show\n");

947         p->ps->flags &= ~PS_INLINE;
948 }


951 static void
952 ps_fclose(struct termp *p)
953 {

955         /*
956          * Strong closure: if we have a last-char, spit it out after
957          * checking that we're in the right font mode.  This will of
958          * course open a new scope, if applicable.
959          *
960          * Following this, close out any scope that's open.
961          */

963         if ('\0' != p->ps->last) {
964                 if (p->ps->lastf != TERMFONT_NONE) {
965                         ps_pclose(p);
966                         ps_setfont(p, TERMFONT_NONE);
967                 }
968                 ps_pletter(p, p->ps->last);
969                 p->ps->last = '\0';
970         }

972         if ( ! (PS_INLINE & p->ps->flags))
973                 return;

975         ps_pclose(p);
976 }


979 static void
980 ps_letter(struct termp *p, int arg)
981 {
982         char            cc, c;

984         /* LINTED */
985         c = arg >= 128 || arg <= 0 ? '?' : arg;
```



```
987         /*
988          * State machine dictates whether to buffer the last character
989          * or not.  Basically, encoded words are detected by checking if
990          * we're an "8" and switching on the buffer.  Then we put "8" in
991          * our buffer, and on the next charater, flush both character
992          * and buffer.  Thus, "regular" words are detected by having a
993          * regular character and a regular buffer character.
994          */

996         if ('\0' == p->ps->last) {
997                 assert(8 != c);
998                 p->ps->last = c;
999                 return;
1000        } else if (8 == p->ps->last) {
1001                assert(8 != c);
1002                p->ps->last = '\0';
1003        } else if (8 == c) {
1004                assert(8 != p->ps->last);
1005                if ('_' == p->ps->last) {
1006                        if (p->ps->lastf != TERMFONT_UNDER) {
1007                                ps_pclose(p);
1008                                ps_setfont(p, TERMFONT_UNDER);
1009                        }
1010                } else if (p->ps->lastf != TERMFONT_BOLD) {
1011                        ps_pclose(p);
1012                        ps_setfont(p, TERMFONT_BOLD);
1013                }
1014                p->ps->last = c;
1015                return;
1016        } else {
1017                if (p->ps->lastf != TERMFONT_NONE) {
1018                        ps_pclose(p);
1019                        ps_setfont(p, TERMFONT_NONE);
1020                }
1021                cc = p->ps->last;
1022                p->ps->last = c;
1023                c = cc;
1024        }

1026        ps_pletter(p, c);
1027 }


1030 static void
1031 ps_advance(struct termp *p, size_t len)
1032 {

1034        /*
1035         * Advance some spaces.  This can probably be made smarter,
1036         * i.e., to have multiple space-separated words in the same
1037         * scope, but this is easier:  just close out the current scope
1038         * and readjust our column settings.
1039         */

1041        ps_fclose(p);
1042        p->ps->pscol += len;
1043 }


1046 static void
1047 ps_endline(struct termp *p)
1048 {

1050        /* Close out any scopes we have open: we're at eoln. */
```

```
1052         ps_fclose(p);

1054         /*
1055          * If we're in the margin, don't try to recalculate our current
1056          * row.  XXX: if the column tries to be fancy with multiple
1057          * lines, we'll do nasty stuff.
1058          */

1060         if (PS_MARGINS & p->ps->flags)
1061                 return;

1063         /* Left-justify. */

1065         p->ps->pscol = p->ps->left;

1067         /* If we haven't printed anything, return. */

1069         if (PS_NEWPAGE & p->ps->flags)
1070                 return;

1072         /*
1073          * Put us down a line.  If we're at the page bottom, spit out a
1074          * showpage and restart our row.
1075          */

1077         if (p->ps->psrow >= p->ps->lineheight +
1078                         p->ps->bottom) {
1079                 p->ps->psrow -= p->ps->lineheight;
1080                 return;
1081         }

1083         ps_closepage(p);
1084 }


1087 static void
1088 ps_setfont(struct termp *p, enum termfont f)
1089 {

1091         assert(f < TERMFONT__MAX);
1092         p->ps->lastf = f;
1093
1094         /*
1095          * If we're still at the top of the page, let the font-setting
1096          * be delayed until we actually have stuff to print.
1097          */

1099         if (PS_NEWPAGE & p->ps->flags)
1100                 return;

1102         if (TERMTYPE_PS == p->type)
1103                 ps_printf(p, "/%s %zu selectfont\n",
1104                                 fonts[(int)f].name,
1105                                 p->ps->scale);
1106         else
1107                 ps_printf(p, "/F%d %zu Tf\n",
1108                                 (int)f,
1109                                 p->ps->scale);
1110 }


1113 /* ARGSUSED */
1114 static size_t
1115 ps_width(const struct termp *p, int c)
1116 {
```



```
1118         if (c <= 32 || c - 32 >= MAXCHAR)
1119                 return((size_t)fonts[(int)TERMFONT_NONE].gly[0].wx);

1121         c -= 32;
1122         return((size_t)fonts[(int)TERMFONT_NONE].gly[c].wx);
1123 }


1126 static double
1127 ps_hspan(const struct termp *p, const struct roffsu *su)
1128 {
1129         double           r;
1130
1131         /*
1132          * All of these measurements are derived by converting from the
1133          * native measurement to AFM units.
1134          */

1136         switch (su->unit) {
1137         case (SCALE_CM):
1138                 r = PNT2AFM(p, su->scale * 28.34);
1139                 break;
1140         case (SCALE_IN):
1141                 r = PNT2AFM(p, su->scale * 72);
1142                 break;
1143         case (SCALE_PC):
1144                 r = PNT2AFM(p, su->scale * 12);
1145                 break;
1146         case (SCALE_PT):
1147                 r = PNT2AFM(p, su->scale * 100);
1148                 break;
1149         case (SCALE_EM):
1150                 r = su->scale *
1151                         fonts[(int)TERMFONT_NONE].gly[109 - 32].wx;
1152                 break;
1153         case (SCALE_MM):
1154                 r = PNT2AFM(p, su->scale * 2.834);
1155                 break;
1156         case (SCALE_EN):
1157                 r = su->scale *
1158                         fonts[(int)TERMFONT_NONE].gly[110 - 32].wx;
1159                 break;
1160         case (SCALE_VS):
1161                 r = su->scale * p->ps->lineheight;
1162                 break;
1163         default:
1164                 r = su->scale;
1165                 break;
1166         }

1168         return(r);
1169 }

1171 static void
1172 ps_growbuf(struct termp *p, size_t sz)
1173 {
1174         if (p->ps->psmargcur + sz <= p->ps->psmargsz)
1175                 return;

1177         if (sz < PS_BUFSLOP)
1178                 sz = PS_BUFSLOP;

1180         p->ps->psmargsz += sz;

1182         p->ps->psmarg = mandoc_realloc
1183                 (p->ps->psmarg, p->ps->psmargsz);
```

```
1184 }
```

```
**********************************************************
    6488 Tue Jul 15 13:48:09 2014
new/usr/src/cmd/mandoc/tree.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: tree.c,v 1.47 2011/09/18 14:14:15 schwarze Exp $ */
/*
 * Copyright (c) 2008, 2009, 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 <assert.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

static  void    print_box(const struct eqn_box *, int);
static  void    print_man(const struct man_node *, int);
static  void    print_mdoc(const struct mdoc_node *, int);
static  void    print_span(const struct tbl_span *, int);


/* ARGSUSED */
void
tree_mdoc(void *arg, const struct mdoc *mdoc)
{

        print_mdoc(mdoc_node(mdoc), 0);
}


/* ARGSUSED */
void
tree_man(void *arg, const struct man *man)
{

        print_man(man_node(man), 0);
}


static void
print_mdoc(const struct mdoc_node *n, int indent)
{
        const char       *p, *t;
        int               i, j;
        size_t            argc, sz;
```



```
        char            **params;
        struct mdoc_argv *argv;

        argv = NULL;
        argc = sz = 0;
        params = NULL;
        t = p = NULL;

        switch (n->type) {
        case (MDOC_ROOT):
                t = "root";
                break;
        case (MDOC_BLOCK):
                t = "block";
                break;
        case (MDOC_HEAD):
                t = "block-head";
                break;
        case (MDOC_BODY):
                if (n->end)
                        t = "body-end";
                else
                        t = "block-body";
                break;
        case (MDOC_TAIL):
                t = "block-tail";
                break;
        case (MDOC_ELEM):
                t = "elem";
                break;
        case (MDOC_TEXT):
                t = "text";
                break;
        case (MDOC_TBL):
                /* FALLTHROUGH */
        case (MDOC_EQN):
                break;
        default:
                abort();
                /* NOTREACHED */
        }

        switch (n->type) {
        case (MDOC_TEXT):
                p = n->string;
                break;
        case (MDOC_BODY):
                p = mdoc_macronames[n->tok];
                break;
        case (MDOC_HEAD):
                p = mdoc_macronames[n->tok];
                break;
        case (MDOC_TAIL):
                p = mdoc_macronames[n->tok];
                break;
        case (MDOC_ELEM):
                p = mdoc_macronames[n->tok];
                if (n->args) {
                        argv = n->args->argv;
                        argc = n->args->argc;
                }
                break;
        case (MDOC_BLOCK):
                p = mdoc_macronames[n->tok];
                if (n->args) {
                        argv = n->args->argv;
```

```
128                         argc = n->args->argc;
129                 }
130                 break;
131         case (MDOC_TBL):
132                 /* FALLTHROUGH */
133         case (MDOC_EQN):
134                 break;
135         case (MDOC_ROOT):
136                 p = "root";
137                 break;
138         default:
139                 abort();
140                 /* NOTREACHED */
141         }

143         if (n->span) {
144                 assert(NULL == p && NULL == t);
145                 print_span(n->span, indent);
146         } else if (n->eqn) {
147                 assert(NULL == p && NULL == t);
148                 print_box(n->eqn->root, indent);
149         } else {
150                 for (i = 0; i < indent; i++)
151                         putchar('\t');

153                 printf("%s (%s)", p, t);

155                 for (i = 0; i < (int)argc; i++) {
156                         printf(" -%s", mdoc_argnames[argv[i].arg]);
157                         if (argv[i].sz > 0)
158                                 printf(" [");
159                         for (j = 0; j < (int)argv[i].sz; j++)
160                                 printf(" [%s]", argv[i].value[j]);
161                         if (argv[i].sz > 0)
162                                 printf(" ]");
163                 }
164
165                 for (i = 0; i < (int)sz; i++)
166                         printf(" [%s]", params[i]);

168                 printf(" %d:%d\n", n->line, n->pos);
169         }

171         if (n->child)
172                 print_mdoc(n->child, indent + 1);
173         if (n->next)
174                 print_mdoc(n->next, indent);
175 }


178 static void
179 print_man(const struct man_node *n, int indent)
180 {
181         const char       *p, *t;
182         int               i;

184         t = p = NULL;

186         switch (n->type) {
187         case (MAN_ROOT):
188                 t = "root";
189                 break;
190         case (MAN_ELEM):
191                 t = "elem";
192                 break;
193         case (MAN_TEXT):
```



```
194                 t = "text";
195                 break;
196         case (MAN_BLOCK):
197                 t = "block";
198                 break;
199         case (MAN_HEAD):
200                 t = "block-head";
201                 break;
202         case (MAN_BODY):
203                 t = "block-body";
204                 break;
205         case (MAN_TAIL):
206                 t = "block-tail";
207                 break;
208         case (MAN_TBL):
209                 /* FALLTHROUGH */
210         case (MAN_EQN):
211                 break;
212         default:
213                 abort();
214                 /* NOTREACHED */
215         }

217         switch (n->type) {
218         case (MAN_TEXT):
219                 p = n->string;
220                 break;
221         case (MAN_ELEM):
222                 /* FALLTHROUGH */
223         case (MAN_BLOCK):
224                 /* FALLTHROUGH */
225         case (MAN_HEAD):
226                 /* FALLTHROUGH */
227         case (MAN_TAIL):
228                 /* FALLTHROUGH */
229         case (MAN_BODY):
230                 p = man_macronames[n->tok];
231                 break;
232         case (MAN_ROOT):
233                 p = "root";
234                 break;
235         case (MAN_TBL):
236                 /* FALLTHROUGH */
237         case (MAN_EQN):
238                 break;
239         default:
240                 abort();
241                 /* NOTREACHED */
242         }

244         if (n->span) {
245                 assert(NULL == p && NULL == t);
246                 print_span(n->span, indent);
247         } else if (n->eqn) {
248                 assert(NULL == p && NULL == t);
249                 print_box(n->eqn->root, indent);
250         } else {
251                 for (i = 0; i < indent; i++)
252                         putchar('\t');
253                 printf("%s (%s) %d:%d\n", p, t, n->line, n->pos);
254         }

256         if (n->child)
257                 print_man(n->child, indent + 1);
258         if (n->next)
259                 print_man(n->next, indent);
```

```
260 }

262 static void
263 print_box(const struct eqn_box *ep, int indent)
264 {
265         int              i;
266         const char      *t;

268         if (NULL == ep)
269                 return;
270         for (i = 0; i < indent; i++)
271                 putchar('\t');

273         t = NULL;
274         switch (ep->type) {
275         case (EQN_ROOT):
276                 t = "eqn-root";
277                 break;
278         case (EQN_LIST):
279                 t = "eqn-list";
280                 break;
281         case (EQN_SUBEXPR):
282                 t = "eqn-expr";
283                 break;
284         case (EQN_TEXT):
285                 t = "eqn-text";
286                 break;
287         case (EQN_MATRIX):
288                 t = "eqn-matrix";
289                 break;
290         }

292         assert(t);
293         printf("%s(%d, %d, %d, %d, %d, \"%s\", \"%s\") %s\n",
294                 t, EQN_DEFSIZE == ep->size ? 0 : ep->size,
295                 ep->pos, ep->font, ep->mark, ep->pile,
296                 ep->left ? ep->left : "",
297                 ep->right ? ep->right : "",
298                 ep->text ? ep->text : "");

300         print_box(ep->first, indent + 1);
301         print_box(ep->next, indent);
302 }

304 static void
305 print_span(const struct tbl_span *sp, int indent)
306 {
307         const struct tbl_dat *dp;
308         int              i;

310         for (i = 0; i < indent; i++)
311                 putchar('\t');

313         switch (sp->pos) {
314         case (TBL_SPAN_HORIZ):
315                 putchar('-');
316                 return;
317         case (TBL_SPAN_DHORIZ):
318                 putchar('=');
319                 return;
320         default:
321                 break;
322         }

324         for (dp = sp->first; dp; dp = dp->next) {
325                 switch (dp->pos) {
```



```
326                 case (TBL_DATA_HORIZ):
327                         /* FALLTHROUGH */
328                 case (TBL_DATA_NHORIZ):
329                         putchar('-');
330                         continue;
331                 case (TBL_DATA_DHORIZ):
332                         /* FALLTHROUGH */
333                 case (TBL_DATA_NDHORIZ):
334                         putchar('=');
335                         continue;
336                 default:
337                         break;
338                 }
339                 printf("[\"%s\"", dp->string ? dp->string : "");
340                 if (dp->spans)
341                         printf("(%d)", dp->spans);
342                 if (NULL == dp->layout)
343                         putchar('*');
344                 putchar(']');
345                 putchar(' ');
346         }

348         printf("(tbl) %d:1\n", sp->line);
349 }
```

```
**********************************************************
    1155 Tue Jul 15 13:48:09 2014
new/usr/src/cmd/mandoc/vol.c
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: vol.c,v 1.9 2011/03/22 14:33:05 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 <stdlib.h>
#include <string.h>
#include <time.h>

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

#define LINE(x, y) \
        if (0 == strcmp(p, x)) return(y);

const char *
mdoc_a2vol(const char *p)
{

#include "vol.in"

        return(NULL);
}
```

```
**********************************************************
    1408 Tue Jul 15 13:48:09 2014
new/usr/src/cmd/mandoc/vol.in
Initial import of man functionality.
**********************************************************
```

```
/*      $Id: vol.in,v 1.6 2010/06/19 20:46:28 kristaps Exp $ */
/*
 * Copyright (c) 2009 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 volume titles for .Dt.
 *
 * Be sure to escape strings.
 */

LINE("USD",             "User\'s Supplementary Documents")
LINE("PS1",             "Programmer\'s Supplementary Documents")
LINE("AMD",             "Ancestral Manual Documents")
LINE("SMM",             "System Manager\'s Manual")
LINE("URM",             "User\'s Reference Manual")
LINE("PRM",             "Programmer\'s Manual")
LINE("KM",              "Kernel Manual")
LINE("IND",             "Manual Master Index")
LINE("MMI",             "Manual Master Index")
LINE("LOCAL",           "Local Manual")
LINE("LOC",             "Local Manual")
LINE("CON",             "Contributed Software Manual")
```

```
**********************************************************
   13186 Tue Jul 15 13:48:09 2014
new/usr/src/man/man1/Makefile
Finish integration.  Use mandoc_preconv, etc.
import complete (hopefully)
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet
   9 # at http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright 2011, Richard Lowe
  14 # Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
  15 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  16 #

  18 include         $(SRC)/Makefile.master

  20 MANSECT=        1

  22 MANFILES=       acctcom.1                       \
  23                 adb.1                           \
  24                 addbib.1                        \
  25                 alias.1                         \
  26                 allocate.1                      \
  27                 amt.1                           \
  28                 appcert.1                       \
  29                 apptrace.1                      \
  30                 apropos.1                       \
  31                 ar.1                            \
  32                 arch.1                          \
  33                 asa.1                           \
  34                 at.1                            \
  35                 atq.1                           \
  36                 atrm.1                          \
  37                 audioconvert.1                  \
  38                 audioctl.1                      \
  39                 audioplay.1                     \
  40                 audiorecord.1                   \
  41                 audiotest.1                     \
  42                 auths.1                         \
  43                 awk.1                           \
  44                 banner.1                        \
  45                 basename.1                      \
  46                 bc.1                            \
  47                 bdiff.1                         \
  48                 bfs.1                           \
  49                 break.1                         \
  50                 builtin.1                       \
  51                 cal.1                           \
  52                 calendar.1                      \
  53                 cancel.1                        \
  54                 cat.1                           \
  55                 cd.1                            \
  56                 cdrw.1                          \
  57                 checknr.1                       \
  58                 chgrp.1                         \
  59                 chkey.1                         \
  60                 chmod.1                         \
```



```
  61                 chown.1                         \
  62                 ckdate.1                        \
  63                 ckgid.1                         \
  64                 ckint.1                         \
  65                 ckitem.1                        \
  66                 ckkeywd.1                       \
  67                 ckpath.1                        \
  68                 ckrange.1                       \
  69                 ckstr.1                         \
  70                 cksum.1                         \
  71                 cktime.1                        \
  72                 ckuid.1                         \
  73                 ckyorn.1                        \
  74                 clear.1                         \
  75                 cmp.1                           \
  76                 col.1                           \
  77                 comm.1                          \
  78                 command.1                       \
  79                 compress.1                      \
  80                 cp.1                            \
  81                 cpio.1                          \
  82                 cputrack.1                      \
  83                 crle.1                          \
  84                 crontab.1                       \
  85                 crypt.1                         \
  86                 csh.1                           \
  87                 csplit.1                        \
  88                 ctags.1                         \
  89                 ctrun.1                         \
  90                 ctstat.1                        \
  91                 ctwatch.1                       \
  92                 cut.1                           \
  93                 date.1                          \
  94                 dc.1                            \
  95                 deallocate.1                    \
  96                 deroff.1                        \
  97                 dhcpinfo.1                      \
  98                 diff.1                          \
  99                 diff3.1                         \
 100                 diffmk.1                        \
 101                 digest.1                        \
 102                 dircmp.1                        \
 103                 dis.1                           \
 104                 disown.1                        \
 105                 dispgid.1                       \
 106                 dispuid.1                       \
 107                 dos2unix.1                      \
 108                 download.1                      \
 109                 dpost.1                         \
 110                 du.1                            \
 111                 dump.1                          \
 112                 dumpcs.1                        \
 113                 echo.1                          \
 114                 ed.1                            \
 115                 egrep.1                         \
 116                 eject.1                         \
 117                 elfdump.1                       \
 118                 elfedit.1                       \
 119                 elfsign.1                       \
 120                 elfwrap.1                       \
 121                 enable.1                        \
 122                 encrypt.1                       \
 123                 enhance.1                       \
 124                 env.1                           \
 125                 eqn.1                           \
 126                 exec.1                          \
```

```
127                exit.1                  \
128                expand.1                \
129                expr.1                  \
130                exstr.1                 \
131                factor.1                \
132                fdformat.1              \
133                fgrep.1                 \
134                file.1                  \
135                filesync.1              \
136                find.1                  \
137                finger.1                \
138                fmt.1                   \
139                fmtmsg.1                \
140                fold.1                  \
141                ftp.1                   \
142                ftpcount.1              \
143                ftpwho.1                \
144                gcore.1                 \
145                gencat.1                \
146                genmsg.1                \
147                getconf.1               \
148                getfacl.1               \
149                getlabel.1              \
150                getopt.1                \
151                getoptcvt.1             \
152                getopts.1               \
153                gettext.1               \
154                gettxt.1                \
155                getzonepath.1           \
156                glob.1                  \
157                gprof.1                 \
158                grep.1                  \
159                groups.1                \
160                hash.1                  \
161                head.1                  \
162                history.1               \
163                hostid.1                \
164                hostname.1              \
165                iconv.1                 \
166                indxbib.1               \
167                Intro.1                 \
168                ipcrm.1                 \
169                ipcs.1                  \
170                isainfo.1               \
171                isalist.1               \
172                jobs.1                  \
173                join.1                  \
174                kbd.1                   \
175                kdestroy.1              \
176                keylogin.1              \
177                keylogout.1             \
178                kill.1                  \
179                kinit.1                 \
180                klist.1                 \
181                kmdb.1                  \
182                kmfcfg.1                \
183                kpasswd.1               \
184                krb5-config.1           \
185                ksh93.1                 \
186                ktutil.1                \
187                lari.1                  \
188                last.1                  \
189                lastcomm.1              \
190                ld.1                    \
191                ldap.1                  \
192                ldapdelete.1            \
```



```
193                ldaplist.1              \
194                ldapmodify.1            \
195                ldapmodrdn.1            \
196                ldapsearch.1            \
197                ldd.1                   \
198                ld.so.1.1               \
199                let.1                   \
200                lex.1                   \
201                lgrpinfo.1              \
202                limit.1                 \
203                line.1                  \
204                list_devices.1          \
205                listusers.1             \
206                ln.1                    \
207                loadkeys.1              \
208                locale.1                \
209                localedef.1             \
210                logger.1                \
211                login.1                 \
212                logname.1               \
213                logout.1                \
214                look.1                  \
215                lookbib.1               \
216                lorder.1                \
217                lp.1                    \
218                lpstat.1                \
219                ls.1                    \
220                m4.1                    \
221                mac.1                   \
222                mach.1                  \
223                machid.1                \
224                madv.so.1.1             \
225                mail.1                  \
226                mailcompat.1            \
227                mailq.1                 \
228                mailstats.1             \
229                mailx.1                 \
230                makekey.1               \
231                man.1                   \
232                mandoc.1                \
233                mconnect.1              \
234                mcs.1                   \
235                mdb.1                   \
236                mesg.1                  \
237                mkdir.1                 \
238                mkmsgs.1                \
239                mktemp.1                \
240                moe.1                   \
241                more.1                  \
242                mpss.so.1.1             \
243                msgcc.1                 \
244                msgcpp.1                \
245                msgcvt.1                \
246                msgfmt.1                \
247                msggen.1                \
248                msgget.1                \
249                mt.1                    \
250                mv.1                    \
251                nawk.1                  \
252                nc.1                    \
253                nca.1                   \
254                ncab2clf.1              \
255                ncakmod.1               \
256                newform.1               \
257                newgrp.1                \
258                news.1                  \
```

```
259                 newtask.1                   \
260                 nice.1                      \
261                 nl.1                        \
262                 nm.1                        \
263                 nohup.1                     \
264                 nroff.1                     \
265                 od.1                        \
266                 on.1                        \
267                 optisa.1                    \
268                 pack.1                      \
269                 pagesize.1                  \
270                 pargs.1                     \
271                 passwd.1                    \
272                 paste.1                     \
273                 pathchk.1                   \
274                 pax.1                       \
275                 pfexec.1                    \
276                 pg.1                        \
277                 pgrep.1                     \
278                 pkginfo.1                   \
279                 pkgmk.1                     \
280                 pkgparam.1                  \
281                 pkgproto.1                  \
282                 pkgtrans.1                  \
283                 pktool.1                    \
284                 plabel.1                    \
285                 plgrp.1                     \
286                 plimit.1                    \
287                 pmadvise.1                  \
288                 pmap.1                      \
289                 postio.1                    \
290                 postprint.1                 \
291                 postreverse.1               \
292                 ppgsz.1                     \
293                 ppriv.1                     \
294                 pr.1                        \
295                 praliases.1                 \
296                 prctl.1                     \
297                 preap.1                     \
298                 prex.1                      \
299                 print.1                     \
300                 printf.1                    \
301                 priocntl.1                  \
302                 proc.1                      \
303                 prof.1                      \
304                 profiles.1                  \
305                 projects.1                  \
306                 ps.1                        \
307                 ptree.1                     \
308                 pvs.1                       \
309                 pwd.1                       \
310                 ranlib.1                    \
311                 rcapstat.1                  \
312                 rcp.1                       \
313                 rdist.1                     \
314                 read.1                      \
315                 readonly.1                  \
316                 refer.1                     \
317                 regcmp.1                    \
318                 renice.1                    \
319                 rev.1                       \
320                 rlogin.1                    \
321                 rm.1                        \
322                 rmformat.1                  \
323                 rmmount.1                   \
324                 roffbib.1                   \
```



```
325                 roles.1                     \
326                 rpcgen.1                    \
327                 rsh.1                       \
328                 runat.1                     \
329                 rup.1                       \
330                 ruptime.1                   \
331                 rusers.1                    \
332                 rwho.1                      \
333                 sar.1                       \
334                 scp.1                       \
335                 script.1                    \
336                 sdiff.1                     \
337                 sed.1                       \
338                 set.1                       \
339                 setfacl.1                   \
340                 setlabel.1                  \
341                 setpgrp.1                   \
342                 sftp.1                      \
343                 shcomp.1                    \
344                 shell_builtins.1            \
345                 shift.1                     \
346                 size.1                      \
347                 sleep.1                     \
348                 smbutil.1                   \
349                 soelim.1                    \
350                 sort.1                      \
351                 sortbib.1                   \
352                 sotruss.1                   \
353                 spell.1                     \
354                 split.1                     \
355                 srchtxt.1                   \
356                 ssh.1                       \
357                 ssh-add.1                   \
358                 ssh-agent.1                 \
359                 ssh-http-proxy-connect.1    \
360                 ssh-keygen.1                \
361                 ssh-keyscan.1               \
362                 ssh-socks5-proxy-connect.1  \
363                 strchg.1                    \
364                 strings.1                   \
365                 strip.1                     \
366                 stty.1                      \
367                 sum.1                       \
368                 suspend.1                   \
369                 svcprop.1                   \
370                 svcs.1                      \
371                 symorder.1                  \
372                 sys-suspend.1               \
373                 tabs.1                      \
374                 tail.1                      \
375                 talk.1                      \
376                 tar.1                       \
377                 tbl.1                       \
378                 tcopy.1                     \
379                 tee.1                       \
380                 telnet.1                    \
381                 test.1                      \
382                 tftp.1                      \
383                 time.1                      \
384                 times.1                     \
385                 timex.1                     \
386                 tip.1                       \
387                 tnfdump.1                   \
388                 tnfxtract.1                 \
389                 touch.1                     \
390                 tput.1                      \
```

```
391                 tr.1                        \
392                 trap.1                      \
393                 troff.1                     \
394                 true.1                      \
395                 truss.1                     \
396                 tsort.1                     \
397                 tty.1                       \
398                 type.1                      \
399                 typeset.1                   \
400                 ul.1                        \
401                 umask.1                     \
402                 uname.1                     \
403                 unifdef.1                   \
404                 uniq.1                      \
405                 units.1                     \
406                 unix2dos.1                  \
407                 uptime.1                    \
408                 vacation.1                  \
409                 vgrind.1                    \
410                 volcheck.1                  \
411                 volrmmount.1                \
412                 w.1                         \
413                 wait.1                      \
414                 wc.1                        \
413                 whatis.1                    \
415                 which.1                     \
416                 who.1                       \
417                 whocalls.1                  \
418                 whois.1                     \
419                 write.1                     \
420                 xargs.1                     \
421                 xgettext.1                  \
422                 xstr.1                      \
423                 yacc.1                      \
424                 yes.1                       \
425                 ypcat.1                     \
426                 ypmatch.1                   \
427                 yppasswd.1                  \
428                 ypwhich.1                   \
429                 zlogin.1                    \
430                 zonename.1

432 MANLINKS=       batch.1         \
433                 bg.1            \
434                 case.1          \
435                 chdir.1         \
436                 checkeq.1       \
437                 continue.1      \
438                 decrypt.1       \
439                 dirname.1       \
440                 dirs.1          \
441                 disable.1       \
442                 dumpkeys.1      \
443                 edit.1          \
444                 errange.1       \
445                 errdate.1       \
446                 errgid.1        \
447                 errint.1        \
448                 erritem.1       \
449                 errpath.1       \
450                 errstr.1        \
451                 errtime.1       \
452                 erruid.1        \
453                 erryorn.1       \
454                 eval.1          \
455                 export.1        \
```



```
456                 false.1         \
457                 fc.1            \
458                 fg.1            \
459                 for.1           \
460                 foreach.1       \
461                 function.1      \
462                 goto.1          \
463                 hashcheck.1     \
464                 hashmake.1      \
465                 hashstat.1      \
466                 helpdate.1      \
467                 helpgid.1       \
468                 helpint.1       \
469                 helpitem.1      \
470                 helppath.1      \
471                 helprange.1     \
472                 helpstr.1       \
473                 helptime.1      \
474                 helpuid.1       \
475                 helpyorn.1      \
476                 hist.1          \
477                 i286.1          \
478                 i386.1          \
479                 i486.1          \
480                 i860.1          \
481                 iAPX286.1       \
482                 if.1            \
483                 intro.1         \
484                 jsh.1           \
485                 ksh.1           \
486                 ldapadd.1       \
487                 neqn.1          \
488                 notify.1        \
489                 onintr.1        \
490                 page.1          \
491                 pcat.1          \
492                 pcred.1         \
493                 pdp11.1         \
494                 pfcsh.1         \
495                 pfiles.1        \
496                 pfksh.1         \
497                 pflags.1        \
498                 pfsh.1          \
499                 pkill.1         \
500                 pldd.1          \
501                 popd.1          \
502                 prun.1          \
503                 psig.1          \
504                 pstack.1        \
505                 pstop.1         \
506                 ptime.1         \
507                 pushd.1         \
508                 pwait.1         \
509                 pwdx.1          \
510                 red.1           \
511                 rehash.1        \
512                 remote_shell.1  \
513                 remsh.1         \
514                 repeat.1        \
515                 return.1        \
516                 rksh.1          \
517                 rksh93.1        \
518                 rmail.1         \
519                 rmdir.1         \
520                 rmumount.1      \
521                 select.1        \
```

```
522                 setenv.1        \
523                 settime.1       \
524                 sh.1            \
525                 snca.1          \
526                 source.1        \
527                 sparc.1         \
528                 spellin.1       \
529                 stop.1          \
530                 strconf.1       \
531                 sun.1           \
532                 switch.1        \
533                 u370.1          \
534                 u3b.1           \
535                 u3b15.1         \
536                 u3b2.1          \
537                 u3b5.1          \
538                 ulimit.1        \
539                 unalias.1       \
540                 uncompress.1    \
541                 unexpand.1      \
542                 unhash.1        \
543                 unlimit.1       \
544                 unpack.1        \
545                 unset.1         \
546                 unsetenv.1      \
547                 until.1         \
548                 valdate.1       \
549                 valgid.1        \
550                 valint.1        \
551                 valpath.1       \
552                 valrange.1      \
553                 valstr.1        \
554                 valtime.1       \
555                 valuid.1        \
556                 valyorn.1       \
557                 vax.1           \
558                 vedit.1         \
559                 whatis.1        \
560                 whence.1        \
561                 while.1         \
562                 zcat.1

564 intro.1         := LINKSRC = Intro.1

566 whatis.1        := LINKSRC = apropos.1

568 unalias.1       := LINKSRC = alias.1

570 batch.1         := LINKSRC = at.1

572 dirname.1       := LINKSRC = basename.1

574 continue.1      := LINKSRC = break.1

576 chdir.1         := LINKSRC = cd.1
577 dirs.1          := LINKSRC = cd.1
578 popd.1          := LINKSRC = cd.1
579 pushd.1         := LINKSRC = cd.1

581 errdate.1       := LINKSRC = ckdate.1
582 helpdate.1      := LINKSRC = ckdate.1
583 valdate.1       := LINKSRC = ckdate.1

585 errgid.1        := LINKSRC = ckgid.1
586 helpgid.1       := LINKSRC = ckgid.1
587 valgid.1        := LINKSRC = ckgid.1
```



```
589 errint.1        := LINKSRC = ckint.1
590 helpint.1       := LINKSRC = ckint.1
591 valint.1        := LINKSRC = ckint.1

593 erritem.1       := LINKSRC = ckitem.1
594 helpitem.1      := LINKSRC = ckitem.1

596 errpath.1       := LINKSRC = ckpath.1
597 helppath.1      := LINKSRC = ckpath.1
598 valpath.1       := LINKSRC = ckpath.1

600 errange.1       := LINKSRC = ckrange.1
601 helprange.1     := LINKSRC = ckrange.1
602 valrange.1      := LINKSRC = ckrange.1

604 errstr.1        := LINKSRC = ckstr.1
605 helpstr.1       := LINKSRC = ckstr.1
606 valstr.1        := LINKSRC = ckstr.1

608 errtime.1       := LINKSRC = cktime.1
609 helptime.1      := LINKSRC = cktime.1
610 valtime.1       := LINKSRC = cktime.1

612 erruid.1        := LINKSRC = ckuid.1
613 helpuid.1       := LINKSRC = ckuid.1
614 valuid.1        := LINKSRC = ckuid.1

616 erryorn.1       := LINKSRC = ckyorn.1
617 helpyorn.1      := LINKSRC = ckyorn.1
618 valyorn.1       := LINKSRC = ckyorn.1

620 uncompress.1    := LINKSRC = compress.1
621 zcat.1          := LINKSRC = compress.1

623 red.1           := LINKSRC = ed.1

625 disable.1       := LINKSRC = enable.1

627 decrypt.1       := LINKSRC = encrypt.1

629 checkeq.1       := LINKSRC = eqn.1
630 neqn.1          := LINKSRC = eqn.1

632 eval.1          := LINKSRC = exec.1
633 source.1        := LINKSRC = exec.1

635 goto.1          := LINKSRC = exit.1
636 return.1        := LINKSRC = exit.1

638 unexpand.1      := LINKSRC = expand.1

640 hashstat.1      := LINKSRC = hash.1
641 rehash.1        := LINKSRC = hash.1
642 unhash.1        := LINKSRC = hash.1

644 fc.1            := LINKSRC = history.1
645 hist.1          := LINKSRC = history.1

647 bg.1            := LINKSRC = jobs.1
648 fg.1            := LINKSRC = jobs.1
649 notify.1        := LINKSRC = jobs.1
650 stop.1          := LINKSRC = jobs.1

652 jsh.1           := LINKSRC = ksh93.1
653 ksh.1           := LINKSRC = ksh93.1
```

```
654 rksh.1          := LINKSRC = ksh93.1
655 rksh93.1        := LINKSRC = ksh93.1
656 sh.1            := LINKSRC = ksh93.1

658 ldapadd.1       := LINKSRC = ldapmodify.1

660 ulimit.1        := LINKSRC = limit.1
661 unlimit.1       := LINKSRC = limit.1

663 dumpkeys.1      := LINKSRC = loadkeys.1

665 i286.1          := LINKSRC = machid.1
666 i386.1          := LINKSRC = machid.1
667 i486.1          := LINKSRC = machid.1
668 i860.1          := LINKSRC = machid.1
669 iAPX286.1       := LINKSRC = machid.1
670 pdp11.1         := LINKSRC = machid.1
671 sparc.1         := LINKSRC = machid.1
672 sun.1           := LINKSRC = machid.1
673 u370.1          := LINKSRC = machid.1
674 u3b.1           := LINKSRC = machid.1
675 u3b15.1         := LINKSRC = machid.1
676 u3b2.1          := LINKSRC = machid.1
677 u3b5.1          := LINKSRC = machid.1
678 vax.1           := LINKSRC = machid.1

680 rmail.1         := LINKSRC = mail.1

682 page.1          := LINKSRC = more.1

684 snca.1          := LINKSRC = nca.1

686 pcat.1          := LINKSRC = pack.1
687 unpack.1        := LINKSRC = pack.1

689 pfcsh.1         := LINKSRC = pfexec.1
690 pfksh.1         := LINKSRC = pfexec.1
691 pfsh.1          := LINKSRC = pfexec.1

693 pkill.1         := LINKSRC = pgrep.1

695 pcred.1         := LINKSRC = proc.1
696 pfiles.1        := LINKSRC = proc.1
697 pflags.1        := LINKSRC = proc.1
698 pldd.1          := LINKSRC = proc.1
699 prun.1          := LINKSRC = proc.1
700 psig.1          := LINKSRC = proc.1
701 pstack.1        := LINKSRC = proc.1
702 pstop.1         := LINKSRC = proc.1
703 ptime.1         := LINKSRC = proc.1
704 pwait.1         := LINKSRC = proc.1
705 pwdx.1          := LINKSRC = proc.1

707 rmdir.1         := LINKSRC = rm.1

709 rmumount.1      := LINKSRC = rmmount.1

711 remote_shell.1  := LINKSRC = rsh.1
712 remsh.1         := LINKSRC = rsh.1

714 export.1        := LINKSRC = set.1
715 setenv.1        := LINKSRC = set.1
716 unset.1         := LINKSRC = set.1
717 unsetenv.1      := LINKSRC = set.1

719 case.1          := LINKSRC = shell_builtins.1
```



```
720 for.1           := LINKSRC = shell_builtins.1
721 foreach.1       := LINKSRC = shell_builtins.1
722 function.1      := LINKSRC = shell_builtins.1
723 if.1            := LINKSRC = shell_builtins.1
724 repeat.1        := LINKSRC = shell_builtins.1
725 select.1        := LINKSRC = shell_builtins.1
726 switch.1        := LINKSRC = shell_builtins.1
727 until.1         := LINKSRC = shell_builtins.1
728 while.1         := LINKSRC = shell_builtins.1

730 hashcheck.1     := LINKSRC = spell.1
731 hashmake.1      := LINKSRC = spell.1
732 spellin.1       := LINKSRC = spell.1

734 strconf.1       := LINKSRC = strchg.1

736 settime.1       := LINKSRC = touch.1

738 onintr.1        := LINKSRC = trap.1

740 false.1         := LINKSRC = true.1

742 whence.1        := LINKSRC = typeset.1

744 # Links to usr/has/man

746 edit.1          := LINKSRC = ../../../has/man/man1has/edit.1has

748 vedit.1         := LINKSRC = ../../../has/man/man1has/vi.1has

750 .KEEP_STATE:

752 include         $(SRC)/man/Makefile.man

754 install:        $(ROOTMANFILES) $(ROOTMANLINKS)
```

```
**********************************************************
    1724 Tue Jul 15 13:48:09 2014
new/usr/src/man/man1/apropos.1
import complete (hopefully)
**********************************************************
.\"
.\" 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>
.\"
.Dd Jul 13, 2014
.Dt APROPOS 1
.Os
.Sh NAME
.Nm apropos ,
.Nm whatis
.Nd keyword search in
.Nm whatis
database files
.Sh SYNOPSIS
.Nm
.Op Fl M Ar path
.Op Fl s Ar section
.Ar keyword ...
.Nm whatis
.Op Fl M Ar path
.Op Fl s Ar section
.Ar keyword ...
.Sh DESCRIPTION
The
.Nm
utility searches a set of
.Nm whatis
database files matching each
.Ar keyword .
The
.Nm whatis
utility does the same search but only on complete words. The
.Nm whatis
database files are created using
.Xr man 1
command.
.Bl -tag -width ".Fl d"
.It Fl M Ar path
Force a specific colon separated manual path instead of the default search path.
Overrides the
.Ev MANPATH
environment variable.
.It Fl s Ar section
Restrict search to specified
.Ar section .
.El
.Sh ENVIRONMENT
The following environment variables affect the execution of
.Nm :
.Bl -tag -width ".Ev MANPATH , PATH"
.It Ev MANPATH , PATH
```



```
Used to find the location of the
.Nm whatis
database files.
.El
.Sh DIAGNOSTICS
The
.Nm
utility exits 0 if a keyword matched and 1 if no keywords are matched or no
.Nm whatis
databases are found.
.Sh INTERFACE STABILITY
Committed.
.Sh CODE SET INDEPENDENCE
Enabled.
.Sh SEE ALSO
.Xr man 1 ,
.Xr mandoc 1
'\" te
.\"  Copyright (c) 1996, Sun Microsystems, Inc.  All Rights Reserved
.\" 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
.TH APROPOS 1 "Dec 20, 1996"
.SH NAME
apropos \- locate commands by keyword lookup
.SH SYNOPSIS
.LP
.nf
\fBapropos\fR \fIkeyword\fR...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBapropos\fR utility displays the man page name, section number, and a
short description for each man page whose \fBNAME\fR line contains
\fIkeyword\fR. This information is contained in the \fB/usr/share/man/windex\fR
database created by \fBcatman\fR(1M). If \fBcatman\fR(1M) was not run, or was
run with the \fB-n\fR option, \fBapropos\fR fails. Each word is considered
separately and the case of letters is ignored. Words which are part of other
words are considered; for example, when looking for 'compile', \fBapropos\fR
finds all instances of 'compiler' also.
.sp
.LP
\fBapropos\fR is actually just the \fB-k\fR option to the \fBman\fR(1) command.
.SH EXAMPLES
.LP
\fBExample 1 \fRTo find a man page whose NAME line contains a keyword
.sp
.LP
Try

.sp
.in +2
.nf
example% \fBapropos password\fR
.fi
.in -2
.sp

.sp
.LP
and

.sp
.in +2
```

```
.nf
example% \fBapropos editor\fR
.fi
.in -2
.sp

.sp
.LP
If the line starts ‘\fIfilename\fR(\fIsection\fR) .\|.\|.’ you can run

.sp
.in +2
.nf
man -s \fIsection filename\fR
.fi
.in -2
.sp

.sp
.LP
to display the man page for \fIfilename\fR.

.LP
\fBExample 2 \fRTo find the man page for the subroutine \fBprintf()\fR
.sp
.LP
Try

.sp
.in +2
.nf
example% \fBapropos format\fR
.fi
.in -2
.sp

.sp
.LP
and then

.sp
.in +2
.nf
example% \fBman -s 3s printf\fR
.fi
.in -2
.sp

.sp
.LP
to get the manual page on the subroutine \fBprintf()\fR.

.SH FILES
.sp
.ne 2
.na
\fB\fB/usr/share/man/windex\fR \fR
.ad
.RS 26n
table of contents and keyword database
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
```



```
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
.TE

.SH SEE ALSO
.sp
.LP
\fBman\fR(1), \fBwhatis\fR(1), \fBcatman\fR(1M), \fBattributes\fR(5)
.SH DIAGNOSTICS
.sp
.ne 2
.na
\fB\f(CW/usr/share/man/windex: No such file or directory\fR\fR
.ad
.sp .6
.RS 4n
This database does not exist. \fBcatman\fR(1M) must be run to create it.
.RE
```

```
**********************************************************
    3089 Tue Jul 15 13:48:09 2014
new/usr/src/man/man1/man.1
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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>
.\"
.Dd October 18, 2012
.Dt MAN 1
.Os
.Sh NAME
.Nm man
.Nd find and display reference manual pages
.Sh SYNOPSIS
.Nm
.Op Fl alptw
.Op Fl M Ar manpath
.Op Fl s Ar mansect
.Ar page ...
.Nm
.Op Fl s Ar mansect
.Fl f
.Ar keyword ...
.Nm
.Op Fl s Ar mansect
.Fl k
.Ar keyword ...
.Sh DESCRIPTION
The
.Nm
utility finds and displays reference manual pages.
.Pp
Options that
.Nm
understands:
.Bl -tag -width indent
.It Fl M Ar manpath
Forces a specific colon separated manual path instead of the default
search path.
Overrides the
.Ev MANPATH
environment variable.
.It Fl a
Display all manual pages instead of just the first found for each
.Ar page
argument.
.It Fl f
Emulate
.Xr whatis 1 .
.It Fl k
Emulate
.Xr apropos 1 .
.It Fl l
Display the location of the manual page instead of the contents of
```



```
the manual page.
.It Fl p
Output current path used for searching.
.It Fl s Ar mansect
Restrict manual sections searched to the specified colon delimited list.
.It Fl t
Send the content formatted as PostScript to the default printer.
.It Fl w
Create
.Nm whatis
databases used by
.Xr apropos 1
and
.Xr whatis 1 .
.El
.Sh ENVIRONMENT
The following environment variables affect the execution of
.Nm :
.Bl -tag -width ".Ev MANPATH"
.It Ev LC_ALL , LC_CTYPE , LANG
Used to find locale-specific manual pages.
.It Ev MANPATH
Used to find the location of the manual files.
Corresponds to the
.Fl M
option.
.It Ev MANWIDTH
Defines the width of output. If set to
.Dq Li tty ,
and output is to a terminal, full width of terminal is used.
.It Ev PAGER
Program used to display files. If unset,
.Dq Li "less -ins"
is used.
.It Ev PATH
Used to find location of manual files if
.Ev MANPATH
and
.Fl M
are not specified.
.El
.Sh FILES
.Bl -tag -width indent -compact
.It Pa man.cf
Per-manpath configuration settings.  The file is formatted as follows:
.Bd -literal -offset indent
MANSECT=\fIsection\fR[,\fIsection\fR]...
.Ed
.Pp
Each section consists of a section in the reference manual.  The file
may also contain comment blank lines or lines consisting of comments, where
the first character in the line is '#'.  Both blank lines and comment lines are
ignored.
.El
.Sh CODE SET INDEPENDENCE
Enabled.
.Sh INTERFACE STABILITY
The
.Nm
utility is Standard, as is the
.Fl k
option. The other options are Committed.
.Sh SEE ALSO
.Xr apropos 1 ,
.Xr intro 1 ,
.Xr mandoc 1 ,
```

```
.Xr whatis 1 ,
.Xr man 5 ,
.Xr mdoc 5 ,
.Xr standards 5
.Sh NOTES
Some pages may contain information which cannot be properly displayed on
all terminals.  In such cases, some information may be lost.
'\" te
.\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
.\" Copyright (c) 1980 Regents of the University of California. The Berkeley sof
.TH MAN 1 "May 8, 2008"
.SH NAME
man \- find and display reference manual pages
.SH SYNOPSIS
.LP
.nf
\fBman\fR [\fB-\fR] [\fB-adFlrt\fR] [\fB-M\fR \fIpath\fR] [\fB-T\fR \fImacro-pac
.fi

.LP
.nf
\fBman\fR [\fB-M\fR \fIpath\fR] \fB-k\fR \fIkeyword\fR...
.fi

.LP
.nf
\fBman\fR [\fB-M\fR \fIpath\fR] \fB-f\fR \fIfile\fR...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBman\fR command displays information from the reference manuals. It
displays complete manual pages that you select by \fIname\fR, or one-line
summaries selected either by \fIkeyword\fR (\fB-k\fR), or by the name of an
associated file (\fB-f\fR). If no manual page is located, \fBman\fR prints an
error message.
.SS "Source Format"
.sp
.LP
Reference Manual pages are marked up with either \fBnroff\fR (see
\fBnroff\fR(1)) or \fBSGML\fR (Standard Generalized Markup Language) tags (see
\fBsgml\fR(5)). The \fBman\fR command recognizes the type of markup and
processes the file accordingly. The various source files are kept in separate
directories depending on the type of markup.
.SS "Location of Manual Pages"
.sp
.LP
The online Reference Manual page directories are conventionally located in
\fB/usr/share/man\fR. The nroff sources are located in the
\fB/usr/share/man/man\fR* directories. The \fBSGML\fR sources are located in
the \fB/usr/share/man/sman\fR* directories. 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 \fB/usr/share/man\fR
from a host on which they do reside.
.sp
.LP
If there are preformatted, up-to-date versions in the corresponding \fBcat\fR*
or \fBfmt\fR* directories, \fBman\fR simply displays or prints those versions.
If the preformatted version of interest is out of date or missing, \fBman\fR
reformats it prior to display and stores the preformatted version if \fBcat\fR*
or \fBfmt\fR* is writable. The \fBwindex\fR database is not updated. See
\fBcatman\fR(1M). If directories for the preformatted versions are not
provided, \fBman\fR reformats a page whenever it is requested. \fBman\fR uses a
temporary file to store the formatted text during display.
.sp
```



```
.LP
If the standard output is not a terminal, or if the '\fB-\fR' flag is given,
\fBman\fR pipes its output through \fBcat\fR(1). Otherwise, \fBman\fR pipes its
output through \fBmore\fR(1) to handle paging and underlining on the screen.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-a\fR\fR
.ad
.RS 20n
Shows all manual pages matching \fIname\fR within the \fBMANPATH\fR search
path. Manual pages are displayed in the order found.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR\fR
.ad
.RS 20n
Debugs. Displays what a section-specifier evaluates to, method used for
searching, and paths searched by \fBman\fR.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIfile ...\fR\fR
.ad
.RS 20n
\fBman\fR attempts to locate manual pages related to any of the given
\fIfile\fRs. It strips the leading path name components from each \fIfile\fR,
and then prints one-line summaries containing the resulting basename or names.
This option also uses the \fBwindex\fR database.
.RE

.sp
.ne 2
.na
\fB\fB-F\fR\fR
.ad
.RS 20n
Forces \fBman\fR to search all directories specified by \fBMANPATH\fR or the
\fBman.cf\fR file, rather than using the \fBwindex\fR lookup database. This
option is useful if the database is not up to date and it has been made the
default behavior of the \fBman\fR command. The option therefore does not have
to be invoked and is documented here for reference only.
.RE

.sp
.ne 2
.na
\fB\fB-k\fR \fIkeyword ...\fR\fR
.ad
.RS 20n
Prints out one-line summaries from the \fBwindex\fR database (table of
contents) that contain any of the given \fIkeyword\fRs. The \fBwindex\fR
database is created using \fBcatman\fR(1M).
.RE

.sp
.ne 2
```

```
.na
\fB\fB-l\fR\fR
.ad
.RS 20n
Lists all manual pages found matching \fIname\fR within the search path.
.RE

.sp
.ne 2
.na
\fB\fB-M\fR \fIpath\fR\fR
.ad
.RS 20n
Specifies an alternate search path for manual pages. \fIpath\fR is a
colon-separated list of directories that contain manual page directory
subtrees. For example, if \fIpath\fR is \fB/usr/share/man:/usr/local/man\fR,
\fBman\fR searches for \fIname\fR in the standard location, and then
\fB/usr/local/man\fR. When used with the \fB-k\fR or \fB-f\fR options, the
\fB-M\fR option must appear first. Each directory in the \fIpath\fR is assumed
to contain subdirectories of the form \fBman\fR* or \fBsman\fR* , one for each
section. This option overrides the \fBMANPATH\fR environment variable.
.RE

.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.RS 20n
Reformats the manual page, but does not display it. This replaces the \fBman\fR
\fB-\fR \fB-t\fR \fIname\fR combination.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIsection ...\fR\fR
.ad
.RS 20n
Specifies sections of the manual for \fBman\fR to search. The directories
searched for \fIname\fR are limited to those specified by \fIsection\fR.
\fIsection\fR can be a numerical digit, perhaps followed by one or more letters
to match the desired section of the manual, for example, "\fB3libucb\fR". Also,
\fIsection\fR can be a word, for example, \fBlocal\fR, \fBnew\fR, \fBold\fR,
\fBpublic\fR. \fIsection\fR can also be a letter. To specify multiple sections,
separate each section with a comma. This option overrides the \fBMANPATH\fR
environment variable and the \fBman.cf\fR file. See \fBSearch\fR \fBPath\fR
below for an explanation of how \fBman\fR conducts its search.
.RE

.sp
.ne 2
.na
\fB\fB-t\fR\fR
.ad
.RS 20n
\fBman\fR arranges for the specified manual pages to be \fBtroff\fRed to a
suitable raster output device (see \fBtroff\fR(1)). If both the \fB-\fR and
\fB-t\fR flags are given, \fBman\fR updates the \fBtroff\fRed versions of each
named \fIname\fR (if necessary), but does not display them.
.RE

.sp
.ne 2
.na
\fB\fB-T\fR \fImacro-package\fR\fR
```



```
.ad
.RS 20n
Formats manual pages using \fImacro-package\fR rather than the standard
\fB-man\fR macros defined in \fB/usr/share/lib/tmac/an\fR. See \fBSearch
Path\fR under USAGE for a complete explanation of the default search path
order.
.RE

.SH OPERANDS
.sp
.LP
The following operand is supported:
.sp
.ne 2
.na
\fB\fIname\fR\fR
.ad
.RS 8n
The name of a standard utility or a keyword.
.RE

.SH USAGE
.sp
.LP
The usage of \fBman\fR is described below:
.SS "Manual Page Sections"
.sp
.LP
Entries in the reference manuals are organized into \fIsection\fRs. 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, "\fB9\fR", does not act as an abbreviation for
the subsections of that name, such as "\fB9e\fR", "\fB9f\fR", or "\fB9s\fR".
That is, each subsection must be searched separately by \fBman\fR \fB-s\fR.
Each section contains descriptions apropos to a particular reference category,
with subsections refining these distinctions. See the \fBintro\fR manual pages
for an explanation of the classification used in this release.
.SS "Search Path"
.sp
.LP
Before searching for a given \fIname\fR, \fBman\fR constructs a list of
candidate directories and sections. \fBman\fR searches for \fIname\fR in the
directories specified by the \fBMANPATH\fR environment variable.
.sp
.LP
In the absence of \fBMANPATH\fR, \fBman\fR constructs its search path based
upon the \fBPATH\fR environment variable, primarily by substituting \fBman\fR
for the last component of the \fBPATH\fR element. Special provisions are added
to account for unique characteristics of directories such as \fB/sbin\fR,
\fB/usr/ucb\fR, \fB/usr/xpg4/bin\fR, and others. If the file argument contains
a \fB/\fR character, the \fIdirname\fR portion of the argument is used in place
of \fBPATH\fR elements to construct the search path.
.sp
.LP
Within the manual page directories, \fBman\fR confines its search to the
sections specified in the following order:
.RS +4
.TP
.ie t \(bu
.el o
\fIsection\fRs specified on the command line with the \fB-s\fR option
.RE
.RS +4
.TP
.ie t \(bu
.el o
```

```
\fIsection\fRs embedded in the \fBMANPATH\fR environment variable
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fIsection\fRs specified in the \fBman.cf\fR file for each directory specified
in the \fBMANPATH\fR environment variable
.RE
.sp
.LP
If none of the above exist, \fBman\fR searches each directory in the manual
page path, and displays the first matching manual page found.
.sp
.LP
The \fBman.cf\fR file has the following format:
.sp
.in +2
.nf
MANSECTS=\fIsection\fR[,\fIsection\fR]...
.fi
.in -2
.sp

.sp
.LP
Lines beginning with ‘\fB#\fR’ and blank lines are considered comments, and are
ignored. Each directory specified in \fBMANPATH\fR can contain a manual page
configuration file, specifying the default search order for that directory.
.SH FORMATTING MANUAL PAGES
.sp
.LP
Manual pages are marked up in \fBnroff\fR(1) or \fBsgml\fR(5). Nroff manual
pages are processed by \fBnroff\fR(1) or \fBtroff\fR(1) with the \fB-man\fR
macro package. Please refer to \fBman\fR(5) for information on macro usage.
\fBSGML\fR\(emtagged manual pages are processed by an \fBSGML\fR parser and
passed to the formatter.
.SS "Preprocessing Nroff Manual Pages"
.sp
.LP
When formatting an nroff manual page, \fBman\fR examines the first line to
determine whether it requires special processing. If the first line is a string
of the form:
.sp
.in +2
.nf
\&’\e" \fIX\fR
.fi
.in -2
.sp

.sp
.LP
where \fIX\fR is separated from the ‘\fB"\fR’ by a single SPACE and consists of
any combination of characters in the following list, \fBman\fR pipes its input
to \fBtroff\fR(1) or \fBnroff\fR(1) through the corresponding preprocessors.
.sp
.ne 2
.na
\fB\fBe\fR\fR
.ad
.RS 5n
\fBeqn\fR(1), or \fBneqn\fR for \fBnroff\fR
.RE

.sp
```



```
.ne 2
.na
\fB\fBr\fR\fR
.ad
.RS 5n
\fBrefer\fR(1)
.RE

.sp
.ne 2
.na
\fB\fBt\fR\fR
.ad
.RS 5n
\fBtbl\fR(1)
.RE

.sp
.ne 2
.na
\fB\fBv\fR\fR
.ad
.RS 5n
\fBvgrind\fR(1)
.RE

.sp
.LP
If \fBeqn\fR or \fBneqn\fR is invoked, it automatically reads the file
\fB/usr/pub/eqnchar\fR (see \fBeqnchar\fR(5)). If \fBnroff\fR(1) is invoked,
\fBcol\fR(1) is automatically used.
.SS "Referring to Other nroff Manual Pages"
.sp
.LP
If the first line of the nroff manual page is a reference to another manual
page entry fitting the pattern:
.sp
.in +2
.nf
\&.so man*/\fIsourcefile\fR
.fi
.in -2
.sp

.sp
.LP
\fBman\fR processes the indicated file in place of the current one. The
reference must be expressed as a path name relative to the root of the manual
page directory subtree.
.sp
.LP
When the second or any subsequent line starts with \fB\&.so\fR, \fBman\fR
ignores it; \fBtroff\fR(1) or \fBnroff\fR(1) processes the request in the usual
manner.
.SS "Processing SGML Manual Pages"
.sp
.LP
Manual pages are identified as being marked up in SGML by the presence of the
string \fB<!DOCTYPE\fR\&. If the file also contains the string
\fBSHADOW_PAGE\fR, the file refers to another manual page for the content. The
reference is made with a file entity reference to the manual page that contains
the text. This is similar to the \fB\&.so\fR mechanism used in the nroff
formatted man pages.
.SH ENVIRONMENT VARIABLES
.sp
.LP
```

```
390 See \fBenviron\fR(5) for descriptions of the following environment variables
391 that affect the execution of \fBman\fR: \fBLANG\fR, \fBLC_ALL\fR,
392 \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
393 .sp
394 .ne 2
395 .na
396 \fB\fBMANPATH\fR\fR
397 .ad
398 .RS 11n
399 A colon-separated list of directories; each directory can be followed by a
400 comma-separated list of sections. If set, its value overrides
401 \fB/usr/share/man\fR as the default directory search path, and the \fBman.cf\fR
402 file as the default section search path. The \fB-M\fR and \fB-s\fR flags, in
403 turn, override these values.)
404 .RE

406 .sp
407 .ne 2
408 .na
409 \fB\fBPAGER\fR\fR
410 .ad
411 .RS 11n
412 A program to use for interactively delivering \fBman\fR's output to the screen.
413 If not set, '\fBmore\fR \fB-s\fR' is used. See \fBmore\fR(1).
414 .RE

416 .sp
417 .ne 2
418 .na
419 \fB\fBTCAT\fR\fR
420 .ad
421 .RS 11n
422 The name of the program to use to display \fBtroff\fRed manual pages.
423 .RE

425 .sp
426 .ne 2
427 .na
428 \fB\fBTROFF\fR\fR
429 .ad
430 .RS 11n
431 The name of the formatter to use when the \fB-t\fR flag is given. If not set,
432 \fBtroff\fR(1) is used.
433 .RE

435 .SH EXAMPLES
436 .LP
437 \fBExample 1 \fRCreating a PostScript Version of a man page
438 .sp
439 .LP
440 The following example creates the \fBpipe\fR(2) man page in postscript for csh,
441 tcsh, ksh and sh users:

443 .sp
444 .in +2
445 .nf
446         % env TCAT=/usr/lib/lp/postscript/dpost man -t -s 2 pipe > pipe.ps
447 .fi
448 .in -2
449 .sp

451 .sp
452 .LP
453 This is an alternative to using \fBman\fR \fB-t\fR, which sends the man page to
454 the default printer, if the user wants a postscript file version of the man
455 page.
```



```
457 .LP
458 \fBExample 2 \fRCreating a Text Version of a man page
459 .sp
460 .LP
461 The following example creates the \fBpipe\fR(2) man page in ascii text:

463 .sp
464 .in +2
465 .nf
466 man pipe.2 | col -x -b > pipe.text
467 .fi
468 .in -2
469 .sp

471 .sp
472 .LP
473 This is an alternative to using \fBman\fR \fB-t\fR, which sends the man page to
474 the default printer, if the user wants a text file version of the man page.

476 .SH EXIT STATUS
477 .sp
478 .LP
479 The following exit values are returned:
480 .sp
481 .ne 2
482 .na
483 \fB\fB0\fR\fR
484 .ad
485 .RS 6n
486 Successful completion.
487 .RE

489 .sp
490 .ne 2
491 .na
492 \fB\fB>0\fR\fR
493 .ad
494 .RS 6n
495 An error occurred.
496 .RE

498 .SH FILES
499 .sp
500 .ne 2
501 .na
502 \fB\fB/usr/share/man\fR\fR
503 .ad
504 .sp .6
505 .RS 4n
506 Root of the standard manual page directory subtree
507 .RE

509 .sp
510 .ne 2
511 .na
512 \fB\fB/usr/share/man/man?/*\fR\fR
513 .ad
514 .sp .6
515 .RS 4n
516 Unformatted nroff manual entries
517 .RE

519 .sp
520 .ne 2
521 .na
```

```
\fB\fB/usr/share/man/sman?/*\fR\fR
.ad
.sp .6
.RS 4n
Unformatted \fBSGML\fR manual entries
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/man/cat?/*\fR\fR
.ad
.sp .6
.RS 4n
\fBnroff\fRed manual entries
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/man/fmt?/*\fR\fR
.ad
.sp .6
.RS 4n
\fBtroff\fRed manual entries
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/man/windex\fR\fR
.ad
.sp .6
.RS 4n
Table of contents and keyword database
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/lib/tmac/an\fR\fR
.ad
.sp .6
.RS 4n
Standard \fB-man\fR macro package
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/lib/sgml/locale/C/dtd/*\fR\fR
.ad
.sp .6
.RS 4n
\fBSGML\fR document type definition files
.RE

.sp
.ne 2
.na
\fB\fB/usr/share/lib/sgml/locale/C/solbook/*\fR\fR
.ad
.sp .6
.RS 4n
\fBSGML\fR style sheet and entity definitions directories
.RE
```



```
.sp
.ne 2
.na
\fB\fB/usr/share/lib/pub/eqnchar\fR\fR
.ad
.sp .6
.RS 4n
Standard definitions for \fBeqn\fR and \fBneqn\fR
.RE

.sp
.ne 2
.na
\fB\fBman.cf\fR\fR
.ad
.sp .6
.RS 4n
Default search order by section
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled, see \fBNOTES\fR.
_
Interface Stability     Committed
_
Standard        See \fBstandards\fR(5).
.TE

.SH SEE ALSO
.sp
.LP
\fBapropos\fR(1), \fBcat\fR(1), \fBcol\fR(1), \fBdpost\fR(1), \fBeqn\fR(1),
\fBmore\fR(1), \fBnroff\fR(1), \fBrefer\fR(1), \fBtbl\fR(1), \fBtroff\fR(1),
\fBvgrind\fR(1), \fBwhatis\fR(1), \fBcatman\fR(1M), \fBattributes\fR(5),
\fBenviron\fR(5), \fBeqnchar\fR(5), \fBman\fR(5), \fBsgml\fR(5),
\fBstandards\fR(5)
.SH NOTES
.sp
.LP
The \fB-f\fR and \fB-k\fR options use the \fBwindex\fR database, which is
created by \fBcatman\fR(1M).
.sp
.LP
The \fBman\fR command is CSI-capable. However, some utilities invoked by the
\fBman\fR command, namely, \fBtroff\fR, \fBeqn\fR, \fBneqn\fR, \fBrefer\fR,
\fBtbl\fR, and \fBvgrind\fR, are not verified to be CSI-capable. Because of
this, the man command with the \fB-t\fR option can not handle non-EUC data.
Also, using the \fBman\fR command to display man pages that require special
processing through \fBeqn\fR, \fBneqn\fR, \fBrefer\fR, \fBtbl\fR, or
\fBvgrind\fR can not be CSI-capable.
.SH BUGS
.sp
.LP
```

```
654 The manual is supposed to be reproducible either on a phototypesetter or on an
655 \fBASCII\fR terminal. However, on a terminal some information (indicated by
656 font changes, for instance) is lost.
657 .sp
658 .LP
659 Some dumb terminals cannot process the vertical motions produced by the \fBe\fR
660 (see \fBeqn\fR(1)) preprocessing flag. To prevent garbled output on these
661 terminals, when you use \fBe\fR, also use \fBt\fR, to invoke \fBcol\fR(1)
662 implicitly. This workaround has the disadvantage of eliminating superscripts
663 and subscripts, even on those terminals that can display them. Control-q clears
664 a terminal that gets confused by \fBeqn\fR(1) output.
```

```
**********************************************************
   14740 Tue Jul 15 13:48:09 2014
new/usr/src/man/man1/mandoc.1
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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 13, 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 ,
respectively.
The
.Xr mdoc 5
```

```
format is
.Em strongly
recommended;
.Xr man 5
should only be used for legacy manuals.
.Pp
A third option,
.Fl m Ns Cm andoc ,
which is also the default, determines encoding on-the-fly: if the first
non-comment macro is
.Sq \&Dd
or
.Sq \&Dt ,
the
.Xr mdoc 5
parser is used; otherwise, the
.Xr man 5
parser is used.
.Pp
If multiple
files are specified with
.Fl m Ns Cm andoc ,
each has its file-type determined this way.
If multiple files are
specified and
.Fl m Ns Cm doc
or
.Fl m Ns Cm an
is specified, then this format is used exclusively.
.Ss Output Formats
The
.Nm
utility accepts the following
.Fl T
arguments, which correspond to output modes:
.Bl -tag -width "-Tlocale"
.It Fl T Ns Cm ascii
Produce 7-bit ASCII output.
This is the default.
See
.Sx ASCII Output .
.It Fl T Ns Cm html
Produce strict CSS1/HTML-4.01 output.
See
.Sx HTML Output .
.It Fl T Ns Cm lint
Parse only: produce no output.
Implies
.Fl W Ns Cm warning .
.It Fl T Ns Cm locale
Encode output using the current locale.
See
.Sx Locale Output .
.It Fl T Ns Cm man
Produce
.Xr man 5
format output.
See
.Sx Man Output .
.It Fl T Ns Cm pdf
Produce PDF output.
See
.Sx PDF Output .
.It Fl T Ns Cm ps
Produce PostScript output.
See
```



```
.Sx PostScript Output .
.It Fl T Ns Cm tree
Produce an indented parse tree.
.It Fl T Ns Cm utf8
Encode output in the UTF\-8 multi-byte format.
See
.Sx UTF\-8 Output .
.It Fl T Ns Cm xhtml
Produce strict CSS1/XHTML-1.0 output.
See
.Sx XHTML Output .
.El
.Pp
If multiple input files are specified, these will be processed by the
corresponding filter in-order.
.Ss ASCII Output
Output produced by
.Fl T Ns Cm ascii ,
which is the default, is rendered in standard 7-bit ASCII documented in
.Xr ascii 5 .
.Pp
Font styles are applied by using back-spaced encoding such that an
underlined character
.Sq c
is rendered as
.Sq _ Ns \e[bs] Ns c ,
where
.Sq \e[bs]
is the back-space character number 8.
Emboldened characters are rendered as
.Sq c Ns \e[bs] Ns c .
.Pp
The special characters documented in
.Xr mandoc_char 5
are rendered best-effort in an ASCII equivalent.
If no equivalent is found,
.Sq \&?
is used instead.
.Pp
Output width is limited to 78 visible columns unless literal input lines
exceed this limit.
.Pp
The following
.Fl O
arguments are accepted:
.Bl -tag -width Ds
.It Cm indent Ns = Ns Ar indent
The left margin for normal text is set to
.Ar indent
blank characters instead of the default of five for
.Xr mdoc 5
and seven for
.Xr man 5 .
Increasing this is not recommended; it may result in degraded formatting,
for example overfull lines or ugly line breaks.
.It Cm width Ns = Ns Ar width
The output width is set to
.Ar width ,
which will normalise to \(>=60.
.El
.Ss HTML Output
Output produced by
.Fl T Ns Cm html
conforms to HTML-4.01 strict.
.Pp
The
```

```
.Pa example.style.css
file documents style-sheet classes available for customising output.
If a style-sheet is not specified with
.Fl O Ns Ar style ,
.Fl T Ns Cm html
defaults to simple output readable in any graphical or text-based web
browser.
.Pp
Special characters are rendered in decimal-encoded UTF\-8.
.Pp
The following
.Fl O
arguments are accepted:
.Bl -tag -width Ds
.It Cm fragment
Omit the
.Aq !DOCTYPE
declaration and the
.Aq html ,
.Aq head ,
and
.Aq body
elements and only emit the subtree below the
.Aq body
element.
The
.Cm style
argument will be ignored.
This is useful when embedding manual content within existing documents.
.It Cm includes Ns = Ns Ar fmt
The string
.Ar fmt ,
for example,
.Ar ../src/%I.html ,
is used as a template for linked header files (usually via the
.Sq \&In
macro).
Instances of
.Sq \&%I
are replaced with the include filename.
The default is not to present a
hyperlink.
.It Cm man Ns = Ns Ar fmt
The string
.Ar fmt ,
for example,
.Ar ../html%S/%N.%S.html ,
is used as a template for linked manuals (usually via the
.Sq \&Xr
macro).
Instances of
.Sq \&%N
and
.Sq %S
are replaced with the linked manual’s name and section, respectively.
If no section is included, section 1 is assumed.
The default is not to
present a hyperlink.
.It Cm style Ns = Ns Ar style.css
The file
.Ar style.css
is used for an external style-sheet.
This must be a valid absolute or
relative URI.
.El
.Ss Locale Output
```



```
Locale-depending output encoding is triggered with
.Fl T Ns Cm locale .
This option is not available on all systems: systems without locale
support, or those whose internal representation is not natively UCS-4,
will fall back to
.Fl T Ns Cm ascii .
See
.Sx ASCII Output
for font style specification and available command-line arguments.
.Ss Man Output
Translate input format into
.Xr man 5
output format.
This is useful for distributing manual sources to legacy systems
lacking
.Xr mdoc 5
formatters.
.Pp
If
.Xr mdoc 5
is passed as input, it is translated into
.Xr man 5 .
If the input format is
.Xr man 5 ,
the input is copied to the output, expanding any
.Xr mandoc_roff 5
.Sq so
requests.
The parser is also run, and as usual, the
.Fl W
level controls which
.Sx DIAGNOSTICS
are displayed before copying the input to the output.
.Ss PDF Output
PDF-1.1 output may be generated by
.Fl T Ns Cm pdf .
See
.Sx PostScript Output
for
.Fl O
arguments and defaults.
.Ss PostScript Output
PostScript
.Qq Adobe-3.0
Level-2 pages may be generated by
.Fl T Ns Cm ps .
Output pages default to letter sized and are rendered in the Times font
family, 11-point.
Margins are calculated as 1/9 the page length and width.
Line-height is 1.4m.
.Pp
Special characters are rendered as in
.Sx ASCII Output .
.Pp
The following
.Fl O
arguments are accepted:
.Bl -tag -width Ds
.It Cm paper Ns = Ns Ar name
The paper size
.Ar name
may be one of
.Ar a3 ,
.Ar a4 ,
.Ar a5 ,
.Ar legal ,
```

```
392 or
393 .Ar letter .
394 You may also manually specify dimensions as
395 .Ar NNxNN ,
396 width by height in millimetres.
397 If an unknown value is encountered,
398 .Ar letter
399 is used.
400 .El
401 .Ss UTF\-8 Output
402 Use
403 .Fl T Ns Cm utf8
404 to force a UTF\-8 locale.
405 See
406 .Sx Locale Output
407 for details and options.
408 .Ss XHTML Output
409 Output produced by
410 .Fl T Ns Cm xhtml
411 conforms to XHTML-1.0 strict.
412 .Pp
413 See
414 .Sx HTML Output
415 for details; beyond generating XHTML tags instead of HTML tags, these
416 output modes are identical.
417 .Sh EXIT STATUS
418 The
419 .Nm
420 utility exits with one of the following values, controlled by the message
421 .Ar level
422 associated with the
423 .Fl W
424 option:
425 .Pp
426 .Bl -tag -width Ds -compact
427 .It 0
428 No warnings or errors occurred, or those that did were ignored because
429 they were lower than the requested
430 .Ar level .
431 .It 2
432 At least one warning occurred, but no error, and
433 .Fl W Ns Cm warning
434 was specified.
435 .It 3
436 At least one parsing error occurred, but no fatal error, and
437 .Fl W Ns Cm error
438 or
439 .Fl W Ns Cm warning
440 was specified.
441 .It 4
442 A fatal parsing error occurred.
443 .It 5
444 Invalid command line arguments were specified.
445 No input files have been read.
446 .It 6
447 An operating system error occurred, for example memory exhaustion or an
448 error accessing input files.
449 Such errors cause
450 .Nm
451 to exit at once, possibly in the middle of parsing or formatting a file.
452 .El
453 .Pp
454 Note that selecting
455 .Fl T Ns Cm lint
456 output mode implies
457 .Fl W Ns Cm warning .
```



```
458 .Sh EXAMPLES
459 To page manuals to the terminal:
460 .Pp
461 .Dl $ mandoc \-Wall,stop mandoc.1 2\*(Gt&1 | less
462 .Dl $ mandoc mandoc.1 mdoc.3 mdoc.7 | less
463 .Pp
464 To produce HTML manuals with
465 .Ar style.css
466 as the style-sheet:
467 .Pp
468 .Dl $ mandoc \-Thtml -Ostyle=style.css mdoc.7 \*(Gt mdoc.7.html
469 .Pp
470 To check over a large set of manuals:
471 .Pp
472 .Dl $ mandoc \-Tlint ‘find /usr/src -name \e*\e.[1-9]‘
473 .Pp
474 To produce a series of PostScript manuals for A4 paper:
475 .Pp
476 .Dl $ mandoc \-Tps \-Opaper=a4 mdoc.7 man.7 \*(Gt manuals.ps
477 .Pp
478 Convert a modern
479 .Xr mdoc 5
480 manual to the older
481 .Xr man 5
482 format, for use on systems lacking an
483 .Xr mdoc 5
484 parser:
485 .Pp
486 .Dl $ mandoc \-Tman foo.mdoc \*(Gt foo.man
487 .Sh DIAGNOSTICS
488 Standard error messages reporting parsing errors are prefixed by
489 .Pp
490 .Sm off
491 .D1 Ar file : line : column : \ level :
492 .Sm on
493 .Pp
494 where the fields have the following meanings:
495 .Bl -tag -width "column"
496 .It Ar file
497 The name of the input file causing the message.
498 .It Ar line
499 The line number in that input file.
500 Line numbering starts at 1.
501 .It Ar column
502 The column number in that input file.
503 Column numbering starts at 1.
504 If the issue is caused by a word, the column number usually
505 points to the first character of the word.
506 .It Ar level
507 The message level, printed in capital letters.
508 .El
509 .Pp
510 Message levels have the following meanings:
511 .Bl -tag -width "warning"
512 .It Cm fatal
513 The parser is unable to parse a given input file at all.
514 No formatted output is produced from that input file.
515 .It Cm error
516 An input file contains syntax that cannot be safely interpreted,
517 either because it is invalid or because
518 .Nm
519 does not implement it yet.
520 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 Committed, but the details of specific output formats other than
635 ASCII are Uncommitted.
636 .Sh SEE ALSO
637 .Xr eqn 5 ,
638 .Xr man 5 ,
639 .Xr mandoc_char 5 ,
640 .Xr mdoc 5 ,
641 .Xr mandoc_roff 5 ,
642 .Xr tbl 5
643 .Sh AUTHORS
644 The
645 .Nm
646 utility was written by
647 .An Kristaps Dzonsons ,
648 .Mt kristaps@bsd.lv .
649 .Sh CAVEATS
650 In
651 .Fl T Ns Cm html
652 and
653 .Fl T Ns Cm xhtml ,
654 the maximum size of an element attribute is determined by
655 .Dv BUFSIZ ,
```

```
656 which is usually 1024 bytes.
657 Be aware of this when setting long link
658 formats such as
659 .Fl O Ns Cm style Ns = Ns Ar really/long/link .
660 .Pp
661 Nesting elements within next-line element scopes of
662 .Fl m Ns Cm an ,
663 such as
664 .Sq br
665 within an empty
666 .Sq B ,
667 will confuse
668 .Fl T Ns Cm html
669 and
670 .Fl T Ns Cm xhtml
671 and cause them to forget the formatting of the prior next-line scope.
672 .Pp
673 The
674 .Sq \(aq
675 control character is an alias for the standard macro control character
676 and does not emit a line-break as stipulated in GNU troff.
```

```
**********************************************************
    1724 Tue Jul 15 13:48:09 2014
new/usr/src/man/man1/whatis.1
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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>
.\"
.Dd Jul 13, 2014
.Dt APROPOS 1
.Os
.Sh NAME
.Nm apropos ,
.Nm whatis
.Nd keyword search in
.Nm whatis
database files
.Sh SYNOPSIS
.Nm
.Op Fl M Ar path
.Op Fl s Ar section
.Ar keyword ...
.Nm whatis
.Op Fl M Ar path
.Op Fl s Ar section
.Ar keyword ...
.Sh DESCRIPTION
The
.Nm
utility searches a set of
.Nm whatis
database files matching each
.Ar keyword .
The
.Nm whatis
utility does the same search but only on complete words. The
.Nm whatis
database files are created using
.Xr man 1
command.
.Bl -tag -width ".Fl d"
.It Fl M Ar path
Force a specific colon separated manual path instead of the default search path.
Overrides the
.Ev MANPATH
environment variable.
.It Fl s Ar section
Restrict search to specified
.Ar section .
.El
.Sh ENVIRONMENT
The following environment variables affect the execution of
.Nm :
.Bl -tag -width ".Ev MANPATH , PATH"
.It Ev MANPATH , PATH
```



```
Used to find the location of the
.Nm whatis
database files.
.El
.Sh DIAGNOSTICS
The
.Nm
utility exits 0 if a keyword matched and 1 if no keywords are matched or no
.Nm whatis
databases are found.
.Sh INTERFACE STABILITY
Committed.
.Sh CODE SET INDEPENDENCE
Enabled.
.Sh SEE ALSO
.Xr man 1 ,
.Xr mandoc 1
'\" te
.\" Copyright (c) 1992, 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
.TH WHATIS 1 "Sep 14, 1992"
.SH NAME
whatis \- display a one-line summary about a keyword
.SH SYNOPSIS
.LP
.nf
\fBwhatis\fR \fIcommand\fR...
.fi

.SH DESCRIPTION
.sp
.LP
\fBwhatis\fR looks up a given \fIcommand\fR and displays the header line from
the manual section. You can then run the \fBman\fR(1) command to get more
information. If the line starts \fBname(\fIsection\fR)\fR.\|.\|. you can do
\fBman\fR \fB-s\fR\fB\fIsection\fR name\fR to get the documentation for it.
Try \fBwhatis ed\fR and then you should do \fBman\fR \fB-s\fR \fB1 ed\fR to get
the manual page for \fBed\fR(1).
.sp
.LP
\fBwhatis\fR is actually just the \fB-f\fR option to the \fBman\fR(1) command.
.sp
.LP
\fBwhatis\fR uses the \fB/usr/share/man/windex\fR database.  This database is
created by \fBcatman\fR(1M). If this database does not exist, \fBwhatis\fR will
fail.
.SH FILES
.sp
.ne 2
.na
\fB\fB/usr/share/man/windex\fR\fR
.ad
.RS 25n
Table of contents and keyword database
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
```

```
 50 box;
 51 c | c
 52 l | l .
 53 ATTRIBUTE TYPE  ATTRIBUTE VALUE
 54 _
 55 CSI     Enabled
 56 .TE

 58 .SH SEE ALSO
 59 .sp
 60 .LP
 61 \fBapropos\fR(1), \fBman\fR(1), \fBcatman\fR(1M), \fBattributes\fR(5)
```

```
**********************************************************
   15004 Tue Jul 15 13:48:10 2014
new/usr/src/man/man1m/Makefile
import complete (hopefully)
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet
   9 # at http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright 2011, Richard Lowe
  14 # Copyright (c) 2012, Joyent, Inc.  All rights reserved.
  15 # Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
  16 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  17 #

  19 include         $(SRC)//Makefile.master

  21 MANSECT=        1m

  23 _MANFILES=      6to4relay.1m            \
  24                 Intro.1m                \
  25                 Uutry.1m                \
  26                 accept.1m               \
  27                 acct.1m                 \
  28                 acctadm.1m              \
  29                 acctcms.1m              \
  30                 acctcon.1m              \
  31                 acctmerg.1m             \
  32                 acctprc.1m              \
  33                 acctsh.1m               \
  34                 adbgen.1m               \
  35                 add_allocatable.1m      \
  36                 add_drv.1m              \
  37                 addbadsec.1m            \
  38                 arcstat.1m              \
  39                 arp.1m                  \
  40                 atohexlabel.1m          \
  41                 audit.1m                \
  42                 audit_warn.1m           \
  43                 auditconfig.1m          \
  44                 auditd.1m               \
  45                 auditrecord.1m          \
  46                 auditreduce.1m          \
  47                 auditstat.1m            \
  48                 automount.1m            \
  49                 automountd.1m           \
  50                 autopush.1m             \
  51                 bart.1m                 \
  52                 beadm.1m                \
  53                 boot.1m                 \
  54                 bootadm.1m              \
  55                 bootconfchk.1m          \
  56                 busstat.1m              \
  57                 cachefsd.1m             \
  58                 cachefslog.1m           \
  59                 cachefspack.1m          \
  60                 cachefsstat.1m          \
  61                 cachefswssize.1m        \
```



```
  62                 captoinfo.1m            \
  62                 catman.1m               \
  63                 cfgadm.1m               \
  64                 cfgadm_ac.1m            \
  65                 cfgadm_cardbus.1m       \
  66                 cfgadm_fp.1m            \
  67                 cfgadm_ib.1m            \
  68                 cfgadm_pci.1m           \
  69                 cfgadm_sata.1m          \
  70                 cfgadm_sbd.1m           \
  71                 cfgadm_scsi.1m          \
  72                 cfgadm_sdcard.1m        \
  73                 cfgadm_sysctrl.1m       \
  74                 cfgadm_usb.1m           \
  75                 cfsadmin.1m             \
  76                 chat.1m                 \
  77                 check-hostname.1m       \
  78                 check-permissions.1m    \
  79                 chroot.1m               \
  80                 clear_locks.1m          \
  81                 clinfo.1m               \
  82                 clri.1m                 \
  83                 consadm.1m              \
  84                 conv_lp.1m              \
  85                 conv_lpd.1m             \
  86                 coreadm.1m              \
  87                 cpustat.1m              \
  88                 cron.1m                 \
  89                 cryptoadm.1m            \
  90                 datadm.1m               \
  91                 dd.1m                   \
  92                 devattr.1m              \
  93                 devfree.1m              \
  94                 devfsadm.1m             \
  95                 device_remap.1m         \
  96                 devinfo.1m              \
  97                 devlinks.1m             \
  98                 devnm.1m                \
  99                 devprop.1m              \
 100                 devreserv.1m            \
 101                 df.1m                   \
 102                 df_ufs.1m               \
 103                 dfmounts.1m             \
 104                 dfmounts_nfs.1m         \
 105                 dfshares.1m             \
 106                 dfshares_nfs.1m         \
 107                 dhcpagent.1m            \
 108                 dhcpconfig.1m           \
 109                 dhcpmgr.1m              \
 110                 dhtadm.1m               \
 111                 disks.1m                \
 112                 diskscan.1m             \
 113                 dispadmin.1m            \
 114                 dladm.1m                \
 115                 dlmgmtd.1m              \
 116                 dmesg.1m                \
 117                 dminfo.1m               \
 118                 dns-sd.1m               \
 119                 domainname.1m           \
 120                 drvconfig.1m            \
 121                 dsbitmap.1m             \
 122                 dscfg.1m                \
 123                 dscfgadm.1m             \
 124                 dscfglockd.1m           \
 125                 dsstat.1m               \
 126                 dsvclockd.1m            \
```

```
127                 dtrace.1m               \
128                 dumpadm.1m              \
129                 editmap.1m              \
130                 edquota.1m              \
131                 eeprom.1m               \
132                 embedded_su.1m          \
133                 etrn.1m                 \
134                 fcinfo.1m               \
135                 fdetach.1m              \
136                 fdisk.1m                \
137                 ff.1m                   \
138                 ff_ufs.1m               \
139                 fiocompress.1m          \
140                 flowadm.1m              \
141                 fmadm.1m                \
142                 fmd.1m                  \
143                 fmdump.1m               \
144                 fmstat.1m               \
145                 fmthard.1m              \
146                 format.1m               \
147                 fruadm.1m               \
148                 fsck.1m                 \
149                 fsck_cachefs.1m         \
150                 fsck_pcfs.1m            \
151                 fsck_udfs.1m            \
152                 fsck_ufs.1m             \
153                 fsdb.1m                 \
154                 fsdb_udfs.1m            \
155                 fsdb_ufs.1m             \
156                 fsirand.1m              \
157                 fssnap.1m               \
158                 fssnap_ufs.1m           \
159                 fsstat.1m               \
160                 fstyp.1m                \
161                 ftpaddhost.1m           \
162                 ftpconfig.1m            \
163                 ftprestart.1m           \
164                 ftpshut.1m              \
165                 fuser.1m                \
166                 fwflash.1m              \
167                 fwtmp.1m                \
168                 getdev.1m               \
169                 getdevpolicy.1m         \
170                 getdgrp.1m              \
171                 getent.1m               \
172                 gettable.1m             \
173                 getty.1m                \
174                 getvol.1m               \
175                 groupadd.1m             \
176                 groupdel.1m             \
177                 groupmod.1m             \
178                 growfs.1m               \
179                 gsscred.1m              \
180                 gssd.1m                 \
181                 hal-device.1m           \
182                 hal-fdi-validate.1m     \
183                 hal-find.1m             \
184                 hal-get-property.1m     \
185                 hald.1m                 \
186                 halt.1m                 \
187                 hextoalabel.1m          \
188                 hostconfig.1m           \
189                 htable.1m               \
190                 ickey.1m                \
191                 id.1m                   \
192                 idmap.1m                \
```



```
193                 idmapd.1m               \
194                 idsconfig.1m            \
195                 if_mpadm.1m             \
196                 ifconfig.1m             \
197                 ifparse.1m              \
198                 iiadm.1m                \
199                 iicpbmp.1m              \
200                 iicpshd.1m              \
201                 ikeadm.1m               \
202                 ikecert.1m              \
203                 in.chargend.1m          \
204                 in.comsat.1m            \
205                 in.daytimed.1m          \
206                 in.dhcpd.1m             \
207                 in.discardd.1m          \
208                 in.echod.1m             \
209                 in.fingerd.1m           \
210                 in.ftpd.1m              \
211                 in.iked.1m              \
212                 in.lpd.1m               \
213                 in.mpathd.1m            \
214                 in.ndpd.1m              \
215                 in.rarpd.1m             \
216                 in.rdisc.1m             \
217                 in.rexecd.1m            \
218                 in.ripngd.1m            \
219                 in.rlogind.1m           \
220                 in.routed.1m            \
221                 in.rshd.1m              \
222                 in.rwhod.1m             \
223                 in.talkd.1m             \
224                 in.telnetd.1m           \
225                 in.tftpd.1m             \
226                 in.timed.1m             \
227                 in.uucpd.1m             \
228                 inetadm.1m              \
229                 inetconv.1m             \
230                 inetd.1m                \
231                 infocmp.1m              \
232                 init.1m                 \
233                 inityp2l.1m             \
234                 install.1m              \
235                 installf.1m             \
236                 installgrub.1m          \
237                 intrd.1m                \
238                 intrstat.1m             \
239                 iostat.1m               \
240                 ipaddrsel.1m            \
241                 ipadm.1m                \
242                 ipdadm.1m               \
243                 ipf.1m                  \
244                 ipfs.1m                 \
245                 ipfstat.1m              \
246                 ipmon.1m                \
247                 ipmpstat.1m             \
248                 ipnat.1m                \
249                 ippool.1m               \
250                 ipqosconf.1m            \
251                 ipsecalgs.1m            \
252                 ipsecconf.1m            \
253                 ipseckey.1m             \
254                 iscsiadm.1m             \
255                 isns.1m                 \
256                 isnsadm.1m              \
257                 itadm.1m                \
258                 k5srvutil.1m            \
```

```
259                 kadb.1m                 \
260                 kadmin.1m               \
261                 kadmind.1m              \
262                 kclient.1m              \
263                 kdb5_ldap_util.1m       \
264                 kdb5_util.1m            \
265                 kdcmgr.1m               \
266                 kernel.1m               \
267                 keyserv.1m              \
268                 killall.1m              \
269                 kprop.1m                \
270                 kpropd.1m               \
271                 kproplog.1m             \
272                 krb5kdc.1m              \
273                 ksslcfg.1m              \
274                 kstat.1m                \
275                 ktkt_warnd.1m           \
276                 labelit.1m              \
277                 labelit_hsfs.1m         \
278                 labelit_udfs.1m         \
279                 labelit_ufs.1m          \
280                 latencytop.1m           \
281                 ldap_cachemgr.1m        \
282                 ldapaddent.1m           \
283                 ldapclient.1m           \
284                 link.1m                 \
285                 listdgrp.1m             \
286                 listen.1m               \
287                 locator.1m              \
288                 lockfs.1m               \
289                 lockstat.1m             \
290                 lofiadm.1m              \
291                 logadm.1m               \
292                 logins.1m               \
293                 lpadmin.1m              \
294                 lpfilter.1m             \
295                 lpforms.1m              \
296                 lpget.1m                \
297                 lpmove.1m               \
298                 lpsched.1m              \
299                 lpset.1m                \
300                 lpshut.1m               \
301                 lpsystem.1m             \
302                 lpusers.1m              \
303                 luxadm.1m               \
304                 mail.local.1m           \
305                 makedbm.1m              \
306                 makemap.1m              \
307                 mdmonitord.1m           \
308                 medstat.1m              \
309                 metaclear.1m            \
310                 metadb.1m               \
311                 metadevadm.1m           \
312                 metahs.1m               \
313                 metaimport.1m           \
314                 metainit.1m             \
315                 metaoffline.1m          \
316                 metaparam.1m            \
317                 metarecover.1m          \
318                 metarename.1m           \
319                 metareplace.1m          \
320                 metaroot.1m             \
321                 metaset.1m              \
322                 metassist.1m            \
323                 metastat.1m             \
324                 metasync.1m             \
```



```
325                 metattach.1m            \
326                 mkdevalloc.1m           \
327                 mkdevmaps.1m            \
328                 mkfifo.1m               \
329                 mkfile.1m               \
330                 mkfs.1m                 \
331                 mkfs_pcfs.1m            \
332                 mkfs_udfs.1m            \
333                 mkfs_ufs.1m             \
334                 mknod.1m                \
335                 mkpwdict.1m             \
336                 modinfo.1m              \
337                 modload.1m              \
338                 modunload.1m            \
339                 mount.1m                \
340                 mount_cachefs.1m        \
341                 mount_hsfs.1m           \
342                 mount_nfs.1m            \
343                 mount_pcfs.1m           \
344                 mount_smbfs.1m          \
345                 mount_tmpfs.1m          \
346                 mount_udfs.1m           \
347                 mount_ufs.1m            \
348                 mountall.1m             \
349                 mountd.1m               \
350                 mpathadm.1m             \
351                 mpstat.1m               \
352                 msgid.1m                \
353                 mvdir.1m                \
354                 ncaconfd.1m             \
355                 ncheck.1m               \
356                 ncheck_ufs.1m           \
357                 ndd.1m                  \
358                 ndmpadm.1m              \
359                 ndmpd.1m                \
360                 ndmpstat.1m             \
361                 netstat.1m              \
362                 netstrategy.1m          \
363                 newaliases.1m           \
364                 newfs.1m                \
365                 newkey.1m               \
366                 nfs4cbd.1m              \
367                 nfsd.1m                 \
368                 nfslogd.1m              \
369                 nfsmapid.1m             \
370                 nfsstat.1m              \
371                 nlsadmin.1m             \
372                 nscadm.1m               \
373                 nscd.1m                 \
374                 nwamd.1m                \
375                 passmgmt.1m             \
376                 pbind.1m                \
377                 picld.1m                \
378                 ping.1m                 \
379                 pkgadd.1m               \
380                 pkgadm.1m               \
381                 pkgask.1m               \
382                 pkgchk.1m               \
383                 pkgrm.1m                \
384                 plockstat.1m            \
385                 pmadm.1m                \
386                 pmconfig.1m             \
387                 pntadm.1m               \
388                 polkit-is-privileged.1m \
389                 pooladm.1m              \
390                 poolbind.1m             \
```

```
391                 poolcfg.1m              \
392                 poold.1m                \
393                 poolstat.1m             \
394                 ports.1m                \
395                 powerd.1m               \
396                 powertop.1m             \
397                 ppdmgr.1m               \
398                 pppd.1m                 \
399                 pppdump.1m              \
400                 pppoec.1m               \
401                 pppoed.1m               \
402                 pppstats.1m             \
403                 praudit.1m              \
404                 print-service.1m        \
405                 printmgr.1m             \
406                 projadd.1m              \
407                 projdel.1m              \
408                 projmod.1m              \
409                 prstat.1m               \
410                 prtconf.1m              \
411                 prtdiag.1m              \
412                 prtfru.1m               \
413                 prtpicl.1m              \
414                 prtvtoc.1m              \
415                 psradm.1m               \
416                 psrinfo.1m              \
417                 psrset.1m               \
418                 putdev.1m               \
419                 putdgrp.1m              \
420                 pwck.1m                 \
421                 pwconv.1m               \
422                 quot.1m                 \
423                 quota.1m                \
424                 quotacheck.1m           \
425                 quotaon.1m              \
426                 raidctl.1m              \
427                 ramdiskadm.1m           \
428                 rcapadm.1m              \
429                 rcapd.1m                \
430                 rctladm.1m              \
431                 rdate.1m                \
432                 reboot.1m               \
433                 rem_drv.1m              \
434                 remove_allocatable.1m   \
435                 removef.1m              \
436                 repquota.1m             \
437                 rmmount.1m              \
438                 rmt.1m                  \
439                 rmvolmgr.1m             \
440                 roleadd.1m              \
441                 roledel.1m              \
442                 rolemod.1m              \
443                 root_archive.1m         \
444                 route.1m                \
445                 routeadm.1m             \
446                 rpc.bootparamd.1m       \
447                 rpc.mdcommd.1m          \
448                 rpc.metad.1m            \
449                 rpc.metamedd.1m         \
450                 rpc.metamhd.1m          \
451                 rpc.rexd.1m             \
452                 rpc.rstatd.1m           \
453                 rpc.rusersd.1m          \
454                 rpc.rwalld.1m           \
455                 rpc.smserverd.1m        \
456                 rpc.sprayd.1m           \
```



```
457                 rpc.yppasswdd.1m        \
458                 rpc.ypupdated.1m        \
459                 rpcbind.1m              \
460                 rpcinfo.1m              \
461                 rquotad.1m              \
462                 rsh.1m                  \
463                 rtc.1m                  \
464                 rtquery.1m              \
465                 runacct.1m              \
466                 rwall.1m                \
467                 sac.1m                  \
468                 sacadm.1m               \
469                 saf.1m                  \
470                 sar.1m                  \
471                 sasinfo.1m              \
472                 savecore.1m             \
473                 sbdadm.1m               \
474                 scadm.1m                \
475                 scmadm.1m               \
476                 sdpadm.1m               \
477                 sendmail.1m             \
478                 setuname.1m             \
479                 sftp-server.1m          \
480                 share.1m                \
481                 share_nfs.1m            \
482                 shareall.1m             \
483                 sharectl.1m             \
484                 sharemgr.1m             \
485                 showmount.1m            \
486                 shutdown.1m             \
487                 slpd.1m                 \
488                 smbadm.1m               \
489                 smbd.1m                 \
490                 smbiod.1m               \
491                 smbios.1m               \
492                 smbstat.1m              \
493                 smrsh.1m                \
494                 smtnrhdb.1m             \
495                 smtnrhtp.1m             \
496                 smtnzonecfg.1m          \
497                 sndradm.1m              \
498                 sndrd.1m                \
499                 sndrsyncd.1m            \
500                 snoop.1m                \
501                 soconfig.1m             \
502                 sppptun.1m              \
503                 spray.1m                \
504                 ssh-keysign.1m          \
505                 sshd.1m                 \
506                 statd.1m                \
507                 stmfadm.1m              \
508                 stmsboot.1m             \
509                 strace.1m               \
510                 strclean.1m             \
511                 strerr.1m               \
512                 sttydefs.1m             \
513                 su.1m                   \
514                 sulogin.1m              \
515                 svadm.1m                \
516                 svc.configd.1m          \
517                 svc.ipfd.1m             \
518                 svc.startd.1m           \
519                 svcadm.1m               \
520                 svccfg.1m               \
521                 swap.1m                 \
522                 sync.1m                 \
```

```
523                 syncinit.1m             \
524                 syncloop.1m             \
525                 syncstat.1m             \
526                 sysdef.1m               \
527                 syseventadm.1m          \
528                 syseventconfd.1m        \
529                 syseventd.1m            \
530                 syslogd.1m              \
531                 tapes.1m                \
532                 tcpd.1m                 \
533                 tcpdchk.1m              \
534                 tcpdmatch.1m            \
535                 th_define.1m            \
536                 th_manage.1m            \
537                 tic.1m                  \
538                 tnchkdb.1m              \
539                 tnctl.1m                \
540                 tnd.1m                  \
541                 tninfo.1m               \
542                 tpmadm.1m               \
543                 traceroute.1m           \
544                 trapstat.1m             \
545                 ttyadm.1m               \
546                 ttymon.1m               \
547                 tunefs.1m               \
548                 txzonemgr.1m            \
549                 tzselect.1m             \
550                 uadmin.1m               \
551                 ucodeadm.1m             \
552                 ufsdump.1m              \
553                 ufsrestore.1m           \
554                 unshare.1m              \
555                 unshare_nfs.1m          \
556                 update_drv.1m           \
557                 updatehome.1m           \
558                 useradd.1m              \
559                 userdel.1m              \
560                 usermod.1m              \
561                 utmpd.1m                \
562                 uucheck.1m              \
563                 uucico.1m               \
564                 uucleanup.1m            \
565                 uusched.1m              \
566                 uuxqt.1m                \
567                 vmstat.1m               \
568                 volcopy.1m              \
569                 volcopy_ufs.1m          \
570                 vscanadm.1m             \
571                 vscand.1m               \
572                 wall.1m                 \
573                 wanboot_keygen.1m       \
574                 wanboot_keymgmt.1m      \
575                 wanboot_p12split.1m     \
576                 wanbootutil.1m          \
577                 whodo.1m                \
578                 wificonfig.1m           \
579                 wpad.1m                 \
580                 wracct.1m               \
581                 wusbadm.1m              \
582                 ypbind.1m               \
583                 ypinit.1m               \
584                 ypmake.1m               \
585                 ypmap2src.1m            \
586                 yppoll.1m               \
587                 yppush.1m               \
588                 ypserv.1m               \
```



```
589                 ypset.1m                \
590                 ypstart.1m              \
591                 ypxfr.1m                \
592                 zdb.1m                  \
593                 zdump.1m                \
594                 zfs.1m                  \
595                 zic.1m                  \
596                 zoneadm.1m              \
597                 zoneadmd.1m             \
598                 zonecfg.1m              \
599                 zpool.1m                \
600                 zstreamdump.1m

602 i386_MANFILES=  lms.1m                 \
603                 parted.1m               \
604                 mkntfs.1m               \
605                 ntfscat.1m              \
606                 ntfsclone.1m            \
607                 ntfscluster.1m          \
608                 ntfscmp.1m              \
609                 ntfscp.1m               \
610                 ntfsfix.1m              \
611                 ntfsinfo.1m             \
612                 ntfslabel.1m            \
613                 ntfsls.1m               \
614                 ntfsprogs.1m            \
615                 ntfsresize.1m           \
616                 ntfsundelete.1m

618 sparc_MANFILES= cvcd.1m                 \
619                 dcs.1m                  \
620                 drd.1m                  \
621                 efdaemon.1m             \
622                 ldmad.1m                \
623                 monitor.1m              \
624                 obpsym.1m               \
625                 oplhpd.1m               \
626                 prtdscp.1m              \
627                 sckmd.1m                \
628                 sf880drd.1m             \
629                 vntsd.1m

631 MANLINKS=       acctcon1.1m                     \
632                 acctcon2.1m                     \
633                 acctdisk.1m                     \
634                 acctdusg.1m                     \
635                 accton.1m                       \
636                 acctprc1.1m                     \
637                 acctprc2.1m                     \
638                 acctwtmp.1m                     \
639                 bootparamd.1m                   \
640                 chargefee.1m                    \
641                 ckpacct.1m                      \
642                 closewtmp.1m                    \
643                 comsat.1m                       \
644                 dcopy.1m                        \
645                 devfsadmd.1m                    \
646                 dodisk.1m                       \
647                 fcadm.1m                        \
648                 fingerd.1m                      \
649                 ftpd.1m                         \
650                 grpck.1m                        \
651                 hal-find-by-capability.1m       \
652                 hal-find-by-property.1m         \
653                 hal-set-property.1m             \
654                 intro.1m                        \
```

```
655                 kadmin.local.1m                 \
656                 lastlogin.1m                    \
657                 metadetach.1m                   \
658                 metaonline.1m                   \
659                 monacct.1m                      \
660                 nulladm.1m                      \
661                 poweroff.1m                     \
662                 prctmp.1m                       \
663                 prdaily.1m                      \
664                 prtacct.1m                      \
665                 quotaoff.1m                     \
666                 rarpd.1m                        \
667                 rdisc.1m                        \
668                 reject.1m                       \
669                 restricted_shell.1m             \
670                 rexd.1m                         \
671                 rexecd.1m                       \
672                 rlogind.1m                      \
673                 routed.1m                       \
674                 rshd.1m                         \
675                 rstatd.1m                       \
676                 rusersd.1m                      \
677                 rwalld.1m                       \
678                 rwhod.1m                        \
679                 sa1.1m                          \
680                 sa2.1m                          \
681                 sadc.1m                         \
682                 shutacct.1m                     \
683                 sprayd.1m                       \
684                 startup.1m                      \
685                 talkd.1m                        \
686                 telinit.1m                      \
687                 telnetd.1m                      \
688                 tftpd.1m                        \
689                 turnacct.1m                     \
690                 umount.1m                       \
691                 umount_smbfs.1m                 \
692                 umountall.1m                    \
693                 unlink.1m                       \
694                 unshareall.1m                   \
695                 utmp2wtmp.1m                    \
696                 uucpd.1m                        \
697                 uutry.1m                        \
698                 wtmpfix.1m                      \
699                 yppasswdd.1m                    \
700                 ypstop.1m                       \
701                 ypupdated.1m                    \
702                 ypxfr_1perday.1m                \
703                 ypxfr_1perhour.1m               \
704                 ypxfr_2perday.1m                \
705                 ypxfrd.1m

707 MANFILES=       $(_MANFILES) $($(MACH)_MANFILES)

709 intro.1m                       := LINKSRC = Intro.1m

711 uutry.1m                       := LINKSRC = Uutry.1m

713 reject.1m                      := LINKSRC = accept.1m

715 acctdisk.1m                    := LINKSRC = acct.1m
716 acctdusg.1m                    := LINKSRC = acct.1m
717 accton.1m                      := LINKSRC = acct.1m
718 acctwtmp.1m                    := LINKSRC = acct.1m
719 closewtmp.1m                   := LINKSRC = acct.1m
720 utmp2wtmp.1m                   := LINKSRC = acct.1m
```



```
722 acctcon1.1m                    := LINKSRC = acctcon.1m
723 acctcon2.1m                    := LINKSRC = acctcon.1m

725 acctprc1.1m                    := LINKSRC = acctprc.1m
726 acctprc2.1m                    := LINKSRC = acctprc.1m

728 chargefee.1m                   := LINKSRC = acctsh.1m
729 ckpacct.1m                     := LINKSRC = acctsh.1m
730 dodisk.1m                      := LINKSRC = acctsh.1m
731 lastlogin.1m                   := LINKSRC = acctsh.1m
732 monacct.1m                     := LINKSRC = acctsh.1m
733 nulladm.1m                     := LINKSRC = acctsh.1m
734 prctmp.1m                      := LINKSRC = acctsh.1m
735 prdaily.1m                     := LINKSRC = acctsh.1m
736 prtacct.1m                     := LINKSRC = acctsh.1m
737 shutacct.1m                    := LINKSRC = acctsh.1m
738 startup.1m                     := LINKSRC = acctsh.1m
739 turnacct.1m                    := LINKSRC = acctsh.1m

741 dcopy.1m                       := LINKSRC = clri.1m

743 devfsadmd.1m                   := LINKSRC = devfsadm.1m

745 fcadm.1m                       := LINKSRC = fcinfo.1m

747 wtmpfix.1m                     := LINKSRC = fwtmp.1m

749 hal-find-by-capability.1m      := LINKSRC = hal-find.1m
750 hal-find-by-property.1m        := LINKSRC = hal-find.1m

752 hal-set-property.1m           := LINKSRC = hal-get-property.1m

754 poweroff.1m                    := LINKSRC = halt.1m

756 comsat.1m                      := LINKSRC = in.comsat.1m
757 fingerd.1m                     := LINKSRC = in.fingerd.1m
758 ftpd.1m                       := LINKSRC = in.ftpd.1m
759 rarpd.1m                      := LINKSRC = in.rarpd.1m
760 rdisc.1m                      := LINKSRC = in.rdisc.1m
761 rexecd.1m                     := LINKSRC = in.rexecd.1m
762 rlogind.1m                    := LINKSRC = in.rlogind.1m
763 routed.1m                     := LINKSRC = in.routed.1m
764 rshd.1m                       := LINKSRC = in.rshd.1m
765 rwhod.1m                      := LINKSRC = in.rwhod.1m
766 talkd.1m                      := LINKSRC = in.talkd.1m
767 telnetd.1m                    := LINKSRC = in.telnetd.1m
768 tftpd.1m                      := LINKSRC = in.tftpd.1m
769 uucpd.1m                      := LINKSRC = in.uucpd.1m

771 telinit.1m                    := LINKSRC = init.1m

773 kadmin.local.1m               := LINKSRC = kadmin.1m

775 unlink.1m                     := LINKSRC = link.1m

777 metaonline.1m                 := LINKSRC = metaoffline.1m

779 metadetach.1m                 := LINKSRC = metattach.1m

781 umount.1m                     := LINKSRC = mount.1m

783 umount_smbfs.1m               := LINKSRC = mount_smbfs.1m

785 umountall.1m                  := LINKSRC = mountall.1m
```

```
 787 grpck.1m                        := LINKSRC = pwck.1m

 789 quotaoff.1m                     := LINKSRC = quotaon.1m

 791 bootparamd.1m                   := LINKSRC = rpc.bootparamd.1m
 792 rexd.1m                         := LINKSRC = rpc.rexd.1m
 793 rstatd.1m                       := LINKSRC = rpc.rstatd.1m
 794 rusersd.1m                      := LINKSRC = rpc.rusersd.1m
 795 rwalld.1m                       := LINKSRC = rpc.rwalld.1m
 796 sprayd.1m                       := LINKSRC = rpc.sprayd.1m
 797 yppasswdd.1m                    := LINKSRC = rpc.yppasswdd.1m
 798 ypupdated.1m                    := LINKSRC = rpc.ypupdated.1m

 800 restricted_shell.1m             := LINKSRC = rsh.1m

 802 sa1.1m                          := LINKSRC = sar.1m
 803 sa2.1m                          := LINKSRC = sar.1m
 804 sadc.1m                         := LINKSRC = sar.1m

 806 unshareall.1m                   := LINKSRC = shareall.1m

 808 ypxfrd.1m                       := LINKSRC = ypserv.1m

 810 ypstop.1m                       := LINKSRC = ypstart.1m

 812 ypxfr_1perday.1m                := LINKSRC = ypxfr.1m
 813 ypxfr_1perhour.1m               := LINKSRC = ypxfr.1m
 814 ypxfr_2perday.1m                := LINKSRC = ypxfr.1m


 817 .KEEP_STATE:

 819 include         $(SRC)/man/Makefile.man

 821 install:        $(ROOTMANFILES) $(ROOTMANLINKS)
```

```
**********************************************************
    4030 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/Makefile
import complete (hopefully)
**********************************************************
#
# 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 (c) 2012 by Delphix. All rights reserved.
# Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

include         $(SRC)/Makefile.master

MANSECT=        5

MANFILES=       Intro.5                 \
                acl.5                   \
                ad.5                    \
                ascii.5                 \
                attributes.5            \
                audit_binfile.5         \
                audit_remote.5          \
                audit_syslog.5          \
                brands.5                \
                cancellation.5          \
                charmap.5               \
                condition.5             \
                crypt_bsdbf.5           \
                crypt_bsdmd5.5          \
                crypt_sha256.5          \
                crypt_sha512.5          \
                crypt_sunmd5.5          \
                crypt_unix.5            \
                device_clean.5          \
                dhcp.5                  \
                dhcp_modules.5          \
                environ.5               \
                eqn.5                   \
                eqnchar.5               \
                extendedFILE.5          \
                filesystem.5            \
                fnmatch.5               \
                formats.5               \
                fsattr.5                \
                grub.5                  \
                gss_auth_rules.5        \
                hal.5                   \
                iconv.5                 \
                iconv_unicode.5         \
                ieee802.11.5            \
                ipfilter.5              \
                isalist.5               \
                kerberos.5              \
                krb5_auth_rules.5       \
```



```
                krb5envvar.5            \
                largefile.5             \
                lf64.5                  \
                lfcompile.5             \
                lfcompile64.5           \
                locale.5                \
                man.5                   \
                mandoc_char.5           \
                mandoc_roff.5           \
                mdoc.5                  \
                mansun.5                \
                me.5                    \
                mech_spnego.5           \
                mm.5                    \
                ms.5                    \
                mutex.5                 \
                nfssec.5                \
                pam_allow.5             \
                pam_authtok_check.5     \
                pam_authtok_get.5       \
                pam_authtok_store.5     \
                pam_deny.5              \
                pam_dhkeys.5            \
                pam_dial_auth.5         \
                pam_krb5.5              \
                pam_krb5_migrate.5      \
                pam_ldap.5              \
                pam_list.5              \
                pam_passwd_auth.5       \
                pam_rhosts_auth.5       \
                pam_roles.5             \
                pam_sample.5            \
                pam_smb_passwd.5        \
                pam_smbfs_login.5       \
                pam_tsol_account.5      \
                pam_unix_account.5      \
                pam_unix_auth.5         \
                pam_unix_cred.5         \
                pam_unix_session.5      \
                pkcs11_kernel.5         \
                pkcs11_softtoken.5      \
                pkcs11_tpm.5            \
                privileges.5            \
                prof.5                  \
                rbac.5                  \
                regex.5                 \
                regexp.5                \
                resource_controls.5     \
                smf.5                   \
                smf_bootstrap.5         \
                smf_method.5            \
                smf_restarter.5         \
                smf_security.5          \
                smf_template.5          \
                standards.5             \
                sticky.5                \
                tbl.5                   \
                tecla.5                 \
                term.5                  \
                threads.5               \
                trusted_extensions.5    \
                vgrindefs.5             \
                zones.5                 \
                zpool-features.5

MANLINKS=       ANSI.5          \
```

```
127                 C++.5           \
128                 C.5             \
129                 CSI.5           \
130                 ISO.5           \
131                 MT-Level.5      \
132                 POSIX.1.5       \
133                 POSIX.2.5       \
134                 POSIX.5         \
135                 RBAC.5          \
136                 SUS.5           \
137                 SUSv2.5         \
138                 SUSv3.5         \
139                 SVID.5          \
140                 SVID3.5         \
141                 XNS.5           \
142                 XNS4.5          \
143                 XNS5.5          \
144                 XPG.5           \
145                 XPG3.5          \
146                 XPG4.5          \
147                 XPG4v2.5        \
148                 advance.5       \
149                 architecture.5  \
150                 availability.5  \
151                 compile.5       \
152                 intro.5         \
153                 pthreads.5      \
154                 stability.5     \
155                 standard.5      \
156                 step.5          \
157                 teclarc.5

159 intro.5         := LINKSRC = Intro.5

161 CSI.5           := LINKSRC = attributes.5
162 MT-Level.5      := LINKSRC = attributes.5
163 architecture.5  := LINKSRC = attributes.5
164 availability.5  := LINKSRC = attributes.5
165 stability.5     := LINKSRC = attributes.5
166 standard.5      := LINKSRC = attributes.5

168 RBAC.5          := LINKSRC = rbac.5

170 advance.5       := LINKSRC = regexp.5
171 compile.5       := LINKSRC = regexp.5
172 step.5          := LINKSRC = regexp.5

174 ANSI.5          := LINKSRC = standards.5
175 C++.5           := LINKSRC = standards.5
176 C.5             := LINKSRC = standards.5
177 ISO.5           := LINKSRC = standards.5
178 POSIX.1.5       := LINKSRC = standards.5
179 POSIX.2.5       := LINKSRC = standards.5
180 POSIX.5         := LINKSRC = standards.5
181 SUS.5           := LINKSRC = standards.5
182 SUSv2.5         := LINKSRC = standards.5
183 SUSv3.5         := LINKSRC = standards.5
184 SVID.5          := LINKSRC = standards.5
185 SVID3.5         := LINKSRC = standards.5
186 XNS.5           := LINKSRC = standards.5
187 XNS4.5          := LINKSRC = standards.5
188 XNS5.5          := LINKSRC = standards.5
189 XPG.5           := LINKSRC = standards.5
190 XPG3.5          := LINKSRC = standards.5
191 XPG4.5          := LINKSRC = standards.5
192 XPG4v2.5        := LINKSRC = standards.5
```



```
194 teclarc.5       := LINKSRC = tecla.5

196 pthreads.5      := LINKSRC = threads.5

198 .KEEP_STATE:

200 include         $(SRC)/man/Makefile.man

202 install:        $(ROOTMANFILES) $(ROOTMANLINKS)
```

```
**********************************************************
    7157 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/eqn.5
import complete (hopefully)
**********************************************************
.\"
.\" 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) 2011 Kristaps Dzonsons <kristaps@bsd.lv>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\"
.Dd Sep 25, 2011
.Dt EQN 5
.Os
.Sh NAME
.Nm eqn
.Nd eqn language reference for mandoc
.Sh DESCRIPTION
The
.Nm eqn
language is an equation-formatting language.
It is used within
.Xr mdoc 5
and
.Xr man 5
.Ux
manual pages.
It describes the
.Em structure
of an equation, not its mathematical meaning.
This manual describes the
.Nm
language accepted by the
.Xr mandoc 1
utility, which corresponds to the Second Edition eqn specification (see
.Sx SEE ALSO
for references).
.Pp
Equations within
.Xr mdoc 5
or
.Xr man 5
documents are enclosed by the standalone
.Sq \&.EQ
and
.Sq \&.EN
tags.
Equations are multi-line blocks consisting of formulas and control
statements.
.Sh EQUATION STRUCTURE
Each equation is bracketed by
.Sq \&.EQ
and
.Sq \&.EN
strings.
```



```
.Em Note :
these are not the same as
.Xr roff 5
macros, and may only be invoked as
.Sq \&.EQ .
.Pp
The equation grammar is as follows, where quoted strings are
case-sensitive literals in the input:
.Bd -literal -offset indent
eqn     : box | eqn box
box     : text
        | \*q{\*q eqn \*q}\*q
        | \*qdefine\*q text text
        | \*qndefine\*q text text
        | \*qtdefine\*q text text
        | \*qgfont\*q text
        | \*qgsize\*q text
        | \*qset\*q text text
        | \*qundef\*q text
        | box pos box
        | box mark
        | \*qmatrix\*q \*q{\*q [col \*q{\*q list \*q}\*q ]*
        | pile \*q{\*q list \*q}\*q
        | font box
        | \*qsize\*q text box
        | \*qleft\*q text eqn [\*qright\*q text]
col     : \*qlcol\*q | \*qrcol\*q | \*qccol\*q | \*qcol\*q
text    : [^space\e\*q]+ | \e\*q.*\e\*q
pile    : \*qlpile\*q | \*qcpile\*q | \*qrpile\*q | \*qpile\*q
pos     : \*qover\*q | \*qsup\*q | \*qsub\*q | \*qto\*q | \*qfrom\*q
mark    : \*qdot\*q | \*qdotdot\*q | \*qhat\*q | \*qtilde\*q | \*qvec\*q
        | \*qdyad\*q | \*qbar\*q | \*qunder\*q
font    : \*qroman\*q | \*qitalic\*q | \*qbold\*q | \*qfat\*q
list    : eqn
        | list \*qabove\*q eqn
space   : [\e^~ \et]
.Ed
.Pp
White-space consists of the space, tab, circumflex, and tilde
characters.
If within a quoted string, these space characters are retained.
Quoted strings are also not scanned for replacement definitions.
.Pp
The following text terms are translated into a rendered glyph, if
available: alpha, beta, chi, delta, epsilon, eta, gamma, iota, kappa,
lambda, mu, nu, omega, omicron, phi, pi, psi, rho, sigma, tau, theta,
upsilon, xi, zeta, DELTA, GAMMA, LAMBDA, OMEGA, PHI, PI, PSI, SIGMA,
THETA, UPSILON, XI, inter (intersection), union (union), prod (product),
int (integral), sum (summation), grad (gradient), del (vector
differential), times (multiply), cdot (centre-dot), nothing (zero-width
space), approx (approximately equals), prime (prime), half (one-half),
partial (partial differential), inf (infinity), >> (much greater), <<
(much less), \-> (left arrow), <\- (right arrow), += (plus-minus), !=
(not equal), == (equivalence), <= (less-than-equal), and >=
(more-than-equal).
.Pp
The following control statements are available:
.Bl -tag -width Ds
.It Cm define
Replace all occurrences of a key with a value.
Its syntax is as follows:
.Pp
.D1 define Ar key cvalc
.Pp
The first character of the value string,
.Ar c ,
```

```
is used as the delimiter for the value
.Ar val .
This allows for arbitrary enclosure of terms (not just quotes), such as
.Pp
.D1 define Ar foo ’bar baz’
.D1 define Ar foo cbar bazc
.Pp
It is an error to have an empty
.Ar key
or
.Ar val .
Note that a quoted
.Ar key
causes errors in some
.Nm
implementations and should not be considered portable.
It is not expanded for replacements.
Definitions may refer to other definitions; these are evaluated
recursively when text replacement occurs and not when the definition is
created.
.Pp
Definitions can create arbitrary strings, for example, the following is
a legal construction.
.Bd -literal -offset indent
define foo ’define’
foo bar ’baz’
.Ed
.Pp
Self-referencing definitions will raise an error.
The
.Cm ndefine
statement is a synonym for
.Cm define ,
while
.Cm tdefine
is discarded.
.It Cm gfont
Set the default font of subsequent output.
Its syntax is as follows:
.Pp
.D1 gfont Ar font
.Pp
In mandoc, this value is discarded.
.It Cm gsize
Set the default size of subsequent output.
Its syntax is as follows:
.Pp
.D1 gsize Ar size
.Pp
The
.Ar size
value should be an integer.
.It Cm set
Set an equation mode.
In mandoc, both arguments are thrown away.
Its syntax is as follows:
.Pp
.D1 set Ar key val
.Pp
The
.Ar key
and
.Ar val
are not expanded for replacements.
This statement is a GNU extension.
.It Cm undef
```



```
Unset a previously-defined key.
Its syntax is as follows:
.Pp
.D1 define Ar key
.Pp
Once invoked, the definition for
.Ar key
is discarded.
The
.Ar key
is not expanded for replacements.
This statement is a GNU extension.
.El
.Sh COMPATIBILITY
This section documents the compatibility of mandoc
.Nm
and the troff
.Nm
implementation (including GNU troff).
.Pp
.Bl -dash -compact
.It
The text string
.Sq \e\*q
is interpreted as a literal quote in troff.
In mandoc, this is interpreted as a comment.
.It
In troff, The circumflex and tilde white-space symbols map to
fixed-width spaces.
In mandoc, these characters are synonyms for the space character.
.It
The troff implementation of
.Nm
allows for equation alignment with the
.Cm mark
and
.Cm lineup
tokens.
mandoc discards these tokens.
The
.Cm back Ar n ,
.Cm fwd Ar n ,
.Cm up Ar n ,
and
.Cm down Ar n
commands are also ignored.
.El
.Sh SEE ALSO
.Xr mandoc 1 ,
.Xr man 7 ,
.Xr mandoc_char 5 ,
.Xr mdoc 5 ,
.Xr roff 5
.Rs
.%A Brian W. Kernighan
.%A Lorinda L. Cherry
.%T System for Typesetting Mathematics
.%J Communications of the ACM
.%V 18
.%P 151\(en157
.%D March, 1975
.Re
.Rs
.%A Brian W. Kernighan
.%A Lorinda L. Cherry
.%T Typesetting Mathematics, User’s Guide
```

```
.%D 1976
.Re
.Rs
.%A Brian W. Kernighan
.%A Lorinda L. Cherry
.%T Typesetting Mathematics, User's Guide (Second Edition)
.%D 1978
.Re
.Sh HISTORY
The eqn utility, a preprocessor for troff, was originally written by
Brian W. Kernighan and Lorinda L. Cherry in 1975.
The GNU reimplementation of eqn, part of the GNU troff package, was
released in 1989 by James Clark.
The eqn component of
.Xr mandoc 1
was added in 2011.
.Sh AUTHORS
This
.Nm
reference was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv .
```

```
**********************************************************
   23289 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/man.5
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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 (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\"
.Dd Jan 3, 2012
.Dt MAN 5
.Os
.Sh NAME
.Nm man
.Nd legacy formatting language for manual pages
.Sh DESCRIPTION
Traditionally, the
.Nm man
language has been used to write
.Ux
manuals for the
.Xr man 1
utility.
It supports limited control of presentational details like fonts,
indentation and spacing.
This reference document describes the structure of manual pages
and the syntax and usage of the man language.
.Pp
.Bf -emphasis
Do not use
.Nm
to write your manuals:
.Ef
It lacks support for semantic markup.
Use the
.Xr mdoc 5
language, instead.
.Pp
In a
.Nm
document, lines beginning with the control character
.Sq \&.
are called
.Dq macro lines .
The first word is the macro name.
It usually consists of two capital letters.
For a list of available macros, see
.Sx MACRO OVERVIEW .
The words following the macro name are arguments to the macro.
.Pp
Lines not beginning with the control character are called
.Dq text lines .
```



```
They provide free-form text to be printed; the formatting of the text
depends on the respective processing context:
.Bd -literal -offset indent
\&.SH Macro lines change control state.
Text lines are interpreted within the current state.
.Ed
.Pp
Many aspects of the basic syntax of the
.Nm
language are based on the
.Xr roff 5
language; see the
.Em LANGUAGE SYNTAX
and
.Em MACRO SYNTAX
sections in the
.Xr roff 5
manual for details, in particular regarding
comments, escape sequences, whitespace, and quoting.
.Sh MANUAL STRUCTURE
Each
.Nm
document must contain the
.Sx \&TH
macro describing the document's section and title.
It may occur anywhere in the document, although conventionally it
appears as the first macro.
.Pp
Beyond
.Sx \&TH ,
at least one macro or text line must appear in the document.
.Pp
The following is a well-formed skeleton
.Nm
file for a utility
.Qq progname :
.Bd -literal -offset indent
\&.TH PROGNAME 1 "Oct 10, 2009"
\&.SH NAME
\efBprogname\efR \e(en a description goes here
\&.\e\(dq .SH LIBRARY
\&.\e\(dq For sections 2 & 3 only.
\&.SH SYNOPSIS
\efBprogname\efR [\efB\e-options\efR] arguments...
\&.SH DESCRIPTION
The \efBfoo\efR utility processes files...
\&.\e\(dq .SH IMPLEMENTATION NOTES
\&.\e\(dq .SH RETURN VALUES
\&.\e\(dq For sections 2, 3, & 9 only.
\&.\e\(dq .SH ENVIRONMENT
\&.\e\(dq For sections 1, 1M, 5, & 6 only.
\&.\e\(dq .SH FILES
\&.\e\(dq .SH EXIT STATUS
\&.\e\(dq For sections 1, 1M, & 6 only.
\&.\e\(dq .SH EXAMPLES
\&.\e\(dq .SH DIAGNOSTICS
\&.\e\(dq For sections 1, 1M, 5, 6, & 7 only.
\&.\e\(dq .SH ERRORS
\&.\e\(dq For sections 2, 3, & 9 only.
\&.\e\(dq .SH SEE ALSO
\&.\e\(dq .BR foo ( 1 )
\&.\e\(dq .SH STANDARDS
\&.\e\(dq .SH HISTORY
\&.\e\(dq .SH AUTHORS
\&.\e\(dq .SH CAVEATS
\&.\e\(dq .SH BUGS
```

```
\&.\e\(dq .SH SECURITY CONSIDERATIONS
.Ed
.Pp
The sections in a
.Nm
document are conventionally ordered as they appear above.
Sections should be composed as follows:
.Bl -ohang -offset indent
.It Em NAME
The name(s) and a short description of the documented material.
The syntax for this is generally as follows:
.Pp
.D1 \efBname\efR \e(en description
.It Em LIBRARY
The name of the library containing the documented material, which is
assumed to be a function in a section 2 or 3 manual.
For functions in the C library, this may be as follows:
.Pp
.D1 Standard C Library (libc, -lc)
.It Em SYNOPSIS
Documents the utility invocation syntax, function call syntax, or device
configuration.
.Pp
For the first, utilities (sections 1, 1M, and 6), this is
generally structured as follows:
.Pp
.D1 \efBname\efR [-\efBab\efR] [-\efBc\efR\efIarg\efR] \efBpath\efR...
.Pp
For the second, function calls (sections 2, 3, 9):
.Pp
.D1 \&.B char *name(char *\efIarg\efR);
.Pp
And for the third, configurations (section 7):
.Pp
.D1 \&.B name* at cardbus ? function ?
.Pp
Manuals not in these sections generally don't need a
.Em SYNOPSIS .
.It Em DESCRIPTION
This expands upon the brief, one-line description in
.Em NAME .
It usually contains a break-down of the options (if documenting a
command).
.It Em IMPLEMENTATION NOTES
Implementation-specific notes should be kept here.
This is useful when implementing standard functions that may have side
effects or notable algorithmic implications.
.It Em RETURN VALUES
This section documents the return values of functions in sections 2, 3, and 9.
.It Em ENVIRONMENT
Documents any usages of environment variables, e.g.,
.Xr environ 5 .
.It Em FILES
Documents files used.
It's helpful to document both the file name and a short description of how
the file is used (created, modified, etc.).
.It Em EXIT STATUS
This section documents the command exit status for
section 1, 6, and 8 utilities.
Historically, this information was described in
.Em DIAGNOSTICS ,
a practise that is now discouraged.
.It Em EXAMPLES
Example usages.
This often contains snippets of well-formed,
well-tested invocations.
```



```
Make sure that examples work properly!
.It Em DIAGNOSTICS
Documents error conditions.
This is most useful in section 4 manuals.
Historically, this section was used in place of
.Em EXIT STATUS
for manuals in sections 1, 6, and 8; however, this practise is
discouraged.
.It Em ERRORS
Documents error handling in sections 2, 3, and 9.
.It Em SEE ALSO
References other manuals with related topics.
This section should exist for most manuals.
.Pp
.D1 \&.BR bar \&( 1 \&),
.Pp
Cross-references should conventionally be ordered
first by section, then alphabetically.
.It Em STANDARDS
References any standards implemented or used, such as
.Pp
.D1 IEEE Std 1003.2 (\e(lqPOSIX.2\e(rq)
.Pp
If not adhering to any standards, the
.Em HISTORY
section should be used.
.It Em HISTORY
A brief history of the subject, including where support first appeared.
.It Em AUTHORS
Credits to the person or persons who wrote the code and/or documentation.
Authors should generally be noted by both name and email address.
.It Em CAVEATS
Common misuses and misunderstandings should be explained
in this section.
.It Em BUGS
Known bugs, limitations, and work-arounds should be described
in this section.
.It Em SECURITY CONSIDERATIONS
Documents any security precautions that operators should consider.
.El
.Sh MACRO OVERVIEW
This overview is sorted such that macros of similar purpose are listed
together, to help find the best macro for any given purpose.
Deprecated macros are not included in the overview, but can be found
in the alphabetical reference below.
.Ss Page header and footer meta-data
.Bl -column "PP, LP, P" description
.It Sx TH Ta set the title: Ar title section date Op Ar source Op Ar volume
.It Sx AT Ta display AT&T UNIX version in the page footer (<= 1 argument)
.It Sx UC Ta display BSD version in the page footer (<= 1 argument)
.El
.Ss Sections and paragraphs
.Bl -column "PP, LP, P" description
.It Sx SH Ta section header (one line)
.It Sx SS Ta subsection header (one line)
.It Sx PP , LP , P Ta start an undecorated paragraph (no arguments)
.It Sx RS , RE Ta reset the left margin: Op Ar width
.It Sx IP Ta indented paragraph: Op Ar head Op Ar width
.It Sx TP Ta tagged paragraph: Op Ar width
.It Sx HP Ta hanged paragraph: Op Ar width
.It Sx \&br Ta force output line break in text mode (no arguments)
.It Sx \&sp Ta force vertical space: Op Ar height
.It Sx fi , nf Ta fill mode and no-fill mode (no arguments)
.It Sx in Ta additional indent: Op Ar width
.El
.Ss Physical markup
```

```
.Bl -column "PP, LP, P" description
.It Sx B Ta boldface font
.It Sx I Ta italic font
.It Sx R Ta roman (default) font
.It Sx SB Ta small boldface font
.It Sx SM Ta small roman font
.It Sx BI Ta alternate between boldface and italic fonts
.It Sx BR Ta alternate between boldface and roman fonts
.It Sx IB Ta alternate between italic and boldface fonts
.It Sx IR Ta alternate between italic and roman fonts
.It Sx RB Ta alternate between roman and boldface fonts
.It Sx RI Ta alternate between roman and italic fonts
.El
.Ss Semantic markup
.Bl -column "PP, LP, P" description
.It Sx OP Ta optional arguments
.El
.Sh MACRO REFERENCE
This section is a canonical reference to all macros, arranged
alphabetically.
For the scoping of individual macros, see
.Sx MACRO SYNTAX .
.Ss \&AT
Sets the volume for the footer for compatibility with man pages from
.Tn AT&T UNIX
releases.
The optional arguments specify which release it is from.
.Ss \&B
Text is rendered in bold face.
.Pp
See also
.Sx \&I
and
.Sx \&R .
.Ss \&BI
Text is rendered alternately in bold face and italic.
Thus,
.Sq .BI this word and that
causes
.Sq this
and
.Sq and
to render in bold face, while
.Sq word
and
.Sq that
render in italics.
Whitespace between arguments is omitted in output.
.Pp
Examples:
.Pp
.Dl \&.BI bold italic bold italic
.Pp
The output of this example will be emboldened
.Dq bold
and italicised
.Dq italic ,
with spaces stripped between arguments.
.Pp
See also
.Sx \&IB ,
.Sx \&BR ,
.Sx \&RB ,
.Sx \&RI ,
and
.Sx \&IR .
```



```
.Ss \&BR
Text is rendered alternately in bold face and roman (the default font).
Whitespace between arguments is omitted in output.
.Pp
See
.Sx \&BI
for an equivalent example.
.Pp
See also
.Sx \&BI ,
.Sx \&IB ,
.Sx \&RB ,
.Sx \&RI ,
and
.Sx \&IR .
.Ss \&DT
Has no effect.
Included for compatibility.
.Ss \&HP
Begin a paragraph whose initial output line is left-justified, but
subsequent output lines are indented, with the following syntax:
.Bd -filled -offset indent
.Pf \. Sx \&HP
.Op Cm width
.Ed
.Pp
The
.Cm width
argument must conform to
.Sx Scaling Widths .
If specified, it’s saved for later paragraph left-margins; if unspecified, the
saved or default width is used.
.Pp
See also
.Sx \&IP ,
.Sx \&LP ,
.Sx \&P ,
.Sx \&PP ,
and
.Sx \&TP .
.Ss \&I
Text is rendered in italics.
.Pp
See also
.Sx \&B
and
.Sx \&R .
.Ss \&IB
Text is rendered alternately in italics and bold face.
Whitespace between arguments is omitted in output.
.Pp
See
.Sx \&BI
for an equivalent example.
.Pp
See also
.Sx \&BI ,
.Sx \&BR ,
.Sx \&RB ,
.Sx \&RI ,
and
.Sx \&IR .
.Ss \&IP
Begin an indented paragraph with the following syntax:
.Bd -filled -offset indent
.Pf \. Sx \&IP
```

```
.Op Cm head Op Cm width
.Ed
.Pp
The
.Cm width
argument defines the width of the left margin and is defined by
.Sx Scaling Widths .
It's saved for later paragraph left-margins; if unspecified, the saved or
default width is used.
.Pp
The
.Cm head
argument is used as a leading term, flushed to the left margin.
This is useful for bulleted paragraphs and so on.
.Pp
See also
.Sx \&HP ,
.Sx \&LP ,
.Sx \&P ,
.Sx \&PP ,
and
.Sx \&TP .
.Ss \&IR
Text is rendered alternately in italics and roman (the default font).
Whitespace between arguments is omitted in output.
.Pp
See
.Sx \&BI
for an equivalent example.
.Pp
See also
.Sx \&BI ,
.Sx \&IB ,
.Sx \&BR ,
.Sx \&RB ,
and
.Sx \&RI .
.Ss \&LP
Begin an undecorated paragraph.
The scope of a paragraph is closed by a subsequent paragraph,
sub-section, section, or end of file.
The saved paragraph left-margin width is reset to the default.
.Pp
See also
.Sx \&HP ,
.Sx \&IP ,
.Sx \&P ,
.Sx \&PP ,
and
.Sx \&TP .
.Ss \&OP
Optional command-line argument.
This has the following syntax:
.Bd -filled -offset indent
.Pf \. Sx \&OP
.Cm key Op Cm value
.Ed
.Pp
The
.Cm key
is usually a command-line flag and
.Cm value
its argument.
.Ss \&P
Synonym for
.Sx \&LP .
```



```
.Pp
See also
.Sx \&HP ,
.Sx \&IP ,
.Sx \&LP ,
.Sx \&PP ,
and
.Sx \&TP .
.Ss \&PP
Synonym for
.Sx \&LP .
.Pp
See also
.Sx \&HP ,
.Sx \&IP ,
.Sx \&LP ,
.Sx \&P ,
and
.Sx \&TP .
.Ss \&R
Text is rendered in roman (the default font).
.Pp
See also
.Sx \&I
and
.Sx \&B .
.Ss \&RB
Text is rendered alternately in roman (the default font) and bold face.
Whitespace between arguments is omitted in output.
.Pp
See
.Sx \&BI
for an equivalent example.
.Pp
See also
.Sx \&BI ,
.Sx \&IB ,
.Sx \&BR ,
.Sx \&RI ,
and
.Sx \&IR .
.Ss \&RE
Explicitly close out the scope of a prior
.Sx \&RS .
The default left margin is restored to the state of the original
.Sx \&RS
invocation.
.Ss \&RI
Text is rendered alternately in roman (the default font) and italics.
Whitespace between arguments is omitted in output.
.Pp
See
.Sx \&BI
for an equivalent example.
.Pp
See also
.Sx \&BI ,
.Sx \&IB ,
.Sx \&BR ,
.Sx \&RB ,
and
.Sx \&IR .
.Ss \&RS
Temporarily reset the default left margin.
This has the following syntax:
.Bd -filled -offset indent
```

```
.Pf \. Sx \&RS
.Op Cm width
.Ed
.Pp
The
.Cm width
argument must conform to
.Sx Scaling Widths .
If not specified, the saved or default width is used.
.Pp
See also
.Sx \&RE .
.Ss \&SB
Text is rendered in small size (one point smaller than the default font)
bold face.
.Ss \&SH
Begin a section.
The scope of a section is only closed by another section or the end of
file.
The paragraph left-margin width is reset to the default.
.Ss \&SM
Text is rendered in small size (one point smaller than the default
font).
.Ss \&SS
Begin a sub-section.
The scope of a sub-section is closed by a subsequent sub-section,
section, or end of file.
The paragraph left-margin width is reset to the default.
.Ss \&TH
Sets the title of the manual page with the following syntax:
.Bd -filled -offset indent
.Pf \. Sx \&TH
.Ar title section date
.Op Ar source Op Ar volume
.Ed
.Pp
Conventionally, the document
.Ar title
is given in all caps.
The recommended
.Ar date
format is
.Sy YYYY-MM-DD
as specified in the ISO-8601 standard;
if the argument does not conform, it is printed verbatim.
If the
.Ar date
is empty or not specified, the current date is used.
The optional
.Ar source
string specifies the organisation providing the utility.
The
.Ar volume
string replaces the default rendered volume, which is dictated by the
manual section.
.Pp
Examples:
.Pp
.Dl \&.TH CVS 5 "1992-02-12" GNU
.Ss \&TP
Begin a paragraph where the head, if exceeding the indentation width, is
followed by a newline; if not, the body follows on the same line after a
buffer to the indentation width.
Subsequent output lines are indented.
The syntax is as follows:
.Bd -filled -offset indent
```



```
.Pf \. Sx \&TP
.Op Cm width
.Ed
.Pp
The
.Cm width
argument must conform to
.Sx Scaling Widths .
If specified, it’s saved for later paragraph left-margins; if
unspecified, the saved or default width is used.
.Pp
See also
.Sx \&HP ,
.Sx \&IP ,
.Sx \&LP ,
.Sx \&P ,
and
.Sx \&PP .
.Ss \&UC
Sets the volume for the footer for compatibility with man pages from
BSD releases.
The optional first argument specifies which release it is from.
.Ss \&br
Breaks the current line.
Consecutive invocations have no further effect.
.Pp
See also
.Sx \&sp .
.Ss \&fi
End literal mode begun by
.Sx \&nf .
.Ss \&ft
Change the current font mode.
See
.Sx Text Decoration
for a listing of available font modes.
.Ss \&in
Indent relative to the current indentation:
.Pp
.D1 Pf \. Sx \&in Op Cm width
.Pp
If
.Cm width
is signed, the new offset is relative.
Otherwise, it is absolute.
This value is reset upon the next paragraph, section, or sub-section.
.Ss \&na
Don’t align to the right margin.
.Ss \&nf
Begin literal mode: all subsequent free-form lines have their end of
line boundaries preserved.
May be ended by
.Sx \&fi .
Literal mode is implicitly ended by
.Sx \&SH
or
.Sx \&SS .
.Ss \&sp
Insert vertical spaces into output with the following syntax:
.Bd -filled -offset indent
.Pf \. Sx \&sp
.Op Cm height
.Ed
.Pp
Insert
.Cm height
```

```
spaces, which must conform to
.Sx Scaling Widths .
If 0, this is equivalent to the
.Sx \&br
macro.
Defaults to 1, if unspecified.
.Pp
See also
.Sx \&br .
.Sh MACRO SYNTAX
The
.Nm
macros are classified by scope: line scope or block scope.
Line macros are only scoped to the current line (and, in some
situations, the subsequent line).
Block macros are scoped to the current line and subsequent lines until
closed by another block macro.
.Ss Line Macros
Line macros are generally scoped to the current line, with the body
consisting of zero or more arguments.
If a macro is scoped to the next line and the line arguments are empty,
the next line, which must be text, is used instead.
Thus:
.Bd -literal -offset indent
\&.I
foo
.Ed
.Pp
is equivalent to
.Sq \&.I foo .
If next-line macros are invoked consecutively, only the last is used.
If a next-line macro is followed by a non-next-line macro, an error is
raised, except for
.Sx \&br ,
.Sx \&sp ,
and
.Sx \&na .
.Pp
The syntax is as follows:
.Bd -literal -offset indent
\&.YO \(lBbody...\(rB
\(lBbody...\(rB
.Ed
.Bl -column "MacroX" "ArgumentsX" "ScopeXXXXX" "CompatX" -offset indent
.It Em Macro Ta Em Arguments Ta Em Scope     Ta Em Notes
.It Sx \&AT  Ta    <=1       Ta    current   Ta    \&
.It Sx \&B   Ta    n         Ta    next-line Ta    \&
.It Sx \&BI  Ta    n         Ta    current   Ta    \&
.It Sx \&BR  Ta    n         Ta    current   Ta    \&
.It Sx \&DT  Ta    0         Ta    current   Ta    \&
.It Sx \&I   Ta    n         Ta    next-line Ta    \&
.It Sx \&IB  Ta    n         Ta    current   Ta    \&
.It Sx \&IR  Ta    n         Ta    current   Ta    \&
.It Sx \&OP  Ta    0, 1      Ta    current   Ta    compat
.It Sx \&R   Ta    n         Ta    next-line Ta    \&
.It Sx \&RB  Ta    n         Ta    current   Ta    \&
.It Sx \&RI  Ta    n         Ta    current   Ta    \&
.It Sx \&SB  Ta    n         Ta    next-line Ta    \&
.It Sx \&SM  Ta    n         Ta    next-line Ta    \&
.It Sx \&TH  Ta    >1, <6    Ta    current   Ta    \&
.It Sx \&UC  Ta    <=1       Ta    current   Ta    \&
.It Sx \&br  Ta    0         Ta    current   Ta    compat
.It Sx \&fi  Ta    0         Ta    current   Ta    compat
.It Sx \&ft  Ta    1         Ta    current   Ta    compat
.It Sx \&in  Ta    1         Ta    current   Ta    compat
.It Sx \&na  Ta    0         Ta    current   Ta    compat
```



```
.It Sx \&nf  Ta    0         Ta    current   Ta    compat
.It Sx \&sp  Ta    1         Ta    current   Ta    compat
.El
.Pp
Macros marked as
.Qq compat
are included for compatibility with the significant corpus of existing
manuals that mix dialects of roff.
These macros should not be used for portable
.Nm
manuals.
.Ss Block Macros
Block macros comprise a head and body.
As with in-line macros, the head is scoped to the current line and, in
one circumstance, the next line (the next-line stipulations as in
.Sx Line Macros
apply here as well).
.Pp
The syntax is as follows:
.Bd -literal -offset indent
\&.YO \(lBhead...\(rB
\(lBhead...\(rB
\(lBbody...\(rB
.Ed
.Pp
The closure of body scope may be to the section, where a macro is closed
by
.Sx \&SH ;
sub-section, closed by a section or
.Sx \&SS ;
part, closed by a section, sub-section, or
.Sx \&RE ;
or paragraph, closed by a section, sub-section, part,
.Sx \&HP ,
.Sx \&IP ,
.Sx \&LP ,
.Sx \&P ,
.Sx \&PP ,
or
.Sx \&TP .
No closure refers to an explicit block closing macro.
.Pp
As a rule, block macros may not be nested; thus, calling a block macro
while another block macro scope is open, and the open scope is not
implicitly closed, is syntactically incorrect.
.Bl -column "MacroX" "ArgumentsX" "Head ScopeX" "sub-sectionX" "compatX" -offset
.It Em Macro Ta Em Arguments Ta Em Head Scope Ta Em Body Scope  Ta Em Notes
.It Sx \&HP  Ta    <2        Ta    current    Ta    paragraph   Ta    \&
.It Sx \&IP  Ta    <3        Ta    current    Ta    paragraph   Ta    \&
.It Sx \&LP  Ta    0         Ta    current    Ta    paragraph   Ta    \&
.It Sx \&P   Ta    0         Ta    current    Ta    paragraph   Ta    \&
.It Sx \&PP  Ta    0         Ta    current    Ta    paragraph   Ta    \&
.It Sx \&RE  Ta    0         Ta    current    Ta    none        Ta    compat
.It Sx \&RS  Ta    1         Ta    current    Ta    part        Ta    compat
.It Sx \&SH  Ta    >0        Ta    next-line  Ta    section     Ta    \&
.It Sx \&SS  Ta    >0        Ta    next-line  Ta    sub-section Ta    \&
.It Sx \&TP  Ta    n         Ta    next-line  Ta    paragraph   Ta    \&
.El
.Pp
Macros marked
.Qq compat
are as mentioned in
.Sx Line Macros .
.Pp
If a block macro is next-line scoped, it may only be followed by in-line
macros for decorating text.
```

```
.Ss Font handling
In
.Nm
documents, both
.Sx Physical markup
macros and
.Xr roff 5
.Ql \ef
font escape sequences can be used to choose fonts.
In text lines, the effect of manual font selection by escape sequences
only lasts until the next macro invocation; in macro lines, it only lasts
until the end of the macro scope.
Note that macros like
.Sx \&BR
open and close a font scope for each argument.
.Sh COMPATIBILITY
This section documents areas of questionable portability between
implementations of the
.Nm
language.
.Pp
.Bl -dash -compact
.It
Do not depend on
.Sx \&SH
or
.Sx \&SS
to close out a literal context opened with
.Sx \&nf .
This behaviour may not be portable.
.It
In quoted literals, GNU troff allowed pair-wise double-quotes to produce
a standalone double-quote in formatted output.
It is not known whether this behaviour is exhibited by other formatters.
.It
troff suppresses a newline before
.Sq \(aq
macro output; in mandoc, it is an alias for the standard
.Sq \&.
control character.
.It
The
.Sq \eh
.Pq horizontal position ,
.Sq \ev
.Pq vertical position ,
.Sq \em
.Pq text colour ,
.Sq \eM
.Pq text filling colour ,
.Sq \ez
.Pq zero-length character ,
.Sq \ew
.Pq string length ,
.Sq \ek
.Pq horizontal position marker ,
.Sq \eo
.Pq text overstrike ,
and
.Sq \es
.Pq text size
escape sequences are all discarded in mandoc.
.It
The
.Sq \ef
scaling unit is accepted by mandoc, but rendered as the default unit.
```



```
.It
The
.Sx \&sp
macro does not accept negative values in mandoc.
In GNU troff, this would result in strange behaviour.
.It
In page header lines, GNU troff versions up to and including 1.21
only print
.Ar volume
names explicitly specified in the
.Sx \&TH
macro; mandoc and newer groff print the default volume name
corresponding to the
.Ar section
number when no
.Ar volume
is given, like in
.Xr mdoc 5 .
.El
.Pp
The
.Sx OP
macro is part of the extended
.Nm
macro set, and may not be portable to non-GNU troff implementations.
.Sh SEE ALSO
.Xr man 1 ,
.Xr mandoc 1 ,
.Xr eqn 5 ,
.Xr mandoc_char 5 ,
.Xr mdoc 5 ,
.Xr roff 5 ,
.Xr tbl 5
.Sh HISTORY
The
.Nm
language first appeared as a macro package for the roff typesetting
system in
.At v7 .
It was later rewritten by James Clark as a macro package for groff.
Eric S. Raymond wrote the extended
.Nm
macros for groff in 2007.
The stand-alone implementation that is part of the
.Xr mandoc 1
utility written by Kristaps Dzonsons appeared in
.Ox 4.6 .
.Sh AUTHORS
This
.Nm
reference was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv .
.Sh CAVEATS
Do not use this language.
Use
.Xr mdoc 5 ,
instead.
'\" te
.\" 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
.TH MAN 5 "Jan 30, 1995"
.SH NAME
man \- macros to format Reference Manual pages
```

```
.SH SYNOPSIS
.LP
.nf
\fBnroff\fR \fB-man\fR \fIfilename\fR...
.fi

.LP
.nf
\fBtroff\fR  \fB-man\fR \fIfilename\fR...
.fi

.SH DESCRIPTION
.sp
.LP
These macros are used to lay out the reference pages in this manual. Note: if
\fIfilename\fR contains format input for a preprocessor, the commands shown
above must be piped through the appropriate preprocessor. This is handled
automatically by the \fBman\fR(1) command. See the ''Conventions'' section.
.sp
.LP
Any text argument \fIt\fR may be zero to six words. Quotes may be used to
include SPACE characters in a "word".  If \fItext\fR is empty, the special
treatment is applied to the next input line with text to be printed. In this
way \fB\&.I\fR may be used to italicize a whole line, or \fB\&.SB\fR may be
used to make small bold letters.
.sp
.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 \fIi\fR are ens.
.sp
.LP
Type font and size are reset to default values before each paragraph, and after
processing font and size setting macros.
.sp
.LP
These strings are predefined by \fB-man\fR:
.sp
.ne 2
.na
\fB\fB\e*R\fR\fR
.ad
.RS 8n
‘\(rg’, ‘(Reg)’ in \fBnroff\fR.
.RE

.sp
.ne 2
.na
\fB\fB\e*S\fR\fR
.ad
.RS 8n
Change to default type size.
.RE

.SS "Requests"
.sp
.LP
* n.t.l. = next text line; p.i. = prevailing indent
.sp

.sp
.TS
c c c c
c c c c .
\fIRequest\fR   \fICause\fR     \fIIf no\fR     \fIExplanation\fR
```



```
        \fIBreak\fR     \fIArgument\fR
\fB\&.B \fR\fIt\fR      no      \fIt\fR=n.t.l.* Text is in bold font.
\fB\&.BI \fR\fIt\fR     no      \fIt\fR=n.t.l.  Join words, alternating bold and
\fB\&.BR \fR\fIt\fR     no      \fIt\fR=n.t.l.  Join words, alternating bold and
\fB\&.DT\fR     no      \&.5i 1i...     Restore default tabs.
\fB\&.HP \fR\fIi\fR     yes     \fIi\fR=p.i.*   T{
Begin paragraph with hanging indent. Set prevailing indent to \fIi\fR.
T}
\fB\&.I \fR\fIt\fR      no      \fIt\fR=n.t.l.  Text is italic.
\fB\&.IB \fR\fIt\fR     no      \fIt\fR=n.t.l.  Join words, alternating italic a
\fB\&.IP \fR\fIx i\fR   yes     \fIx\fR=""      Same as \fB\&.TP\fR with tag \fI
\fB\&.IR \fR\fIt\fR     no      \fIt\fR=n.t.l.  T{
Join words, alternating italic and roman.
T}
\fB\&.IX \fR\fIt\fR     no      -       Index macro, for SunSoft internal use.
\fB\&.LP\fR     yes     -       T{
Begin left-aligned paragraph. Set prevailing indent to .5i.
T}
\fB\&.P\fR      yes     -       Same as \fB\&.LP\fR.
\fB\&.PD \fR\fId\fR     no      \fId\fR=.4v     T{
Set vertical distance between paragraphs.
T}
\fB\&.PP\fR     yes     -       Same as \fB\&.LP\fR.
\fB\&.RE\fR     yes     -       T{
End of relative indent. Restores prevailing indent.
T}
\fB\&.RB \fR\fIt\fR     no      \fIt\fR=n.t.l.  Join words, alternating roman an
\fB\&.RI \fR\fIt\fR     no      \fIt\fR=n.t.l.  T{
Join words, alternating roman and italic.
T}
\fB\&.RS \fR\fIi\fR     yes     \fIi\fR=p.i.    T{
Start relative indent, increase indent by \fIi\fR. Sets prevailing indent to .5i
T}
\fB\&.SB \fR\fIt\fR     no      -       T{
Reduce size of text by 1 point, make text bold.
T}
\fB\&.SH \fR\fIt\fR     yes     -       Section Heading.
\fB\&.SM \fR\fIt\fR     no      \fIt\fR=n.t.l.  Reduce size of text by 1 point.
\fB\&.SS \fR\fIt\fR     yes     \fIt\fR=n.t.l.  Section Subheading.
\fB\&.TH \FR\FIN S "f d, m\fR"
\fB\&.TH \fR\fIn s d f m\fR     yes     -       T{
Begin reference page \fIn\fR, of of section \fIs\fR; \fId\fR is the date of the
T}
\fB\&.TP \fR\fIi\fR     yes     \fIi\fR=p.i.    T{
Begin indented paragraph, with the tag given on the next text line. Set prevaili
T}
\fB\&.TX \fR\fIt \fR\fIp\fR     no      -       T{
Resolve the title abbreviation \fIt\fR; join to punctuation mark (or text) \fIp\
T}
.TE

.SS "Conventions"
.sp
.LP
When formatting a manual page, \fBman\fR examines the first line to determine
whether it requires special processing. For example a first line consisting of:
.sp
.LP
\fB\&'\e" t\fR
.sp
.LP
indicates that the manual page must be run through the \fBtbl\fR(1)
preprocessor.
.sp
.LP
A typical manual page for a command or function is laid out as follows:
```

```
.sp
.ne 2
.na
\fB\&.TH\fI TITLE \FR[1-9]\FR " , "
.ad
.RS 23n
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.
.RE

.sp
.ne 2
.na
\fB\&.SH NAME\fR
.ad
.RS 23n
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 \fBtroff\fR(1) commands or escapes, and no macro requests.
It is used to generate the \fBwindex\fR database, which is used by the
\fBwhatis\fR(1) command.
.RE

.sp
.ne 2
.na
\fB\&.SH SYNOPSIS\fR
.ad
.RS 23n
.sp
.ne 2
.na
\fBCommands:\fR
.ad
.RS 13n
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.
.sp
Syntactic symbols appear in roman face:
.sp
.ne 2
.na
\fB[ ]\fR
.ad
.RS 13n
An argument, when surrounded by brackets is optional.
.RE

.sp
.ne 2
.na
\fB|\fR
.ad
.RS 13n
Arguments separated by a vertical bar are exclusive. You can supply only one
item from such a list.
.RE

.sp
.ne 2
.na
\fB\&.\|.\|.\fR
.ad
```



```
.RS 13n
Arguments followed by an ellipsis can be repeated. When an ellipsis follows a
bracketed set, the expression within the brackets can be repeated.
.RE

.RE

.sp
.ne 2
.na
\fBFunctions:\fR
.ad
.RS 14n
If required, the data declaration, or \fB#include\fR directive, is shown first,
followed by the  function declaration. Otherwise, the function declaration is
shown.
.RE

.RE

.sp
.ne 2
.na
\fB\&.SH DESCRIPTION\fR
.ad
.RS 23n
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?"
.sp
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.
.sp
If a command interprets either subcommands or an input grammar, its command
interface or input grammar is normally described in a \fBUSAGE\fR section,
which follows the \fBOPTIONS\fR section.  The  \fBDESCRIPTION\fR section only
describes the behavior of the command itself, not that of subcommands.
.RE

.sp
.ne 2
.na
\fB\&.SH OPTIONS\fR
.ad
.RS 23n
The list of options along with a description of how each affects the command’s
operation.
.RE

.sp
.ne 2
.na
\fB\&.SH RETURN VALUES\fR
.ad
.RS 23n
A list of the values the library routine will return to the calling  program
and the conditions that cause these values to be returned.
.RE

.sp
.ne 2
.na
\fB\&.SH EXIT STATUS\fR
```

```
.ad
.RS 23n
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.
.RE

.sp
.ne 2
.na
\fB\&.SH FILES\fR
.ad
.RS 23n
A list of files associated with the command or function.
.RE

.sp
.ne 2
.na
\fB\&.SH SEE ALSO\fR
.ad
.RS 23n
A comma-separated list of related manual pages, followed by references to other
published materials.
.RE

.sp
.ne 2
.na
\fB\&.SH DIAGNOSTICS\fR
.ad
.RS 23n
A list of diagnostic messages and an explanation of each.
.RE

.sp
.ne 2
.na
\fB\&.SH BUGS\fR
.ad
.RS 23n
A description of limitations, known defects, and possible problems associated
with the command or function.
.RE

.SH FILES
.sp
.ne 2
.na
\fB\fB/usr/share/lib/tmac/an\fR \fR
.ad
.RS 27n

.RE

.sp
.ne 2
.na
\fB\fB/usr/share/man/windex\fR\fR
.ad
.RS 27n

.RE

.SH SEE ALSO
.sp
.LP
```



```
\fBman\fR(1), \fBnroff\fR(1), \fBtroff\fR(1), \fBwhatis\fR(1)
.sp
.LP
Dale Dougherty and   Tim O’Reilly, \fIUnix\fR \fIText\fR \fIProcessing\fR
```

```
************************************************************
   26074 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/mandoc_char.5
import complete (hopefully)
************************************************************
```

```
.\"
.\" 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) 2003 Jason McIntyre <jmc@openbsd.org>
.\" Copyright (c) 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
.\" Copyright (c) 2011 Ingo Schwarze <schwarze@openbsd.org>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\"
.Dd Nov 23, 2011
.Dt MANDOC_CHAR 5
.Os
.Sh NAME
.Nm mandoc_char
.Nd mandoc special characters
.Sh DESCRIPTION
This page documents the
.Xr roff 5
escape sequences accepted by
.Xr mandoc 1
to represent special characters in
.Xr mdoc 5
and
.Xr man 5
documents.
.Pp
The rendering depends on the
.Xr mandoc 1
output mode; in ASCII output, most characters are completely
unintelligible.
For that reason, using any of the special characters documented here,
except those discussed in the
.Sx DESCRIPTION ,
is strongly discouraged; they are supported merely for backwards
compatibility with existing documents.
.Pp
In particular, in English manual pages, do not use special-character
escape sequences to represent national language characters in author
names; instead, provide ASCII transcriptions of the names.
.Ss Dashes and Hyphens
In typography there are different types of dashes of various width:
the hyphen (-),
the minus sign (\-),
the en-dash (\(en),
and the em-dash (\(em).
.Pp
Hyphens are used for adjectives;
to separate the two parts of a compound word;
or to separate a word across two successive lines of text.
The hyphen does not need to be escaped:
.Bd -unfilled -offset indent
```



```
blue-eyed
lorry-driver
.Ed
.Pp
The mathematical minus sign is used for negative numbers or subtraction.
It should be written as
.Sq \e- :
.Bd -unfilled -offset indent
a = 3 \e- 1;
b = \e-2;
.Ed
.Pp
The en-dash is used to separate the two elements of a range,
or can be used the same way as an em-dash.
It should be written as
.Sq \e(en :
.Bd -unfilled -offset indent
pp. 95\e(en97.
Go away \e(en or else!
.Ed
.Pp
The em-dash can be used to show an interruption
or can be used the same way as colons, semi-colons, or parentheses.
It should be written as
.Sq \e(em :
.Bd -unfilled -offset indent
Three things \e(em apples, oranges, and bananas.
This is not that \e(em rather, this is that.
.Ed
.Pp
Note:
hyphens, minus signs, and en-dashes look identical under normal ASCII output.
Other formats, such as PostScript, render them correctly,
with differing widths.
.Ss Spaces
To separate words in normal text, for indenting and alignment
in literal context, and when none of the following special cases apply,
just use the normal space character
.Pq Sq \  .
.Pp
When filling text, lines may be broken between words, i.e. at space
characters.
To prevent a line break between two particular words,
use the non-breaking space escape sequence
.Pq Sq \e~
instead of the normal space character.
For example, the input string
.Dq number\e~1
will be kept together as
.Dq number\~1
on the same output line.
.Pp
On request and macro lines, the normal space character serves as an
argument delimiter.
To include whitespace into arguments, quoting is usually the best choice.
In some cases, using either the non-breaking
.Pq Sq \e~
or the breaking
.Pq Sq \e\ \&
space escape sequence may be preferable.
To escape macro names and to protect whitespace at the end
of input lines, the zero-width space
.Pq Sq \e&
is often useful.
For example, in
.Xr mdoc 5 ,
```

```
a normal space character can be displayed in single quotes in either
of the following ways:
.Pp
.Dl .Sq \(dq \(dq
.Dl .Sq \e \e&
.Ss Quotes
On request and macro lines, the double-quote character
.Pq Sq \(dq
is handled specially to allow quoting.
One way to prevent this special handling is by using the
.Sq \e(dq
escape sequence.
.Pp
Note that on text lines, literal double-quote characters can be used
verbatim.
All other quote-like characters can be used verbatim as well,
even on request and macro lines.
.Ss Periods
The period
.Pq Sq \&.
is handled specially at the beginning of an input line,
where it introduces a
.Xr roff 5
request or a macro, and when appearing alone as a macro argument in
.Xr mdoc 5 .
In such situations, prepend a zero-width space
.Pq Sq \e&.
to make it behave like normal text.
.Pp
Do not use the
.Sq \e.
escape sequence.
It does not prevent special handling of the period.
.Ss Backslashes
To include a literal backslash
.Pq Sq \e
into the output, use the
.Pq Sq \ee
escape sequence.
.Pp
Note that doubling it
.Pq Sq \e\e
is not the right way to output a backslash.
Because
.Xr mandoc 1
does not implement full
.Xr roff 5
functionality, it may work with
.Xr mandoc 1 ,
but it may have weird effects on complete
.Xr roff 5
implementations.
.Sh SPECIAL CHARACTERS
Special characters are encoded as
.Sq \eX
.Pq for a one-character escape ,
.Sq \e(XX
.Pq two-character ,
and
.Sq \e[N]
.Pq N-character .
For details, see the
.Em Special Characters
subsection of the
.Xr roff 5
manual.
```



```
.Pp
Spacing:
.Bl -column "Input" "Description" -offset indent -compact
.It Em Input Ta Em Description
.It \e~      Ta non-breaking, non-collapsing space
.It \e       Ta breaking, non-collapsing n-width space
.It \e^      Ta zero-width space
.It \e%      Ta zero-width space
.It \e&      Ta zero-width space
.It \e|      Ta zero-width space
.It \e0      Ta breaking, non-collapsing digit-width space
.It \ec      Ta removes any trailing space (if applicable)
.El
.Pp
Lines:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(ba    Ta \(ba        Ta bar
.It \e(br    Ta \(br        Ta box rule
.It \e(ul    Ta \(ul        Ta underscore
.It \e(rl    Ta \(rl        Ta overline
.It \e(bb    Ta \(bb        Ta broken bar
.It \e(sl    Ta \(sl        Ta forward slash
.It \e(rs    Ta \(rs        Ta backward slash
.El
.Pp
Text markers:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(ci    Ta \(ci        Ta circle
.It \e(bu    Ta \(bu        Ta bullet
.It \e(dd    Ta \(dd        Ta double dagger
.It \e(dg    Ta \(dg        Ta dagger
.It \e(lz    Ta \(lz        Ta lozenge
.It \e(sq    Ta \(sq        Ta white square
.It \e(ps    Ta \(ps        Ta paragraph
.It \e(sc    Ta \(sc        Ta section
.It \e(lh    Ta \(lh        Ta left hand
.It \e(rh    Ta \(rh        Ta right hand
.It \e(at    Ta \(at        Ta at
.It \e(sh    Ta \(sh        Ta hash (pound)
.It \e(CR    Ta \(CR        Ta carriage return
.It \e(OK    Ta \(OK        Ta check mark
.El
.Pp
Legal symbols:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(co    Ta \(co        Ta copyright
.It \e(rg    Ta \(rg        Ta registered
.It \e(tm    Ta \(tm        Ta trademarked
.El
.Pp
Punctuation:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(em    Ta \(em        Ta em-dash
.It \e(en    Ta \(en        Ta en-dash
.It \e(hy    Ta \(hy        Ta hyphen
.It \ee      Ta \e          Ta back-slash
.It \e.      Ta \.          Ta period
.It \e(r!    Ta \(r!        Ta upside-down exclamation
.It \e(r?    Ta \(r?        Ta upside-down question
.El
.Pp
Quotes:
```

```
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(Bq    Ta \(Bq        Ta right low double-quote
.It \e(bq    Ta \(bq        Ta right low single-quote
.It \e(lq    Ta \(lq        Ta left double-quote
.It \e(rq    Ta \(rq        Ta right double-quote
.It \e(oq    Ta \(oq        Ta left single-quote
.It \e(cq    Ta \(cq        Ta right single-quote
.It \e(aq    Ta \(aq        Ta apostrophe quote (text)
.It \e(dq    Ta \(dq        Ta double quote (text)
.It \e(Fo    Ta \(Fo        Ta left guillemet
.It \e(Fc    Ta \(Fc        Ta right guillemet
.It \e(fo    Ta \(fo        Ta left single guillemet
.It \e(fc    Ta \(fc        Ta right single guillemet
.El
.Pp
Brackets:
.Bl -column "xxbracketrightbpx" Rendered Description -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(lB    Ta \(lB        Ta left bracket
.It \e(rB    Ta \(rB        Ta right bracket
.It \e(lC    Ta \(lC        Ta left brace
.It \e(rC    Ta \(rC        Ta right brace
.It \e(la    Ta \(la        Ta left angle
.It \e(ra    Ta \(ra        Ta right angle
.It \e(bv    Ta \(bv        Ta brace extension
.It \e[braceex] Ta \[braceex] Ta brace extension
.It \e[bracketlefttp] Ta \[bracketlefttp] Ta top-left hooked bracket
.It \e[bracketleftbp] Ta \[bracketleftbp] Ta bottom-left hooked bracket
.It \e[bracketleftex] Ta \[bracketleftex] Ta left hooked bracket extension
.It \e[bracketrighttp] Ta \[bracketrighttp] Ta top-right hooked bracket
.It \e[bracketrightbp] Ta \[bracketrightbp] Ta bottom-right hooked bracket
.It \e[bracketrightex] Ta \[bracketrightex] Ta right hooked bracket extension
.It \e(lt    Ta \(lt        Ta top-left hooked brace
.It \e[bracelefttp] Ta \[bracelefttp] Ta top-left hooked brace
.It \e(lk    Ta \(lk        Ta mid-left hooked brace
.It \e[braceleftmid] Ta \[braceleftmid] Ta mid-left hooked brace
.It \e(lb    Ta \(lb        Ta bottom-left hooked brace
.It \e[braceleftbp] Ta \[braceleftbp] Ta bottom-left hooked brace
.It \e[braceleftex] Ta \[braceleftex] Ta left hooked brace extension
.It \e(rt    Ta \(rt        Ta top-left hooked brace
.It \e[bracerighttp] Ta \[bracerighttp] Ta top-right hooked brace
.It \e(rk    Ta \(rk        Ta mid-right hooked brace
.It \e[bracerightmid] Ta \[bracerightmid] Ta mid-right hooked brace
.It \e(rb    Ta \(rb        Ta bottom-right hooked brace
.It \e[bracerightbp] Ta \[bracerightbp] Ta bottom-right hooked brace
.It \e[bracerightex] Ta \[bracerightex] Ta right hooked brace extension
.It \e[parenlefttp] Ta \[parenlefttp] Ta top-left hooked parenthesis
.It \e[parenleftbp] Ta \[parenleftbp] Ta bottom-left hooked parenthesis
.It \e[parenleftex] Ta \[parenleftex] Ta left hooked parenthesis extension
.It \e[parenrighttp] Ta \[parenrighttp] Ta top-right hooked parenthesis
.It \e[parenrightbp] Ta \[parenrightbp] Ta bottom-right hooked parenthesis
.It \e[parenrightex] Ta \[parenrightex] Ta right hooked parenthesis extension
.El
.Pp
Arrows:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(<-    Ta \(<-        Ta left arrow
.It \e(->    Ta \(->        Ta right arrow
.It \e(<>    Ta \(<>        Ta left-right arrow
.It \e(da    Ta \(da        Ta down arrow
.It \e(ua    Ta \(ua        Ta up arrow
.It \e(va    Ta \(va        Ta up-down arrow
.It \e(lA    Ta \(lA        Ta left double-arrow
.It \e(rA    Ta \(rA        Ta right double-arrow
```



```
.It \e(hA    Ta \(hA        Ta left-right double-arrow
.It \e(uA    Ta \(uA        Ta up double-arrow
.It \e(dA    Ta \(dA        Ta down double-arrow
.It \e(vA    Ta \(vA        Ta up-down double-arrow
.El
.Pp
Logical:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(AN    Ta \(AN        Ta logical and
.It \e(OR    Ta \(OR        Ta logical or
.It \e(no    Ta \(no        Ta logical not
.It \e[tno]  Ta \[tno]      Ta logical not (text)
.It \e(te    Ta \(te        Ta existential quantifier
.It \e(fa    Ta \(fa        Ta universal quantifier
.It \e(st    Ta \(st        Ta such that
.It \e(tf    Ta \(tf        Ta therefore
.It \e(3d    Ta \(3d        Ta therefore
.It \e(or    Ta \(or        Ta bitwise or
.El
.Pp
Mathematical:
.Bl -column "xxcoproductxx" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(pl    Ta \(pl        Ta plus
.It \e(mi    Ta \(mi        Ta minus
.It \e-      Ta \-          Ta minus (text)
.It \e(-+    Ta \(-+        Ta minus-plus
.It \e(+-    Ta \(+-        Ta plus-minus
.It \e[t+-]  Ta \[t+-]      Ta plus-minus (text)
.It \e(pc    Ta \(pc        Ta centre-dot
.It \e(mu    Ta \(mu        Ta multiply
.It \e[tmu]  Ta \[tmu]      Ta multiply (text)
.It \e(c*    Ta \(c*        Ta circle-multiply
.It \e(c+    Ta \(c+        Ta circle-plus
.It \e(di    Ta \(di        Ta divide
.It \e[tdi]  Ta \[tdi]      Ta divide (text)
.It \e(f/    Ta \(f/        Ta fraction
.It \e(**    Ta \(**        Ta asterisk
.It \e(<=    Ta \(<=        Ta less-than-equal
.It \e(>=    Ta \(>=        Ta greater-than-equal
.It \e(<<    Ta \(<<        Ta much less
.It \e(>>    Ta \(>>        Ta much greater
.It \e(eq    Ta \(eq        Ta equal
.It \e(!=    Ta \(!=        Ta not equal
.It \e(==    Ta \(==        Ta equivalent
.It \e(ne    Ta \(ne        Ta not equivalent
.It \e(=~    Ta \(=~        Ta congruent
.It \e(-~    Ta \(-~        Ta asymptotically congruent
.It \e(ap    Ta \(ap        Ta asymptotically similar
.It \e(~~    Ta \(~~        Ta approximately similar
.It \e(~=    Ta \(~=        Ta approximately equal
.It \e(pt    Ta \(pt        Ta proportionate
.It \e(es    Ta \(es        Ta empty set
.It \e(mo    Ta \(mo        Ta element
.It \e(nm    Ta \(nm        Ta not element
.It \e(sb    Ta \(sb        Ta proper subset
.It \e(nb    Ta \(nb        Ta not subset
.It \e(sp    Ta \(sp        Ta proper superset
.It \e(nc    Ta \(nc        Ta not superset
.It \e(ib    Ta \(ib        Ta reflexive subset
.It \e(ip    Ta \(ip        Ta reflexive superset
.It \e(ca    Ta \(ca        Ta intersection
.It \e(cu    Ta \(cu        Ta union
.It \e(/_    Ta \(/_        Ta angle
.It \e(pp    Ta \(pp        Ta perpendicular
```

```
.It \e(is    Ta \(is        Ta integral
.It \e[integral] Ta \[integral] Ta integral
.It \e[sum]    Ta \[sum]   Ta summation
.It \e[product] Ta \[product] Ta product
.It \e[coproduct] Ta \[coproduct] Ta coproduct
.It \e(gr    Ta \(gr        Ta gradient
.It \e(sr    Ta \(sr        Ta square root
.It \e[sqrt] Ta \[sqrt]     Ta square root
.It \e(lc    Ta \(lc        Ta left-ceiling
.It \e(rc    Ta \(rc        Ta right-ceiling
.It \e(lf    Ta \(lf        Ta left-floor
.It \e(rf    Ta \(rf        Ta right-floor
.It \e(if    Ta \(if        Ta infinity
.It \e(Ah    Ta \(Ah        Ta aleph
.It \e(Im    Ta \(Im        Ta imaginary
.It \e(Re    Ta \(Re        Ta real
.It \e(pd    Ta \(pd        Ta partial differential
.It \e(-h    Ta \(-h        Ta Planck constant over 2\(*p
.It \e[12]   Ta \[12]       Ta one-half
.It \e[14]   Ta \[14]       Ta one-fourth
.It \e[34]   Ta \[34]       Ta three-fourths
.El
.Pp
Ligatures:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(ff    Ta \(ff        Ta ff ligature
.It \e(fi    Ta \(fi        Ta fi ligature
.It \e(fl    Ta \(fl        Ta fl ligature
.It \e(Fi    Ta \(Fi        Ta ffi ligature
.It \e(Fl    Ta \(Fl        Ta ffl ligature
.It \e(AE    Ta \(AE        Ta AE
.It \e(ae    Ta \(ae        Ta ae
.It \e(OE    Ta \(OE        Ta OE
.It \e(oe    Ta \(oe        Ta oe
.It \e(ss    Ta \(ss        Ta German eszett
.It \e(IJ    Ta \(IJ        Ta IJ ligature
.It \e(ij    Ta \(ij        Ta ij ligature
.El
.Pp
Accents:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(a"    Ta \(a"        Ta Hungarian umlaut
.It \e(a-    Ta \(a-        Ta macron
.It \e(a.    Ta \(a.        Ta dotted
.It \e(a^    Ta \(a^        Ta circumflex
.It \e(aa    Ta \(aa        Ta acute
.It \e'      Ta \'          Ta acute
.It \e(ga    Ta \(ga        Ta grave
.It \e`      Ta \`          Ta grave
.It \e(ab    Ta \(ab        Ta breve
.It \e(ac    Ta \(ac        Ta cedilla
.It \e(ad    Ta \(ad        Ta dieresis
.It \e(ah    Ta \(ah        Ta caron
.It \e(ao    Ta \(ao        Ta ring
.It \e(a~    Ta \(a~        Ta tilde
.It \e(ho    Ta \(ho        Ta ogonek
.It \e(ha    Ta \(ha        Ta hat (text)
.It \e(ti    Ta \(ti        Ta tilde (text)
.El
.Pp
Accented letters:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e('A    Ta \('A        Ta acute A
```



```
.It \e('E    Ta \('E        Ta acute E
.It \e('I    Ta \('I        Ta acute I
.It \e('O    Ta \('O        Ta acute O
.It \e('U    Ta \('U        Ta acute U
.It \e('a    Ta \('a        Ta acute a
.It \e('e    Ta \('e        Ta acute e
.It \e('i    Ta \('i        Ta acute i
.It \e('o    Ta \('o        Ta acute o
.It \e('u    Ta \('u        Ta acute u
.It \e(`A    Ta \(`A        Ta grave A
.It \e(`E    Ta \(`E        Ta grave E
.It \e(`I    Ta \(`I        Ta grave I
.It \e(`O    Ta \(`O        Ta grave O
.It \e(`U    Ta \(`U        Ta grave U
.It \e(`a    Ta \(`a        Ta grave a
.It \e(`e    Ta \(`e        Ta grave e
.It \e(`i    Ta \(`i        Ta grave i
.It \e(`o    Ta \(`i        Ta grave o
.It \e(`u    Ta \(`u        Ta grave u
.It \e(~A    Ta \(~A        Ta tilde A
.It \e(~N    Ta \(~N        Ta tilde N
.It \e(~O    Ta \(~O        Ta tilde O
.It \e(~a    Ta \(~a        Ta tilde a
.It \e(~n    Ta \(~n        Ta tilde n
.It \e(~o    Ta \(~o        Ta tilde o
.It \e(:A    Ta \(:A        Ta dieresis A
.It \e(:E    Ta \(:E        Ta dieresis E
.It \e(:I    Ta \(:I        Ta dieresis I
.It \e(:O    Ta \(:O        Ta dieresis O
.It \e(:U    Ta \(:U        Ta dieresis U
.It \e(:a    Ta \(:a        Ta dieresis a
.It \e(:e    Ta \(:e        Ta dieresis e
.It \e(:i    Ta \(:i        Ta dieresis i
.It \e(:o    Ta \(:o        Ta dieresis o
.It \e(:u    Ta \(:u        Ta dieresis u
.It \e(:y    Ta \(:y        Ta dieresis y
.It \e(^A    Ta \(^A        Ta circumflex A
.It \e(^E    Ta \(^E        Ta circumflex E
.It \e(^I    Ta \(^I        Ta circumflex I
.It \e(^O    Ta \(^O        Ta circumflex O
.It \e(^U    Ta \(^U        Ta circumflex U
.It \e(^a    Ta \(^a        Ta circumflex a
.It \e(^e    Ta \(^e        Ta circumflex e
.It \e(^i    Ta \(^i        Ta circumflex i
.It \e(^o    Ta \(^o        Ta circumflex o
.It \e(^u    Ta \(^u        Ta circumflex u
.It \e(,C    Ta \(,C        Ta cedilla C
.It \e(,c    Ta \(,c        Ta cedilla c
.It \e(/L    Ta \(/L        Ta stroke L
.It \e(/l    Ta \(/l        Ta stroke l
.It \e(/O    Ta \(/O        Ta stroke O
.It \e(/o    Ta \(/o        Ta stroke o
.It \e(oA    Ta \(oA        Ta ring A
.It \e(oa    Ta \(oa        Ta ring a
.El
.Pp
Special letters:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(-D    Ta \(-D        Ta Eth
.It \e(Sd    Ta \(Sd        Ta eth
.It \e(TP    Ta \(TP        Ta Thorn
.It \e(Tp    Ta \(Tp        Ta thorn
.It \e(.i    Ta \(.i        Ta dotless i
.It \e(.j    Ta \(.j        Ta dotless j
.El
```

```
.Pp
Currency:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(Do    Ta \(Do        Ta dollar
.It \e(ct    Ta \(ct        Ta cent
.It \e(Eu    Ta \(Eu        Ta Euro symbol
.It \e(eu    Ta \(eu        Ta Euro symbol
.It \e(Ye    Ta \(Ye        Ta yen
.It \e(Po    Ta \(Po        Ta pound
.It \e(Cs    Ta \(Cs        Ta Scandinavian
.It \e(Fn    Ta \(Fn        Ta florin
.El
.Pp
Units:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(de    Ta \(de        Ta degree
.It \e(%0    Ta \(%0        Ta per-thousand
.It \e(fm    Ta \(fm        Ta minute
.It \e(sd    Ta \(sd        Ta second
.It \e(mc    Ta \(mc        Ta micro
.El
.Pp
Greek letters:
.Bl -column "Input" "Rendered" "Description" -offset indent -compact
.It Em Input Ta Em Rendered Ta Em Description
.It \e(*A    Ta \(*A        Ta Alpha
.It \e(*B    Ta \(*B        Ta Beta
.It \e(*G    Ta \(*G        Ta Gamma
.It \e(*D    Ta \(*D        Ta Delta
.It \e(*E    Ta \(*E        Ta Epsilon
.It \e(*Z    Ta \(*Z        Ta Zeta
.It \e(*Y    Ta \(*Y        Ta Eta
.It \e(*H    Ta \(*H        Ta Theta
.It \e(*I    Ta \(*I        Ta Iota
.It \e(*K    Ta \(*K        Ta Kappa
.It \e(*L    Ta \(*L        Ta Lambda
.It \e(*M    Ta \(*M        Ta Mu
.It \e(*N    Ta \(*N        Ta Nu
.It \e(*C    Ta \(*C        Ta Xi
.It \e(*O    Ta \(*O        Ta Omicron
.It \e(*P    Ta \(*P        Ta Pi
.It \e(*R    Ta \(*R        Ta Rho
.It \e(*S    Ta \(*S        Ta Sigma
.It \e(*T    Ta \(*T        Ta Tau
.It \e(*U    Ta \(*U        Ta Upsilon
.It \e(*F    Ta \(*F        Ta Phi
.It \e(*X    Ta \(*X        Ta Chi
.It \e(*Q    Ta \(*Q        Ta Psi
.It \e(*W    Ta \(*W        Ta Omega
.It \e(*a    Ta \(*a        Ta alpha
.It \e(*b    Ta \(*b        Ta beta
.It \e(*g    Ta \(*g        Ta gamma
.It \e(*d    Ta \(*d        Ta delta
.It \e(*e    Ta \(*e        Ta epsilon
.It \e(*z    Ta \(*z        Ta zeta
.It \e(*y    Ta \(*y        Ta eta
.It \e(*h    Ta \(*h        Ta theta
.It \e(*i    Ta \(*i        Ta iota
.It \e(*k    Ta \(*k        Ta kappa
.It \e(*l    Ta \(*l        Ta lambda
.It \e(*m    Ta \(*m        Ta mu
.It \e(*n    Ta \(*n        Ta nu
.It \e(*c    Ta \(*c        Ta xi
.It \e(*o    Ta \(*o        Ta omicron
```



```
.It \e(*p    Ta \(*p        Ta pi
.It \e(*r    Ta \(*r        Ta rho
.It \e(*s    Ta \(*s        Ta sigma
.It \e(*t    Ta \(*t        Ta tau
.It \e(*u    Ta \(*u        Ta upsilon
.It \e(*f    Ta \(*f        Ta phi
.It \e(*x    Ta \(*x        Ta chi
.It \e(*q    Ta \(*q        Ta psi
.It \e(*w    Ta \(*w        Ta omega
.It \e(+h    Ta \(+h        Ta theta variant
.It \e(+f    Ta \(+f        Ta phi variant
.It \e(+p    Ta \(+p        Ta pi variant
.It \e(+e    Ta \(+e        Ta epsilon variant
.It \e(ts    Ta \(ts        Ta sigma terminal
.El
.Sh PREDEFINED STRINGS
Predefined strings are inherited from the macro packages of historical
troff implementations.
They are
.Em not recommended
for use, as they differ across implementations.
Manuals using these predefined strings are almost certainly not
portable.
.Pp
Their syntax is similar to special characters, using
.Sq \e*X
.Pq for a one-character escape ,
.Sq \e*(XX
.Pq two-character ,
and
.Sq \e*[N]
.Pq N-character .
For details, see the
.Em Predefined Strings
subsection of the
.Xr roff 5
manual.
.Bl -column "Input" "Rendered" "Description" -offset indent
.It Em Input Ta Em Rendered Ta Em Description
.It \e*(Ba   Ta \*(Ba       Ta vertical bar
.It \e*(Ne   Ta \*(Ne       Ta not equal
.It \e*(Ge   Ta \*(Ge       Ta greater-than-equal
.It \e*(Le   Ta \*(Le       Ta less-than-equal
.It \e*(Gt   Ta \*(Gt       Ta greater-than
.It \e*(Lt   Ta \*(Lt       Ta less-than
.It \e*(Pm   Ta \*(Pm       Ta plus-minus
.It \e*(If   Ta \*(If       Ta infinity
.It \e*(Pi   Ta \*(Pi       Ta pi
.It \e*(Na   Ta \*(Na       Ta NaN
.It \e*(Am   Ta \*(Am       Ta ampersand
.It \e*R     Ta \*R         Ta restricted mark
.It \e*(Tm   Ta \*(Tm       Ta trade mark
.It \e*q     Ta \*q         Ta double-quote
.It \e*(Rq   Ta \*(Rq       Ta right-double-quote
.It \e*(Lq   Ta \*(Lq       Ta left-double-quote
.It \e*(lp   Ta \*(lp       Ta right-parenthesis
.It \e*(rp   Ta \*(rp       Ta left-parenthesis
.It \e*(lq   Ta \*(lq       Ta left double-quote
.It \e*(rq   Ta \*(rq       Ta right double-quote
.It \e*(ua   Ta \*(ua       Ta up arrow
.It \e*(va   Ta \*(va       Ta up-down arrow
.It \e*(<=   Ta \*(<=       Ta less-than-equal
.It \e*(>=   Ta \*(>=       Ta greater-than-equal
.It \e*(aa   Ta \*(aa       Ta acute
.It \e*(ga   Ta \*(ga       Ta grave
.It \e*(Px   Ta \*(Px       Ta POSIX standard name
```

```
.It \e*(Ai   Ta \*(Ai       Ta ANSI standard name
.El
.Sh UNICODE CHARACTERS
The escape sequence
.Pp
.Dl \e[uXXXX]
.Pp
is interpreted as a Unicode codepoint.
The codepoint must be in the range above U+0080 and less than U+10FFFF.
For compatibility, points must be zero-padded to four characters; if
greater than four characters, no zero padding is allowed.
Unicode surrogates are not allowed.
.\" .Pp
.\" Unicode glyphs attenuate to the
.\" .Sq \&?
.\" character if invalid or not rendered by current output media.
.Sh NUMBERED CHARACTERS
For backward compatibility with existing manuals,
.Xr mandoc 1
also supports the
.Pp
.Dl \eN\(aq Ns Ar number Ns \(aq
.Pp
escape sequence, inserting the character
.Ar number
from the current character set into the output.
Of course, this is inherently non-portable and is already marked
as deprecated in the Heirloom roff manual.
For example, do not use \eN’34’, use \e(dq, or even the plain
.Sq \(dq
character where possible.
.Sh COMPATIBILITY
This section documents compatibility between mandoc and other other
troff implementations, at this time limited to GNU troff
.Pq Qq groff .
.Pp
.Bl -dash -compact
.It
The \eN\(aq\(aq escape sequence is limited to printable characters; in
groff, it accepts arbitrary character numbers.
.It
In
.Fl T Ns Cm ascii ,
the
\e(ss, \e(nm, \e(nb, \e(nc, \e(ib, \e(ip, \e(pp, \e[sum], \e[product],
\e[coproduct], \e(gr, \e(\-h, and \e(a. special characters render
differently between mandoc and groff.
.It
In
.Fl T Ns Cm html
and
.Fl T Ns Cm xhtml ,
the \e(~=, \e(nb, and \e(nc special characters render differently
between mandoc and groff.
.It
The
.Fl T Ns Cm ps
and
.Fl T Ns Cm pdf
modes format like
.Fl T Ns Cm ascii
instead of rendering glyphs as in groff.
.It
The \e[radicalex], \e[sqrtex], and \e(ru special characters have been omitted
from mandoc either because they are poorly documented or they have no
known representation.
```



```
.El
.Sh SEE ALSO
.Xr mandoc 1 ,
.Xr man 5 ,
.Xr mdoc 5 ,
.Xr roff 5
.Sh AUTHORS
The
.Nm
manual page was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv .
.Sh CAVEATS
The
.Sq \e*(Ba
escape mimics the behaviour of the
.Sq \&|
character in
.Xr mdoc 5 ;
thus, if you wish to render a vertical bar with no side effects, use
the
.Sq \e(ba
escape.
```

```
**********************************************************
   23887 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/mandoc_roff.5
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
.\" Copyright (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
.\"
.Dd Jul 13, 2014
.Dt MANDOC_ROFF 5
.Os
.Sh NAME
.Nm mandoc_roff
.Nd roff language reference for mandoc
.Sh DESCRIPTION
The
.Nm roff
language is a general purpose text formatting language.
Since traditional implementations of the
.Xr mdoc 5
and
.Xr man 5
manual formatting languages are based on it,
many real-world manuals use small numbers of
.Nm
requests intermixed with their
.Xr mdoc 5
or
.Xr man 5
code.
To properly format such manuals, the
.Xr mandoc 1
utility supports a tiny subset of
.Nm
requests.
Only these requests supported by
.Xr mandoc 1
are documented in the present manual,
together with the basic language syntax shared by
.Nm ,
.Xr mdoc 5 ,
and
.Xr man 5 .
For complete
.Nm
manuals, consult the
.Sx SEE ALSO
section.
.Pp
Input lines beginning with the control character
```



```
.Sq \&.
are parsed for requests and macros.
Such lines are called
.Dq request lines
or
.Dq macro lines ,
respectively.
Requests change the processing state and manipulate the formatting;
some macros also define the document structure and produce formatted
output.
The single quote
.Pq Qq \(aq
is accepted as an alternative control character,
treated by
.Xr mandoc 1
just like
.Ql \&.
.Pp
Lines not beginning with control characters are called
.Dq text lines .
They provide free-form text to be printed; the formatting of the text
depends on the respective processing context.
.Sh LANGUAGE SYNTAX
.Nm
documents may contain only graphable 7-bit ASCII characters, the space
character, and, in certain circumstances, the tab character.
The back-space character
.Sq \e
indicates the start of an escape sequence for
.Sx Comments ,
.Sx Special Characters ,
.Sx Predefined Strings ,
and
user-defined strings defined using the
.Sx ds
request.
.Ss Comments
Text following an escaped double-quote
.Sq \e\(dq ,
whether in a request, macro, or text line, is ignored to the end of the line.
A request line beginning with a control character and comment escape
.Sq \&.\e\(dq
is also ignored.
Furthermore, request lines with only a control character and optional
trailing whitespace are stripped from input.
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.\e\(dq This is a comment line.
\&.\e\(dq The next line is ignored:
\&.
\&.Sh EXAMPLES \e\(dq This is a comment, too.
\&example text \e\(dq And so is this.
.Ed
.Ss Special Characters
Special characters are used to encode special glyphs and are rendered
differently across output media.
They may occur in request, macro, and text lines.
Sequences begin with the escape character
.Sq \e
followed by either an open-parenthesis
.Sq \&(
for two-character sequences; an open-bracket
.Sq \&[
for n-character sequences (terminated at a close-bracket
.Sq \&] ) ;
```

```
or a single one character sequence.
.Pp
Examples:
.Bl -tag -width Ds -offset indent -compact
.It Li \e(em
Two-letter em dash escape.
.It Li \ee
One-letter backslash escape.
.El
.Pp
See
.Xr mandoc_char 5
for a complete list.
.Ss Text Decoration
Terms may be text-decorated using the
.Sq \ef
escape followed by an indicator: B (bold), I (italic), R (regular), or P
(revert to previous mode).
A numerical representation 3, 2, or 1 (bold, italic, and regular,
respectively) may be used instead.
The indicator or numerical representative may be preceded by C
(constant-width), which is ignored.
.Pp
Examples:
.Bl -tag -width Ds -offset indent -compact
.It Li \efBbold\efR
Write in bold, then switch to regular font mode.
.It Li \efIitalic\efP
Write in italic, then return to previous font mode.
.El
.Pp
Text decoration is
.Em not
recommended for
.Xr mdoc 5 ,
which encourages semantic annotation.
.Ss Predefined Strings
Predefined strings, like
.Sx Special Characters ,
mark special output glyphs.
Predefined strings are escaped with the slash-asterisk,
.Sq \e* :
single-character
.Sq \e*X ,
two-character
.Sq \e*(XX ,
and N-character
.Sq \e*[N] .
.Pp
Examples:
.Bl -tag -width Ds -offset indent -compact
.It Li \e*(Am
Two-letter ampersand predefined string.
.It Li \e*q
One-letter double-quote predefined string.
.El
.Pp
Predefined strings are not recommended for use,
as they differ across implementations.
Those supported by
.Xr mandoc 1
are listed in
.Xr mandoc_char 5 .
Manuals using these predefined strings are almost certainly not portable.
.Ss Whitespace
Whitespace consists of the space character.
```



```
In text lines, whitespace is preserved within a line.
In request and macro lines, whitespace delimits arguments and is discarded.
.Pp
Unescaped trailing spaces are stripped from text line input unless in a
literal context.
In general, trailing whitespace on any input line is discouraged for
reasons of portability.
In the rare case that a blank character is needed at the end of an
input line, it may be forced by
.Sq \e\ \e& .
.Pp
Literal space characters can be produced in the output
using escape sequences.
In macro lines, they can also be included in arguments using quotation; see
.Sx MACRO SYNTAX
for details.
.Pp
Blank text lines, which may include whitespace, are only permitted
within literal contexts.
If the first character of a text line is a space, that line is printed
with a leading newline.
.Ss Scaling Widths
Many requests and macros support scaled widths for their arguments.
The syntax for a scaled width is
.Sq Li [+-]?[0-9]*.[0-9]*[:unit:] ,
where a decimal must be preceded or followed by at least one digit.
Negative numbers, while accepted, are truncated to zero.
.Pp
The following scaling units are accepted:
.Pp
.Bl -tag -width Ds -offset indent -compact
.It c
centimetre
.It i
inch
.It P
pica (~1/6 inch)
.It p
point (~1/72 inch)
.It f
synonym for
.Sq u
.It v
default vertical span
.It m
width of rendered
.Sq m
.Pq em
character
.It n
width of rendered
.Sq n
.Pq en
character
.It u
default horizontal span
.It M
mini-em (~1/100 em)
.El
.Pp
Using anything other than
.Sq m ,
.Sq n ,
.Sq u ,
or
.Sq v
```

```
260 is necessarily non-portable across output media.
261 See
262 .Sx COMPATIBILITY .
263 .Pp
264 If a scaling unit is not provided, the numerical value is interpreted
265 under the default rules of
266 .Sq v
267 for vertical spaces and
268 .Sq u
269 for horizontal ones.
270 .Pp
271 Examples:
272 .Bl -tag -width ".Bl -tag -width 2i" -offset indent -compact
273 .It Li \&.Bl -tag -width 2i
274 two-inch tagged list indentation in
275 .Xr mdoc 5
276 .It Li \&.HP 2i
277 two-inch tagged list indentation in
278 .Xr man 5
279 .It Li \&.sp 2v
280 two vertical spaces
281 .El
282 .Ss Sentence Spacing
283 Each sentence should terminate at the end of an input line.
284 By doing this, a formatter will be able to apply the proper amount of
285 spacing after the end of sentence (unescaped) period, exclamation mark,
286 or question mark followed by zero or more non-sentence closing
287 delimiters
288 .Po
289 .Sq \&) ,
290 .Sq \&] ,
291 .Sq \&' ,
292 .Sq \&"
293 .Pc .
294 .Pp
295 The proper spacing is also intelligently preserved if a sentence ends at
296 the boundary of a macro line.
297 .Pp
298 Examples:
299 .Bd -literal -offset indent -compact
300 Do not end sentences mid-line like this.  Instead,
301 end a sentence like this.
302 A macro would end like this:
303 \&.Xr mandoc 1 \&.
304 .Ed
305 .Sh REQUEST SYNTAX
306 A request or macro line consists of:
307 .Pp
308 .Bl -enum -compact
309 .It
310 the control character
311 .Sq \&.
312 or
313 .Sq \(aq
314 at the beginning of the line,
315 .It
316 optionally an arbitrary amount of whitespace,
317 .It
318 the name of the request or the macro, which is one word of arbitrary
319 length, terminated by whitespace,
320 .It
321 and zero or more arguments delimited by whitespace.
322 .El
323 .Pp
324 Thus, the following request lines are all equivalent:
325 .Bd -literal -offset indent
```



```
326 \&.ig end
327 \&.ig    end
328 \&.   ig end
329 .Ed
330 .Sh MACRO SYNTAX
331 Macros are provided by the
332 .Xr mdoc 5
333 and
334 .Xr man 5
335 languages and can be defined by the
336 .Sx \&de
337 request.
338 When called, they follow the same syntax as requests, except that
339 macro arguments may optionally be quoted by enclosing them
340 in double quote characters
341 .Pq Sq \(dq .
342 Quoted text, even if it contains whitespace or would cause
343 a macro invocation when unquoted, is always considered literal text.
344 Inside quoted text, pairs of double quote characters
345 .Pq Sq Qq
346 resolve to single double quote characters.
347 .Pp
348 To be recognised as the beginning of a quoted argument, the opening
349 quote character must be preceded by a space character.
350 A quoted argument extends to the next double quote character that is not
351 part of a pair, or to the end of the input line, whichever comes earlier.
352 Leaving out the terminating double quote character at the end of the line
353 is discouraged.
354 For clarity, if more arguments follow on the same input line,
355 it is recommended to follow the terminating double quote character
356 by a space character; in case the next character after the terminating
357 double quote character is anything else, it is regarded as the beginning
358 of the next, unquoted argument.
359 .Pp
360 Both in quoted and unquoted arguments, pairs of backslashes
361 .Pq Sq \e\e
362 resolve to single backslashes.
363 In unquoted arguments, space characters can alternatively be included
364 by preceding them with a backslash
365 .Pq Sq \e\~ ,
366 but quoting is usually better for clarity.
367 .Pp
368 Examples:
369 .Bl -tag -width Ds -offset indent -compact
370 .It Li .Fn strlen \(dqconst char *s\(dq
371 Group arguments
372 .Qq const char *s
373 into one function argument.
374 If unspecified,
375 .Qq const ,
376 .Qq char ,
377 and
378 .Qq *s
379 would be considered separate arguments.
380 .It Li .Op \(dqFl a\(dq
381 Consider
382 .Qq \&Fl a
383 as literal text instead of a flag macro.
384 .El
385 .Sh REQUEST REFERENCE
386 The
387 .Xr mandoc 1
388 .Nm
389 parser recognises the following requests.
390 Note that the
391 .Nm
```

```
language defines many more requests not implemented in
.Xr mandoc 1 .
.Ss \&ad
Set line adjustment mode.
This line-scoped request is intended to have one argument to select
normal, left, right, or centre adjustment for subsequent text.
Currently, it is ignored including its arguments,
and the number of arguments is not checked.
.Ss \&am
Append to a macro definition.
The syntax of this request is the same as that of
.Sx \&de .
It is currently ignored by
.Xr mandoc 1 ,
as are its children.
.Ss \&ami
Append to a macro definition, specifying the macro name indirectly.
The syntax of this request is the same as that of
.Sx \&dei .
It is currently ignored by
.Xr mandoc 1 ,
as are its children.
.Ss \&am1
Append to a macro definition, switching roff compatibility mode off
during macro execution.
The syntax of this request is the same as that of
.Sx \&de1 .
It is currently ignored by
.Xr mandoc 1 ,
as are its children.
.Ss \&de
Define a
.Nm
macro.
Its syntax can be either
.Bd -literal -offset indent
.Pf . Cm \&de Ar name
.Ar macro definition
\&..
.Ed
.Pp
or
.Bd -literal -offset indent
.Pf . Cm \&de Ar name Ar end
.Ar macro definition
.Pf . Ar end
.Ed
.Pp
Both forms define or redefine the macro
.Ar name
to represent the
.Ar macro definition ,
which may consist of one or more input lines, including the newline
characters terminating each line, optionally containing calls to
.Nm
requests,
.Nm
macros or high-level macros like
.Xr man 5
or
.Xr mdoc 5
macros, whichever applies to the document in question.
.Pp
Specifying a custom
.Ar end
macro works in the same way as for
```



```
.Sx \&ig ;
namely, the call to
.Sq Pf . Ar end
first ends the
.Ar macro definition ,
and after that, it is also evaluated as a
.Nm
request or
.Nm
macro, but not as a high-level macro.
.Pp
The macro can be invoked later using the syntax
.Pp
.D1 Pf . Ar name Op Ar argument Op Ar argument ...
.Pp
Regarding argument parsing, see
.Sx MACRO SYNTAX
above.
.Pp
The line invoking the macro will be replaced
in the input stream by the
.Ar macro definition ,
replacing all occurrences of
.No \e\e$ Ns Ar N ,
where
.Ar N
is a digit, by the
.Ar N Ns th Ar argument .
For example,
.Bd -literal -offset indent
\&.de ZN
\efI\e^\e\e$1\e^\efP\e\e$2
\&..
\&.ZN XtFree .
.Ed
.Pp
produces
.Pp
.D1 \efI\e^XtFree\e^\efP.
.Pp
in the input stream, and thus in the output: \fI\^XtFree\^\fP.
.Pp
Since macros and user-defined strings share a common string table,
defining a macro
.Ar name
clobbers the user-defined string
.Ar name ,
and the
.Ar macro definition
can also be printed using the
.Sq \e*
string interpolation syntax described below
.Sx ds ,
but this is rarely useful because every macro definition contains at least
one explicit newline character.
.Pp
In order to prevent endless recursion, both groff and
.Xr mandoc 1
limit the stack depth for expanding macros and strings
to a large, but finite number.
Do not rely on the exact value of this limit.
.Ss \&dei
Define a
.Nm
macro, specifying the macro name indirectly.
The syntax of this request is the same as that of
```

```
.Sx \&de .
It is currently ignored by
.Xr mandoc 1 ,
as are its children.
.Ss \&de1
Define a
.Nm
macro that will be executed with
.Nm
compatibility mode switched off during macro execution.
This is a GNU extension not available in traditional
.Nm
implementations and not even in older versions of groff.
Since
.Xr mandoc 1
does not implement
.Nm
compatibility mode at all, it handles this request as an alias for
.Sx \&de .
.Ss \&ds
Define a user-defined string.
Its syntax is as follows:
.Pp
.D1 Pf . Cm \&ds Ar name Oo \(dq Oc Ns Ar string
.Pp
The
.Ar name
and
.Ar string
arguments are space-separated.
If the
.Ar string
begins with a double-quote character, that character will not be part
of the string.
All remaining characters on the input line form the
.Ar string ,
including whitespace and double-quote characters, even trailing ones.
.Pp
The
.Ar string
can be interpolated into subsequent text by using
.No \e* Ns Bq Ar name
for a
.Ar name
of arbitrary length, or \e*(NN or \e*N if the length of
.Ar name
is two or one characters, respectively.
Interpolation can be prevented by escaping the leading backslash;
that is, an asterisk preceded by an even number of backslashes
does not trigger string interpolation.
.Pp
Since user-defined strings and macros share a common string table,
defining a string
.Ar name
clobbers the macro
.Ar name ,
and the
.Ar name
used for defining a string can also be invoked as a macro,
in which case the following input line will be appended to the
.Ar string ,
forming a new input line passed to the
.Nm
parser.
For example,
.Bd -literal -offset indent
```



```
\&.ds badidea .S
\&.badidea
H SYNOPSIS
.Ed
.Pp
invokes the
.Cm SH
macro when used in a
.Xr man 5
document.
Such abuse is of course strongly discouraged.
.Ss \&el
The
.Qq else
half of an if/else conditional.
Pops a result off the stack of conditional evaluations pushed by
.Sx \&ie
and uses it as its conditional.
If no stack entries are present (e.g., due to no prior
.Sx \&ie
calls)
then false is assumed.
The syntax of this request is similar to
.Sx \&if
except that the conditional is missing.
.Ss \&EN
End an equation block.
See
.Sx \&EQ .
.Ss \&EQ
Begin an equation block.
See
.Xr eqn 5
for a description of the equation language.
.Ss \&hy
Set automatic hyphenation mode.
This line-scoped request is currently ignored.
.Ss \&ie
The
.Qq if
half of an if/else conditional.
The result of the conditional is pushed into a stack used by subsequent
invocations of
.Sx \&el ,
which may be separated by any intervening input (or not exist at all).
Its syntax is equivalent to
.Sx \&if .
.Ss \&if
Begins a conditional.
Right now, the conditional evaluates to true
if and only if it starts with the letter
.Sy n ,
indicating processing in nroff style as opposed to troff style.
If a conditional is false, its children are not processed, but are
syntactically interpreted to preserve the integrity of the input
document.
Thus,
.Pp
.D1 \&.if t .ig
.Pp
will discard the
.Sq \&.ig ,
which may lead to interesting results, but
.Pp
.D1 \&.if t .if t \e{\e
.Pp
```

```
will continue to syntactically interpret to the block close of the final
conditional.
Sub-conditionals, in this case, obviously inherit the truth value of
the parent.
This request has the following syntax:
.Bd -literal -offset indent
\&.if COND \e{\e
BODY...
\&.\e}
.Ed
.Bd -literal -offset indent
\&.if COND \e{ BODY
BODY... \e}
.Ed
.Bd -literal -offset indent
\&.if COND \e{ BODY
BODY...
\&.\e}
.Ed
.Bd -literal -offset indent
\&.if COND \e
BODY
.Ed
.Pp
COND is a conditional statement.
roff allows for complicated conditionals; mandoc is much simpler.
At this time, mandoc supports only
.Sq n ,
evaluating to true;
and
.Sq t ,
.Sq e ,
and
.Sq o ,
evaluating to false.
All other invocations are read up to the next end of line or space and
evaluate as false.
.Pp
If the BODY section is begun by an escaped brace
.Sq \e{ ,
scope continues until a closing-brace escape sequence
.Sq \.\e} .
If the BODY is not enclosed in braces, scope continues until
the end of the line.
If the COND is followed by a BODY on the same line, whether after a
brace or not, then requests and macros
.Em must
begin with a control character.
It is generally more intuitive, in this case, to write
.Bd -literal -offset indent
\&.if COND \e{\e
\&.foo
bar
\&.\e}
.Ed
.Pp
than having the request or macro follow as
.Pp
.D1 \&.if COND \e{ .foo
.Pp
The scope of a conditional is always parsed, but only executed if the
conditional evaluates to true.
.Pp
Note that the
.Sq \e}
is converted into a zero-width escape sequence if not passed as a
```



```
standalone macro
.Sq \&.\e} .
For example,
.Pp
.D1 \&.Fl a \e} b
.Pp
will result in
.Sq \e}
being considered an argument of the
.Sq \&Fl
macro.
.Ss \&ig
Ignore input.
Its syntax can be either
.Bd -literal -offset indent
.Pf . Cm \&ig
.Ar ignored text
\&..
.Ed
.Pp
or
.Bd -literal -offset indent
.Pf . Cm \&ig Ar end
.Ar ignored text
.Pf . Ar end
.Ed
.Pp
In the first case, input is ignored until a
.Sq \&..
request is encountered on its own line.
In the second case, input is ignored until the specified
.Sq Pf . Ar end
macro is encountered.
Do not use the escape character
.Sq \e
anywhere in the definition of
.Ar end ;
it would cause very strange behaviour.
.Pp
When the
.Ar end
macro is a roff request or a roff macro, like in
.Pp
.D1 \&.ig if
.Pp
the subsequent invocation of
.Sx \&if
will first terminate the
.Ar ignored text ,
then be invoked as usual.
Otherwise, it only terminates the
.Ar ignored text ,
and arguments following it or the
.Sq \&..
request are discarded.
.Ss \&ne
Declare the need for the specified minimum vertical space
before the next trap or the bottom of the page.
This line-scoped request is currently ignored.
.Ss \&nh
Turn off automatic hyphenation mode.
This line-scoped request is currently ignored.
.Ss \&rm
Remove a request, macro or string.
This request is intended to have one argument,
the name of the request, macro or string to be undefined.
```

```
Currently, it is ignored including its arguments,
and the number of arguments is not checked.
.Ss \&nr
Define a register.
A register is an arbitrary string value that defines some sort of state,
which influences parsing and/or formatting.
Its syntax is as follows:
.Pp
.D1 Pf \. Cm \&nr Ar name Ar value
.Pp
The
.Ar value
may, at the moment, only be an integer.
So far, only the following register
.Ar name
is recognised:
.Bl -tag -width Ds
.It Cm nS
If set to a positive integer value, certain
.Xr mdoc 5
macros will behave in the same way as in the
.Em SYNOPSIS
section.
If set to 0, these macros will behave in the same way as outside the
.Em SYNOPSIS
section, even when called within the
.Em SYNOPSIS
section itself.
Note that starting a new
.Xr mdoc 5
section with the
.Cm \&Sh
macro will reset this register.
.El
.Ss \&ns
Turn on no-space mode.
This line-scoped request is intended to take no arguments.
Currently, it is ignored including its arguments,
and the number of arguments is not checked.
.Ss \&ps
Change point size.
This line-scoped request is intended to take one numerical argument.
Currently, it is ignored including its arguments,
and the number of arguments is not checked.
.Ss \&so
Include a source file.
Its syntax is as follows:
.Pp
.D1 Pf \. Cm \&so Ar file
.Pp
The
.Ar file
will be read and its contents processed as input in place of the
.Sq \&.so
request line.
To avoid inadvertent inclusion of unrelated files,
.Xr mandoc 1
only accepts relative paths not containing the strings
.Qq ../
and
.Qq /.. .
.Pp
This request requires
.Xr man 1
to change to the right directory before calling
.Xr mandoc 1 ,
```



```
per convention to the root of the manual tree.
Typical usage looks like:
.Pp
.Dl \&.so man3/Xcursor.3
.Pp
As the whole concept is rather fragile, the use of
.Sx \&so
is discouraged.
Use
.Xr ln 1
instead.
.Ss \&ta
Set tab stops.
This line-scoped request can take an arbitrary number of arguments.
Currently, it is ignored including its arguments.
.Ss \&tr
Output character translation.
Its syntax is as follows:
.Pp
.D1 Pf \. Cm \&tr Ar [ab]+
.Pp
Pairs of
.Ar ab
characters are replaced
.Ar ( a
for
.Ar b ) .
Replacement (or origin) characters may also be character escapes; thus,
.Pp
.Dl tr \e(xx\e(yy
.Pp
replaces all invocations of \e(xx with \e(yy.
.Ss \&T&
Re-start a table layout, retaining the options of the prior table
invocation.
See
.Sx \&TS .
.Ss \&TE
End a table context.
See
.Sx \&TS .
.Ss \&TS
Begin a table, which formats input in aligned rows and columns.
See
.Xr tbl 5
for a description of the tbl language.
.Sh COMPATIBILITY
This section documents compatibility between mandoc and other other
.Nm
implementations, at this time limited to GNU troff
.Pq Qq groff .
The term
.Qq historic groff
refers to groff version 1.15.
.Pp
.Bl -dash -compact
.It
In mandoc, the
.Sx \&EQ ,
.Sx \&TE ,
.Sx \&TS ,
and
.Sx \&T& ,
macros are considered regular macros.
In all other
.Nm
```

```
implementations, these are special macros that must be specified without
spacing between the control character (which must be a period) and the
macro name.
.It
The
.Cm nS
register is only compatible with OpenBSD's groff-1.15.
.It
Historic groff did not accept white-space before a custom
.Ar end
macro for the
.Sx \&ig
request.
.It
The
.Sx \&if
and family would print funny white-spaces with historic groff when
using the next-line syntax.
.El
.Sh SEE ALSO
.Xr mandoc 1 ,
.Xr eqn 5 ,
.Xr man 5 ,
.Xr mandoc_char 5 ,
.Xr mdoc 5 ,
.Xr tbl 5
.Rs
.%A Joseph F. Ossanna
.%A Brian W. Kernighan
.%I AT&T Bell Laboratories
.%T Troff User's Manual
.%R Computing Science Technical Report
.%N 54
.%C Murray Hill, New Jersey
.%D 1976 and 1992
.%U http://www.kohala.com/start/troff/cstr54.ps
.Re
.Rs
.%A Joseph F. Ossanna
.%A Brian W. Kernighan
.%A Gunnar Ritter
.%T Heirloom Documentation Tools Nroff/Troff User's Manual
.%D September 17, 2007
.%U http://heirloom.sourceforge.net/doctools/troff.pdf
.Re
.Sh HISTORY
The RUNOFF typesetting system, whose input forms the basis for
.Nm ,
was written in MAD and FAP for the CTSS operating system by Jerome E.
Saltzer in 1964.
Doug McIlroy rewrote it in BCPL in 1969, renaming it
.Nm .
Dennis M. Ritchie rewrote McIlroy's
.Nm
in PDP-11 assembly for
.At v1 ,
Joseph F. Ossanna improved roff and renamed it nroff
for
.At v2 ,
then ported nroff to C as troff, which Brian W. Kernighan released with
.At v7 .
In 1989, James Clarke re-implemented troff in C++, naming it groff.
.Sh AUTHORS
.An -nosplit
This
.Nm
```



```
reference was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv ;
and
.An Ingo Schwarze ,
.Mt schwarze@openbsd.org .
```

```
**********************************************************
   70698 Tue Jul 15 13:48:10 2014
new/usr/src/man/man5/mdoc.5
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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 (c) 2010, 2011 Ingo Schwarze <schwarze@openbsd.org>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\"
.Dd Jan 3, 2012
.Dt MDOC 5
.Os
.Sh NAME
.Nm mdoc
.Nd semantic markup language for formatting manual pages
.Sh DESCRIPTION
The
.Nm mdoc
language supports authoring of manual pages for the
.Xr man 1
utility by allowing semantic annotations of words, phrases,
page sections and complete manual pages.
Such annotations are used by formatting tools to achieve a uniform
presentation across all manuals written in
.Nm ,
and to support hyperlinking if supported by the output medium.
.Pp
This reference document describes the structure of manual pages
and the syntax and usage of the
.Nm
language.
The reference implementation of a parsing and formatting tool is
.Xr mandoc 1 ;
the
.Sx COMPATIBILITY
section describes compatibility with other implementations.
.Pp
In an
.Nm
document, lines beginning with the control character
.Sq \&.
are called
.Dq macro lines .
The first word is the macro name.
It consists of two or three letters.
Most macro names begin with a capital letter.
For a list of available macros, see
.Sx MACRO OVERVIEW .
The words following the macro name are arguments to the macro, optionally
including the names of other, callable macros; see
.Sx MACRO SYNTAX
for details.
```



```
.Pp
Lines not beginning with the control character are called
.Dq text lines .
They provide free-form text to be printed; the formatting of the text
depends on the respective processing context:
.Bd -literal -offset indent
\&.Sh Macro lines change control state.
Text lines are interpreted within the current state.
.Ed
.Pp
Many aspects of the basic syntax of the
.Nm
language are based on the
.Xr roff 5
language; see the
.Em LANGUAGE SYNTAX
and
.Em MACRO SYNTAX
sections in the
.Xr roff 5
manual for details, in particular regarding
comments, escape sequences, whitespace, and quoting.
However, using
.Xr roff 5
requests in
.Nm
documents is discouraged;
.Xr mandoc 1
supports some of them merely for backward compatibility.
.Sh MANUAL STRUCTURE
A well-formed
.Nm
document consists of a document prologue followed by one or more
sections.
.Pp
The prologue, which consists of the
.Sx \&Dd ,
.Sx \&Dt ,
and
.Sx \&Os
macros in that order, is required for every document.
.Pp
The first section (sections are denoted by
.Sx \&Sh )
must be the NAME section, consisting of at least one
.Sx \&Nm
followed by
.Sx \&Nd .
.Pp
Following that, convention dictates specifying at least the
.Em SYNOPSIS
and
.Em DESCRIPTION
sections, although this varies between manual sections.
.Pp
The following is a well-formed skeleton
.Nm
file for a utility
.Qq progname :
.Bd -literal -offset indent
\&.Dd Jan 1, 1970
\&.Dt PROGNAME section
\&.Os
\&.Sh NAME
\&.Nm progname
\&.Nd one line description
```

```
\&.\e\(dq .Sh LIBRARY
\&.\e\(dq For sections 2, 3, & 9 only.
\&.Sh SYNOPSIS
\&.Nm progname
\&.Op Fl options
\&.Ar
\&.Sh DESCRIPTION
The
\&.Nm
utility processes files ...
\&.\e\(dq .Sh IMPLEMENTATION NOTES
\&.\e\(dq .Sh RETURN VALUES
\&.\e\(dq For sections 2, 3, & 9 only.
\&.\e\(dq .Sh ENVIRONMENT
\&.\e\(dq For sections 1, 1M, 5, & 6 only.
\&.\e\(dq .Sh FILES
\&.\e\(dq .Sh EXIT STATUS
\&.\e\(dq For sections 1, 1M, & 6 only.
\&.\e\(dq .Sh EXAMPLES
\&.\e\(dq .Sh DIAGNOSTICS
\&.\e\(dq For sections 1, 1M, 5, 6, & 7 only.
\&.\e\(dq .Sh ERRORS
\&.\e\(dq For sections 2, 3, & 9 only.
\&.\e\(dq .Sh SEE ALSO
\&.\e\(dq .Xr foobar 1
\&.\e\(dq .Sh STANDARDS
\&.\e\(dq .Sh HISTORY
\&.\e\(dq .Sh AUTHORS
\&.\e\(dq .Sh CAVEATS
\&.\e\(dq .Sh BUGS
\&.\e\(dq .Sh SECURITY CONSIDERATIONS
\&.\e\(dq Not used in OpenBSD.
.Ed
.Pp
The sections in an
.Nm
document are conventionally ordered as they appear above.
Sections should be composed as follows:
.Bl -ohang -offset Ds
.It Em NAME
The name(s) and a one line description of the documented material.
The syntax for this as follows:
.Bd -literal -offset indent
\&.Nm name0 ,
\&.Nm name1 ,
\&.Nm name2
\&.Nd a one line description
.Ed
.Pp
Multiple
.Sq \&Nm
names should be separated by commas.
.Pp
The
.Sx \&Nm
macro(s) must precede the
.Sx \&Nd
macro.
.Pp
See
.Sx \&Nm
and
.Sx \&Nd .
.It Em LIBRARY
The name of the library containing the documented material, which is
assumed to be a function in a section 2, 3, or 9 manual.
```



```
The syntax for this is as follows:
.Bd -literal -offset indent
\&.Lb libarm
.Ed
.Pp
See
.Sx \&Lb .
.It Em SYNOPSIS
Documents the utility invocation syntax, function call syntax, or device
configuration.
.Pp
For the first, utilities (sections 1, 1M, and 6), this is
generally structured as follows:
.Bd -literal -offset indent
\&.Nm bar
\&.Op Fl v
\&.Op Fl o Ar file
\&.Op Ar
\&.Nm foo
\&.Op Fl v
\&.Op Fl o Ar file
\&.Op Ar
.Ed
.Pp
Commands should be ordered alphabetically.
.Pp
For the second, function calls (sections 2, 3, 9):
.Bd -literal -offset indent
\&.In header.h
\&.Vt extern const char *global;
\&.Ft "char *"
\&.Fn foo "const char *src"
\&.Ft "char *"
\&.Fn bar "const char *src"
.Ed
.Pp
Ordering of
.Sx \&In ,
.Sx \&Vt ,
.Sx \&Fn ,
and
.Sx \&Fo
macros should follow C header-file conventions.
.Pp
And for the third, configurations (section 7):
.Bd -literal -offset indent
\&.Cd \(dqit* at isa? port 0x2e\(dq
\&.Cd \(dqit* at isa? port 0x4e\(dq
.Ed
.Pp
Manuals not in these sections generally don't need a
.Em SYNOPSIS .
.Pp
Some macros are displayed differently in the
.Em SYNOPSIS
section, particularly
.Sx \&Nm ,
.Sx \&Cd ,
.Sx \&Fd ,
.Sx \&Fn ,
.Sx \&Fo ,
.Sx \&In ,
.Sx \&Vt ,
and
.Sx \&Ft .
All of these macros are output on their own line.
```

```
If two such dissimilar macros are pairwise invoked (except for
.Sx \&Ft
before
.Sx \&Fo
or
.Sx \&Fn ) ,
they are separated by a vertical space, unless in the case of
.Sx \&Fo ,
.Sx \&Fn ,
and
.Sx \&Ft ,
which are always separated by vertical space.
.Pp
When text and macros following an
.Sx \&Nm
macro starting an input line span multiple output lines,
all output lines but the first will be indented to align
with the text immediately following the
.Sx \&Nm
macro, up to the next
.Sx \&Nm ,
.Sx \&Sh ,
or
.Sx \&Ss
macro or the end of an enclosing block, whichever comes first.
.It Em DESCRIPTION
This begins with an expansion of the brief, one line description in
.Em NAME :
.Bd -literal -offset indent
The
\&.Nm
utility does this, that, and the other.
.Ed
.Pp
It usually follows with a breakdown of the options (if documenting a
command), such as:
.Bd -literal -offset indent
The arguments are as follows:
\&.Bl \-tag \-width Ds
\&.It Fl v
Print verbose information.
\&.El
.Ed
.Pp
Manuals not documenting a command won't include the above fragment.
.Pp
Since the
.Em DESCRIPTION
section usually contains most of the text of a manual, longer manuals
often use the
.Sx \&Ss
macro to form subsections.
In very long manuals, the
.Em DESCRIPTION
may be split into multiple sections, each started by an
.Sx \&Sh
macro followed by a non-standard section name, and each having
several subsections, like in the present
.Nm
manual.
.It Em IMPLEMENTATION NOTES
Implementation-specific notes should be kept here.
This is useful when implementing standard functions that may have side
effects or notable algorithmic implications.
.It Em RETURN VALUES
This section documents the
```



```
return values of functions in sections 2, 3, and 9.
.Pp
See
.Sx \&Rv .
.It Em ENVIRONMENT
Lists the environment variables used by the utility,
and explains the syntax and semantics of their values.
The
.Xr environ 5
manual provides examples of typical content and formatting.
.Pp
See
.Sx \&Ev .
.It Em FILES
Documents files used.
It's helpful to document both the file name and a short description of how
the file is used (created, modified, etc.).
.Pp
See
.Sx \&Pa .
.It Em EXIT STATUS
This section documents the
command exit status for section 1, 6, and 8 utilities.
Historically, this information was described in
.Em DIAGNOSTICS ,
a practise that is now discouraged.
.Pp
See
.Sx \&Ex .
.It Em EXAMPLES
Example usages.
This often contains snippets of well-formed, well-tested invocations.
Make sure that examples work properly!
.It Em DIAGNOSTICS
Documents error conditions.
This is most useful in section 4 manuals.
Historically, this section was used in place of
.Em EXIT STATUS
for manuals in sections 1, 6, and 8; however, this practise is
discouraged.
.Pp
See
.Sx \&Bl
.Fl diag .
.It Em ERRORS
Documents error handling in sections 2, 3, and 9.
.Pp
See
.Sx \&Er .
.It Em SEE ALSO
References other manuals with related topics.
This section should exist for most manuals.
Cross-references should conventionally be ordered first by section, then
alphabetically.
.Pp
References to other documentation concerning the topic of the manual page,
for example authoritative books or journal articles, may also be
provided in this section.
.Pp
See
.Sx \&Rs
and
.Sx \&Xr .
.It Em STANDARDS
References any standards implemented or used.
If not adhering to any standards, the
```

```
.Em HISTORY
section should be used instead.
.Pp
See
.Sx \&St .
.It Em HISTORY
A brief history of the subject, including where it was first implemented,
and when it was ported to or reimplemented for the operating system at hand.
.It Em AUTHORS
Credits to the person or persons who wrote the code and/or documentation.
Authors should generally be noted by both name and email address.
.Pp
See
.Sx \&An .
.It Em CAVEATS
Common misuses and misunderstandings should be explained
in this section.
.It Em BUGS
Known bugs, limitations, and work-arounds should be described
in this section.
.It Em SECURITY CONSIDERATIONS
Documents any security precautions that operators should consider.
.El
.Sh MACRO OVERVIEW
This overview is sorted such that macros of similar purpose are listed
together, to help find the best macro for any given purpose.
Deprecated macros are not included in the overview, but can be found below
in the alphabetical
.Sx MACRO REFERENCE .
.Ss Document preamble and NAME section macros
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Dd Ta document date: Ar month day , year
.It Sx \&Dt Ta document title: Ar TITLE SECTION Op Ar volume | arch
.It Sx \&Os Ta operating system version: Op Ar system Op Ar version
.It Sx \&Nm Ta document name (one argument)
.It Sx \&Nd Ta document description (one line)
.El
.Ss Sections and cross references
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Sh Ta section header (one line)
.It Sx \&Ss Ta subsection header (one line)
.It Sx \&Sx Ta internal cross reference to a section or subsection
.It Sx \&Xr Ta cross reference to another manual page: Ar name section
.It Sx \&Pp , \&Lp Ta start a text paragraph (no arguments)
.El
.Ss Displays and lists
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Bd , \&Ed Ta display block:
.Fl Ar type
.Op Fl offset Ar width
.Op Fl compact
.It Sx \&D1 Ta indented display (one line)
.It Sx \&Dl Ta indented literal display (one line)
.It Sx \&Bl , \&El Ta list block:
.Fl Ar type
.Op Fl width Ar val
.Op Fl offset Ar val
.Op Fl compact
.It Sx \&It Ta list item (syntax depends on Fl Ar type )
.It Sx \&Ta Ta table cell separator in Sx \&Bl Fl column No lists
.It Sx \&Rs , \&%* , \&Re Ta bibliographic block (references)
.El
.Ss Spacing control
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Pf Ta prefix, no following horizontal space (one argument)
.It Sx \&Ns Ta roman font, no preceding horizontal space (no arguments)
```



```
.It Sx \&Ap Ta apostrophe without surrounding whitespace (no arguments)
.It Sx \&Sm Ta switch horizontal spacing mode: Cm on | off
.It Sx \&Bk , \&Ek Ta keep block: Fl words
.It Sx \&br Ta force output line break in text mode (no arguments)
.It Sx \&sp Ta force vertical space: Op Ar height
.El
.Ss Semantic markup for command line utilities:
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Nm Ta start a SYNOPSIS block with the name of a utility
.It Sx \&Fl Ta command line options (flags) (>=0 arguments)
.It Sx \&Cm Ta command modifier (>0 arguments)
.It Sx \&Ar Ta command arguments (>=0 arguments)
.It Sx \&Op , \&Oo , \&Oc Ta optional syntax elements (enclosure)
.It Sx \&Ic Ta internal or interactive command (>0 arguments)
.It Sx \&Ev Ta environmental variable (>0 arguments)
.It Sx \&Pa Ta file system path (>=0 arguments)
.El
.Ss Semantic markup for function libraries:
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Lb Ta function library (one argument)
.It Sx \&In Ta include file (one argument)
.It Sx \&Ft Ta function type (>0 arguments)
.It Sx \&Fo , \&Fc Ta function block: Ar funcname
.It Sx \&Fn Ta function name:
.Op Ar functype
.Ar funcname
.Oo
.Op Ar argtype
.Ar argname
.Oc
.It Sx \&Fa Ta function argument (>0 arguments)
.It Sx \&Vt Ta variable type (>0 arguments)
.It Sx \&Va Ta variable name (>0 arguments)
.It Sx \&Dv Ta defined variable or preprocessor constant (>0 arguments)
.It Sx \&Er Ta error constant (>0 arguments)
.It Sx \&Ev Ta environmental variable (>0 arguments)
.El
.Ss Various semantic markup:
.Bl -column "Brq, Bro, Brc" description
.It Sx \&An Ta author name (>0 arguments)
.It Sx \&Lk Ta hyperlink: Ar uri Op Ar name
.It Sx \&Mt Ta Do mailto Dc hyperlink: Ar address
.It Sx \&Cd Ta kernel configuration declaration (>0 arguments)
.It Sx \&Ad Ta memory address (>0 arguments)
.It Sx \&Ms Ta mathematical symbol (>0 arguments)
.It Sx \&Tn Ta tradename (>0 arguments)
.El
.Ss Physical markup
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Em Ta italic font or underline (emphasis) (>0 arguments)
.It Sx \&Sy Ta boldface font (symbolic) (>0 arguments)
.It Sx \&Li Ta typewriter font (literal) (>0 arguments)
.It Sx \&No Ta return to roman font (normal) (no arguments)
.It Sx \&Bf , \&Ef Ta font block:
.Op Fl Ar type | Cm \&Em | \&Li | \&Sy
.El
.Ss Physical enclosures
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Dq , \&Do , \&Dc Ta enclose in typographic double quotes: Dq text
.It Sx \&Qq , \&Qo , \&Qc Ta enclose in typewriter double quotes: Qq text
.It Sx \&Sq , \&So , \&Sc Ta enclose in single quotes: Sq text
.It Sx \&Ql Ta single-quoted literal text: Ql text
.It Sx \&Pq , \&Po , \&Pc Ta enclose in parentheses: Pq text
.It Sx \&Bq , \&Bo , \&Bc Ta enclose in square brackets: Bq text
.It Sx \&Brq , \&Bro , \&Brc Ta enclose in curly braces: Brq text
.It Sx \&Aq , \&Ao , \&Ac Ta enclose in angle brackets: Aq text
```

```
.It Sx \&Eo , \&Ec Ta generic enclosure
.El
.Ss Text production
.Bl -column "Brq, Bro, Brc" description
.It Sx \&Ex Fl std Ta standard command exit values: Op Ar utility ...
.It Sx \&Rv Fl std Ta standard function return values: Op Ar function ...
.It Sx \&St Ta reference to a standards document (one argument)
.It Sx \&Ux Ta Ux
.It Sx \&At Ta At
.It Sx \&Bx Ta Bx
.It Sx \&Bsx Ta Bsx
.It Sx \&Nx Ta Nx
.It Sx \&Fx Ta Fx
.It Sx \&Ox Ta Ox
.It Sx \&Dx Ta Dx
.El
.Sh MACRO REFERENCE
This section is a canonical reference of all macros, arranged
alphabetically.
For the scoping of individual macros, see
.Sx MACRO SYNTAX .
.Ss \&%A
Author name of an
.Sx \&Rs
block.
Multiple authors should each be accorded their own
.Sx \%%A
line.
Author names should be ordered with full or abbreviated forename(s)
first, then full surname.
.Ss \&%B
Book title of an
.Sx \&Rs
block.
This macro may also be used in a non-bibliographic context when
referring to book titles.
.Ss \&%C
Publication city or location of an
.Sx \&Rs
block.
.Ss \&%D
Publication date of an
.Sx \&Rs
block.
Recommended formats of arguments are
.Ar month day , year
or just
.Ar year .
.Ss \&%I
Publisher or issuer name of an
.Sx \&Rs
block.
.Ss \&%J
Journal name of an
.Sx \&Rs
block.
.Ss \&%N
Issue number (usually for journals) of an
.Sx \&Rs
block.
.Ss \&%O
Optional information of an
.Sx \&Rs
block.
.Ss \&%P
Book or journal page number of an
```



```
.Sx \&Rs
block.
.Ss \&%Q
Institutional author (school, government, etc.) of an
.Sx \&Rs
block.
Multiple institutional authors should each be accorded their own
.Sx \&%Q
line.
.Ss \&%R
Technical report name of an
.Sx \&Rs
block.
.Ss \&%T
Article title of an
.Sx \&Rs
block.
This macro may also be used in a non-bibliographical context when
referring to article titles.
.Ss \&%U
URI of reference document.
.Ss \&%V
Volume number of an
.Sx \&Rs
block.
.Ss \&Ac
Close an
.Sx \&Ao
block.
Does not have any tail arguments.
.Ss \&Ad
Memory address.
Do not use this for postal addresses.
.Pp
Examples:
.Dl \&.Ad [0,$]
.Dl \&.Ad 0x00000000
.Ss \&An
Author name.
Can be used both for the authors of the program, function, or driver
documented in the manual, or for the authors of the manual itself.
Requires either the name of an author or one of the following arguments:
.Pp
.Bl -tag -width "-nosplitX" -offset indent -compact
.It Fl split
Start a new output line before each subsequent invocation of
.Sx \&An .
.It Fl nosplit
The opposite of
.Fl split .
.El
.Pp
The default is
.Fl nosplit .
The effect of selecting either of the
.Fl split
modes ends at the beginning of the
.Em AUTHORS
section.
In the
.Em AUTHORS
section, the default is
.Fl nosplit
for the first author listing and
.Fl split
for all other author listings.
```

```
.Pp
Examples:
.Dl \&.An -nosplit
.Dl \&.An Kristaps Dzonsons \&Aq kristaps@bsd.lv
.Ss \&Ao
Begin a block enclosed by angle brackets.
Does not have any head arguments.
.Pp
Examples:
.Dl \&.Fl -key= \&Ns \&Ao \&Ar val \&Ac
.Pp
See also
.Sx \&Aq .
.Ss \&Ap
Inserts an apostrophe without any surrounding whitespace.
This is generally used as a grammatical device when referring to the verb
form of a function.
.Pp
Examples:
.Dl \&.Fn execve \&Ap d
.Ss \&Aq
Encloses its arguments in angle brackets.
.Pp
Examples:
.Dl \&.Fl -key= \&Ns \&Aq \&Ar val
.Pp
.Em Remarks :
this macro is often abused for rendering URIs, which should instead use
.Sx \&Lk
or
.Sx \&Mt ,
or to note pre-processor
.Dq Li #include
statements, which should use
.Sx \&In .
.Pp
See also
.Sx \&Ao .
.Ss \&Ar
Command arguments.
If an argument is not provided, the string
.Dq file ...\&
is used as a default.
.Pp
Examples:
.Dl ".Fl o Ar file"
.Dl ".Ar"
.Dl ".Ar arg1 , arg2 ."
.Pp
The arguments to the
.Sx \&Ar
macro are names and placeholders for command arguments;
for fixed strings to be passed verbatim as arguments, use
.Sx \&Fl
or
.Sx \&Cm .
.Ss \&At
Formats an AT&T version.
Accepts one optional argument:
.Pp
.Bl -tag -width "v[1-7] | 32vX" -offset indent -compact
.It Cm v[1-7] | 32v
A version of
.At .
.It Cm III
.At III .
```



```
.It Cm V[.[1-4]]?
A version of
.At V .
.El
.Pp
Note that these arguments do not begin with a hyphen.
.Pp
Examples:
.Dl \&.At
.Dl \&.At III
.Dl \&.At V.1
.Pp
See also
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Nx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Bc
Close a
.Sx \&Bo
block.
Does not have any tail arguments.
.Ss \&Bd
Begin a display block.
Its syntax is as follows:
.Bd -ragged -offset indent
.Pf \. Sx \&Bd
.Fl Ns Ar type
.Op Fl offset Ar width
.Op Fl compact
.Ed
.Pp
Display blocks are used to select a different indentation and
justification than the one used by the surrounding text.
They may contain both macro lines and text lines.
By default, a display block is preceded by a vertical space.
.Pp
The
.Ar type
must be one of the following:
.Bl -tag -width 13n -offset indent
.It Fl centered
Produce one output line from each input line, and centre-justify each line.
Using this display type is not recommended; many
.Nm
implementations render it poorly.
.It Fl filled
Change the positions of line breaks to fill each line, and left- and
right-justify the resulting block.
.It Fl literal
Produce one output line from each input line,
and do not justify the block at all.
Preserve white space as it appears in the input.
Always use a constant-width font.
Use this for displaying source code.
.It Fl ragged
Change the positions of line breaks to fill each line, and left-justify
the resulting block.
.It Fl unfilled
The same as
.Fl literal ,
but using the same font as for normal text, which is a variable width font
```

```
788 if supported by the output device.
789 .El
790 .Pp
791 The
792 .Ar type
793 must be provided first.
794 Additional arguments may follow:
795 .Bl -tag -width 13n -offset indent
796 .It Fl offset Ar width
797 Indent the display by the
798 .Ar width ,
799 which may be one of the following:
800 .Bl -item
801 .It
802 One of the pre-defined strings
803 .Cm indent ,
804 the width of a standard indentation (six constant width characters);
805 .Cm indent-two ,
806 twice
807 .Cm indent ;
808 .Cm left ,
809 which has no effect;
810 .Cm right ,
811 which justifies to the right margin; or
812 .Cm center ,
813 which aligns around an imagined centre axis.
814 .It
815 A macro invocation, which selects a predefined width
816 associated with that macro.
817 The most popular is the imaginary macro
818 .Ar \&Ds ,
819 which resolves to
820 .Sy 6n .
821 .It
822 A width using the syntax described in
823 .Sx Scaling Widths .
824 .It
825 An arbitrary string, which indents by the length of this string.
826 .El
827 .Pp
828 When the argument is missing,
829 .Fl offset
830 is ignored.
831 .It Fl compact
832 Do not assert vertical space before the display.
833 .El
834 .Pp
835 Examples:
836 .Bd -literal -offset indent
837 \&.Bd \-literal \-offset indent \-compact
838    Hello       world.
839 \&.Ed
840 .Ed
841 .Pp
842 See also
843 .Sx \&D1
844 and
845 .Sx \&Dl .
846 .Ss \&Bf
847 Change the font mode for a scoped block of text.
848 Its syntax is as follows:
849 .Bd -ragged -offset indent
850 .Pf \. Sx \&Bf
851 .Oo
852 .Fl emphasis | literal | symbolic |
853 .Cm \&Em | \&Li | \&Sy
```



```
854 .Oc
855 .Ed
856 .Pp
857 The
858 .Fl emphasis
859 and
860 .Cm \&Em
861 argument are equivalent, as are
862 .Fl symbolic
863 and
864 .Cm \&Sy ,
865 and
866 .Fl literal
867 and
868 .Cm \&Li .
869 Without an argument, this macro does nothing.
870 The font mode continues until broken by a new font mode in a nested
871 scope or
872 .Sx \&Ef
873 is encountered.
874 .Pp
875 See also
876 .Sx \&Li ,
877 .Sx \&Ef ,
878 .Sx \&Em ,
879 and
880 .Sx \&Sy .
881 .Ss \&Bk
882 For each macro, keep its output together on the same output line,
883 until the end of the macro or the end of the input line is reached,
884 whichever comes first.
885 Line breaks in text lines are unaffected.
886 The syntax is as follows:
887 .Pp
888 .D1 Pf \. Sx \&Bk Fl words
889 .Pp
890 The
891 .Fl words
892 argument is required; additional arguments are ignored.
893 .Pp
894 The following example will not break within each
895 .Sx \&Op
896 macro line:
897 .Bd -literal -offset indent
898 \&.Bk \-words
899 \&.Op Fl f Ar flags
900 \&.Op Fl o Ar output
901 \&.Ek
902 .Ed
903 .Pp
904 Be careful in using over-long lines within a keep block!
905 Doing so will clobber the right margin.
906 .Ss \&Bl
907 Begin a list.
908 Lists consist of items specified using the
909 .Sx \&It
910 macro, containing a head or a body or both.
911 The list syntax is as follows:
912 .Bd -ragged -offset indent
913 .Pf \. Sx \&Bl
914 .Fl Ns Ar type
915 .Op Fl width Ar val
916 .Op Fl offset Ar val
917 .Op Fl compact
918 .Op HEAD ...
919 .Ed
```

```
.Pp
The list
.Ar type
is mandatory and must be specified first.
The
.Fl width
and
.Fl offset
arguments accept
.Sx Scaling Widths
or use the length of the given string.
The
.Fl offset
is a global indentation for the whole list, affecting both item heads
and bodies.
For those list types supporting it, the
.Fl width
argument requests an additional indentation of item bodies,
to be added to the
.Fl offset .
Unless the
.Fl compact
argument is specified, list entries are separated by vertical space.
.Pp
A list must specify one of the following list types:
.Bl -tag -width 12n -offset indent
.It Fl bullet
No item heads can be specified, but a bullet will be printed at the head
of each item.
Item bodies start on the same output line as the bullet
and are indented according to the
.Fl width
argument.
.It Fl column
A columnated list.
The
.Fl width
argument has no effect; instead, each argument specifies the width
of one column, using either the
.Sx Scaling Widths
syntax or the string length of the argument.
If the first line of the body of a
.Fl column
list is not an
.Sx \&It
macro line,
.Sx \&It
contexts spanning one input line each are implied until an
.Sx \&It
macro line is encountered, at which point items start being interpreted as
described in the
.Sx \&It
documentation.
.It Fl dash
Like
.Fl bullet ,
except that dashes are used in place of bullets.
.It Fl diag
Like
.Fl inset ,
except that item heads are not parsed for macro invocations.
Most often used in the
.Em DIAGNOSTICS
section with error constants in the item heads.
.It Fl enum
A numbered list.
```



```
No item heads can be specified.
Formatted like
.Fl bullet ,
except that cardinal numbers are used in place of bullets,
starting at 1.
.It Fl hang
Like
.Fl tag ,
except that the first lines of item bodies are not indented, but follow
the item heads like in
.Fl inset
lists.
.It Fl hyphen
Synonym for
.Fl dash .
.It Fl inset
Item bodies follow items heads on the same line, using normal inter-word
spacing.
Bodies are not indented, and the
.Fl width
argument is ignored.
.It Fl item
No item heads can be specified, and none are printed.
Bodies are not indented, and the
.Fl width
argument is ignored.
.It Fl ohang
Item bodies start on the line following item heads and are not indented.
The
.Fl width
argument is ignored.
.It Fl tag
Item bodies are indented according to the
.Fl width
argument.
When an item head fits inside the indentation, the item body follows
this head on the same output line.
Otherwise, the body starts on the output line following the head.
.El
.Pp
Lists may be nested within lists and displays.
Nesting of
.Fl column
and
.Fl enum
lists may not be portable.
.Pp
See also
.Sx \&El
and
.Sx \&It .
.Ss \&Bo
Begin a block enclosed by square brackets.
Does not have any head arguments.
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.Bo 1 ,
\&.Dv BUFSIZ \&Bc
.Ed
.Pp
See also
.Sx \&Bq .
.Ss \&Bq
Encloses its arguments in square brackets.
.Pp
```

```
Examples:
.Dl \&.Bq 1 , \&Dv BUFSIZ
.Pp
.Em Remarks :
this macro is sometimes abused to emulate optional arguments for
commands; the correct macros to use for this purpose are
.Sx \&Op ,
.Sx \&Oo ,
and
.Sx \&Oc .
.Pp
See also
.Sx \&Bo .
.Ss \&Brc
Close a
.Sx \&Bro
block.
Does not have any tail arguments.
.Ss \&Bro
Begin a block enclosed by curly braces.
Does not have any head arguments.
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.Bro 1 , ... ,
\&.Va n \&Brc
.Ed
.Pp
See also
.Sx \&Brq .
.Ss \&Brq
Encloses its arguments in curly braces.
.Pp
Examples:
.Dl \&.Brq 1 , ... , \&Va n
.Pp
See also
.Sx \&Bro .
.Ss \&Bsx
Format the BSD/OS version provided as an argument, or a default value if
no argument is provided.
.Pp
Examples:
.Dl \&.Bsx 1.0
.Dl \&.Bsx
.Pp
See also
.Sx \&At ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Nx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Bt
Prints
.Dq is currently in beta test.
.Ss \&Bx
Format the BSD version provided as an argument, or a default value if no
argument is provided.
.Pp
Examples:
.Dl \&.Bx 4.3 Tahoe
.Dl \&.Bx 4.4
.Dl \&.Bx
```



```
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Nx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Cd
Kernel configuration declaration.
This denotes strings accepted by
.Xr config 8 .
It is most often used in section 4 manual pages.
.Pp
Examples:
.Dl \&.Cd device le0 at scode?
.Pp
.Em Remarks :
this macro is commonly abused by using quoted literals to retain
whitespace and align consecutive
.Sx \&Cd
declarations.
This practise is discouraged.
.Ss \&Cm
Command modifiers.
Typically used for fixed strings passed as arguments, unless
.Sx \&Fl
is more appropriate.
Also useful when specifying configuration options or keys.
.Pp
Examples:
.Dl ".Nm mt Fl f Ar device Cm rewind"
.Dl ".Nm ps Fl o Cm pid , Ns Cm command"
.Dl ".Nm dd Cm if= Ns Ar file1 Cm of= Ns Ar file2"
.Dl ".Cm IdentityFile Pa ~/.ssh/id_rsa"
.Dl ".Cm LogLevel Dv DEBUG"
.Ss \&D1
One-line indented display.
This is formatted by the default rules and is useful for simple indented
statements.
It is followed by a newline.
.Pp
Examples:
.Dl \&.D1 \&Fl abcdefgh
.Pp
See also
.Sx \&Bd
and
.Sx \&Dl .
.Ss \&Db
Switch debugging mode.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Db Cm on | off
.Pp
This macro is ignored by
.Xr mandoc 1 .
.Ss \&Dc
Close a
.Sx \&Do
block.
Does not have any tail arguments.
.Ss \&Dd
Document date.
```

```
This is the mandatory first macro of any
.Nm
manual.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Dd Ar month day , year
.Pp
The
.Ar month
is the full English month name, the
.Ar day
is an optionally zero-padded numeral, and the
.Ar year
is the full four-digit year.
.Pp
Other arguments are not portable; the
.Xr mandoc 1
utility handles them as follows:
.Bl -dash -offset 3n -compact
.It
To have the date automatically filled in by the
.Ox
version of
.Xr cvs 1 ,
the special string
.Dq $\&Mdocdate$
can be given as an argument.
.It
A few alternative date formats are accepted as well
and converted to the standard form.
.It
If a date string cannot be parsed, it is used verbatim.
.It
If no date string is given, the current date is used.
.El
.Pp
Examples:
.Dl \&.Dd $\&Mdocdate$
.Dl \&.Dd $\&Mdocdate: July 21 2007$
.Dl \&.Dd July 21, 2007
.Pp
See also
.Sx \&Dt
and
.Sx \&Os .
.Ss \&Dl
One-line intended display.
This is formatted as literal text and is useful for commands and
invocations.
It is followed by a newline.
.Pp
Examples:
.Dl \&.Dl % mandoc mdoc.7 \e(ba less
.Pp
See also
.Sx \&Bd
and
.Sx \&D1 .
.Ss \&Do
Begin a block enclosed by double quotes.
Does not have any head arguments.
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.Do
April is the cruellest month
```



```
\&.Dc
\e(em T.S. Eliot
.Ed
.Pp
See also
.Sx \&Dq .
.Ss \&Dq
Encloses its arguments in
.Dq typographic
double-quotes.
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.Dq April is the cruellest month
\e(em T.S. Eliot
.Ed
.Pp
See also
.Sx \&Qq ,
.Sx \&Sq ,
and
.Sx \&Do .
.Ss \&Dt
Document title.
This is the mandatory second macro of any
.Nm
file.
Its syntax is as follows:
.Bd -ragged -offset indent
.Pf \. Sx \&Dt
.Oo
.Ar title
.Oo
.Ar section
.Op Ar volume
.Op Ar arch
.Oc
.Oc
.Ed
.Pp
Its arguments are as follows:
.Bl -tag -width Ds -offset Ds
.It Ar title
The document’s title (name), defaulting to
.Dq UNKNOWN
if unspecified.
It should be capitalised.
.It Ar section
The manual section. It should correspond to the manual’s filename suffix
and defaults to
.Dq 1
if unspecified.
.It Ar volume
This overrides the volume inferred from
.Ar section .
This field is optional.
.It Ar arch
This specifies the machine architecture a manual page applies to,
where relevant.
.El
.Ss \&Dv
Defined variables such as preprocessor constants, constant symbols,
enumeration values, and so on.
.Pp
Examples:
.Dl \&.Dv NULL
```

```
.Dl \&.Dv BUFSIZ
.Dl \&.Dv STDOUT_FILENO
.Pp
See also
.Sx \&Er
and
.Sx \&Ev
for special-purpose constants and
.Sx \&Va
for variable symbols.
.Ss \&Dx
Format the DragonFly BSD version provided as an argument, or a default
value if no argument is provided.
.Pp
Examples:
.Dl \&.Dx 2.4.1
.Dl \&.Dx
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Fx ,
.Sx \&Nx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Ec
Close a scope started by
.Sx \&Eo .
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Ec Op Ar TERM
.Pp
The
.Ar TERM
argument is used as the enclosure tail, for example, specifying \e(rq
will emulate
.Sx \&Dc .
.Ss \&Ed
End a display context started by
.Sx \&Bd .
.Ss \&Ef
End a font mode context started by
.Sx \&Bf .
.Ss \&Ek
End a keep context started by
.Sx \&Bk .
.Ss \&El
End a list context started by
.Sx \&Bl .
.Pp
See also
.Sx \&Bl
and
.Sx \&It .
.Ss \&Em
Denotes text that should be
.Em emphasised .
Note that this is a presentation term and should not be used for
stylistically decorating technical terms.
Depending on the output device, this is usually represented
using an italic font or underlined characters.
.Pp
Examples:
.Dl \&.Em Warnings!
```



```
.Dl \&.Em Remarks :
.Pp
See also
.Sx \&Bf ,
.Sx \&Li ,
.Sx \&No ,
and
.Sx \&Sy .
.Ss \&En
This macro is obsolete and not implemented in
.Xr mandoc 1 .
.Ss \&Eo
An arbitrary enclosure.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Eo Op Ar TERM
.Pp
The
.Ar TERM
argument is used as the enclosure head, for example, specifying \e(lq
will emulate
.Sx \&Do .
.Ss \&Er
Error constants for definitions of the
.Va errno
libc global variable.
This is most often used in section 2 and 3 manual pages.
.Pp
Examples:
.Dl \&.Er EPERM
.Dl \&.Er ENOENT
.Pp
See also
.Sx \&Dv
for general constants.
.Ss \&Es
This macro is obsolete and not implemented.
.Ss \&Ev
Environmental variables such as those specified in
.Xr environ 7 .
.Pp
Examples:
.Dl \&.Ev DISPLAY
.Dl \&.Ev PATH
.Pp
See also
.Sx \&Dv
for general constants.
.Ss \&Ex
Insert a standard sentence regarding command exit values of 0 on success
and >0 on failure.
This is most often used in section 1, 6, and 8 manual pages.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Ex Fl std Op Ar utility ...
.Pp
If
.Ar utility
is not specified, the document’s name set by
.Sx \&Nm
is used.
Multiple
.Ar utility
arguments are treated as separate utilities.
.Pp
See also
```

```
.Sx \&Rv .
.Ss \&Fa
Function argument.
Its syntax is as follows:
.Bd -ragged -offset indent
.Pf \. Sx \&Fa
.Op Cm argtype
.Cm argname
.Ed
.Pp
This may be invoked for names with or without the corresponding type.
It is also used to specify the field name of a structure.
Most often, the
.Sx \&Fa
macro is used in the
.Em SYNOPSIS
within
.Sx \&Fo
section when documenting multi-line function prototypes.
If invoked with multiple arguments, the arguments are separated by a
comma.
Furthermore, if the following macro is another
.Sx \&Fa ,
the last argument will also have a trailing comma.
.Pp
Examples:
.Dl \&.Fa \(dqconst char *p\(dq
.Dl \&.Fa \(dqint a\(dq \(dqint b\(dq \(dqint c\(dq
.Dl \&.Fa foo
.Pp
See also
.Sx \&Fo .
.Ss \&Fc
End a function context started by
.Sx \&Fo .
.Ss \&Fd
Historically used to document include files.
This usage has been deprecated in favour of
.Sx \&In .
Do not use this macro.
.Pp
See also
.Sx MANUAL STRUCTURE
and
.Sx \&In .
.Ss \&Fl
Command-line flag or option.
Used when listing arguments to command-line utilities.
Prints a fixed-width hyphen
.Sq \-
directly followed by each argument.
If no arguments are provided, a hyphen is printed followed by a space.
If the argument is a macro, a hyphen is prefixed to the subsequent macro
output.
.Pp
Examples:
.Dl ".Fl R Op Fl H | L | P"
.Dl ".Op Fl 1AaCcdFfgHhikLlmnopqRrSsTtux"
.Dl ".Fl type Cm d Fl name Pa CVS"
.Dl ".Fl Ar signal_number"
.Dl ".Fl o Fl"
.Pp
See also
.Sx \&Cm .
.Ss \&Fn
A function name.
```



```
Its syntax is as follows:
.Bd -ragged -offset indent
.Pf \. Ns Sx \&Fn
.Op Ar functype
.Ar funcname
.Op Oo Ar argtype Oc Ar argname
.Ed
.Pp
Function arguments are surrounded in parenthesis and
are delimited by commas.
If no arguments are specified, blank parenthesis are output.
In the
.Em SYNOPSIS
section, this macro starts a new output line,
and a blank line is automatically inserted between function definitions.
.Pp
Examples:
.Dl \&.Fn \(dqint funcname\(dq \(dqint arg0\(dq \(dqint arg1\(dq
.Dl \&.Fn funcname \(dqint arg0\(dq
.Dl \&.Fn funcname arg0
.Pp
.Bd -literal -offset indent -compact
\&.Ft functype
\&.Fn funcname
.Ed
.Pp
When referring to a function documented in another manual page, use
.Sx \&Xr
instead.
See also
.Sx MANUAL STRUCTURE ,
.Sx \&Fo ,
and
.Sx \&Ft .
.Ss \&Fo
Begin a function block.
This is a multi-line version of
.Sx \&Fn .
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Fo Ar funcname
.Pp
Invocations usually occur in the following context:
.Bd -ragged -offset indent
.Pf \. Sx \&Ft Ar functype
.br
.Pf \. Sx \&Fo Ar funcname
.br
.Pf \. Sx \&Fa Oo Ar argtype Oc Ar argname
.br
\&.\.\.
.br
.Pf \. Sx \&Fc
.Ed
.Pp
A
.Sx \&Fo
scope is closed by
.Sx \&Fc .
.Pp
See also
.Sx MANUAL STRUCTURE ,
.Sx \&Fa ,
.Sx \&Fc ,
and
.Sx \&Ft .
```

```
.Ss \&Fr
This macro is obsolete and not implemented in
.Xr mandoc 1 .
.Pp
It was used to show function return values.
The syntax was:
.Pp
.Dl Pf . Sx \&Fr Ar value
.Ss \&Ft
A function type.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Ft Ar functype
.Pp
In the
.Em SYNOPSIS
section, a new output line is started after this macro.
.Pp
Examples:
.Dl \&.Ft int
.Bd -literal -offset indent -compact
\&.Ft functype
\&.Fn funcname
.Ed
.Pp
See also
.Sx MANUAL STRUCTURE ,
.Sx \&Fn ,
and
.Sx \&Fo .
.Ss \&Fx
Format the
.Fx
version provided as an argument, or a default value
if no argument is provided.
.Pp
Examples:
.Dl \&.Fx 7.1
.Dl \&.Fx
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Nx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Hf
This macro is not implemented in
.Xr mandoc 1 .
.Pp
It was used to include the contents of a (header) file literally.
The syntax was:
.Pp
.Dl Pf . Sx \&Hf Ar filename
.Ss \&Ic
Designate an internal or interactive command.
This is similar to
.Sx \&Cm
but used for instructions rather than values.
.Pp
Examples:
.Dl \&.Ic :wq
.Dl \&.Ic hash
```



```
.Dl \&.Ic alias
.Pp
Note that using
.Sx \&Bd Fl literal
or
.Sx \&D1
is preferred for displaying code; the
.Sx \&Ic
macro is used when referring to specific instructions.
.Ss \&In
An
.Dq include
file.
When invoked as the first macro on an input line in the
.Em SYNOPSIS
section, the argument is displayed in angle brackets
and preceded by
.Dq #include ,
and a blank line is inserted in front if there is a preceding
function declaration.
This is most often used in section 2, 3, and 9 manual pages.
.Pp
Examples:
.Dl \&.In sys/types.h
.Pp
See also
.Sx MANUAL STRUCTURE .
.Ss \&It
A list item.
The syntax of this macro depends on the list type.
.Pp
Lists
of type
.Fl hang ,
.Fl ohang ,
.Fl inset ,
and
.Fl diag
have the following syntax:
.Pp
.D1 Pf \. Sx \&It Ar args
.Pp
Lists of type
.Fl bullet ,
.Fl dash ,
.Fl enum ,
.Fl hyphen
and
.Fl item
have the following syntax:
.Pp
.D1 Pf \. Sx \&It
.Pp
with subsequent lines interpreted within the scope of the
.Sx \&It
until either a closing
.Sx \&El
or another
.Sx \&It .
.Pp
The
.Fl tag
list has the following syntax:
.Pp
.D1 Pf \. Sx \&It Op Cm args
.Pp
```

```
Subsequent lines are interpreted as with
.Fl bullet
and family.
The line arguments correspond to the list’s left-hand side; body
arguments correspond to the list’s contents.
.Pp
The
.Fl column
list is the most complicated.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&It Ar cell Op <TAB> Ar cell ...
.D1 Pf \. Sx \&It Ar cell Op Sx \&Ta Ar cell ...
.Pp
The arguments consist of one or more lines of text and macros
representing a complete table line.
Cells within the line are delimited by tabs or by the special
.Sx \&Ta
block macro.
The tab cell delimiter may only be used within the
.Sx \&It
line itself; on following lines, only the
.Sx \&Ta
macro can be used to delimit cells, and
.Sx \&Ta
is only recognised as a macro when called by other macros,
not as the first macro on a line.
.Pp
Note that quoted strings may span tab-delimited cells on an
.Sx \&It
line.
For example,
.Pp
.Dl .It \(dqcol1 ; <TAB> col2 ;\(dq \&;
.Pp
will preserve the semicolon whitespace except for the last.
.Pp
See also
.Sx \&Bl .
.Ss \&Lb
Specify a library.
The syntax is as follows:
.Pp
.D1 Pf \. Sx \&Lb Ar library
.Pp
The
.Ar library
parameter may be a system library, such as
.Cm libz
or
.Cm libpam ,
in which case a small library description is printed next to the linker
invocation; or a custom library, in which case the library name is
printed in quotes.
This is most commonly used in the
.Em SYNOPSIS
section as described in
.Sx MANUAL STRUCTURE .
.Pp
Examples:
.Dl \&.Lb libz
.Dl \&.Lb mdoc
.Ss \&Li
Denotes text that should be in a
.Li literal
font mode.
```



```
Note that this is a presentation term and should not be used for
stylistically decorating technical terms.
.Pp
On terminal output devices, this is often indistinguishable from
normal text.
.Pp
See also
.Sx \&Bf ,
.Sx \&Em ,
.Sx \&No ,
and
.Sx \&Sy .
.Ss \&Lk
Format a hyperlink.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Lk Ar uri Op Ar name
.Pp
Examples:
.Dl \&.Lk http://bsd.lv \(dqThe BSD.lv Project\(dq
.Dl \&.Lk http://bsd.lv
.Pp
See also
.Sx \&Mt .
.Ss \&Lp
Synonym for
.Sx \&Pp .
.Ss \&Ms
Display a mathematical symbol.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Ms Ar symbol
.Pp
Examples:
.Dl \&.Ms sigma
.Dl \&.Ms aleph
.Ss \&Mt
Format a
.Dq mailto:
hyperlink.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Mt Ar address
.Pp
Examples:
.Dl \&.Mt discuss@manpages.bsd.lv
.Ss \&Nd
A one line description of the manual’s content.
This may only be invoked in the
.Em SYNOPSIS
section subsequent the
.Sx \&Nm
macro.
.Pp
Examples:
.Dl Pf . Sx \&Nd mdoc language reference
.Dl Pf . Sx \&Nd format and display UNIX manuals
.Pp
The
.Sx \&Nd
macro technically accepts child macros and terminates with a subsequent
.Sx \&Sh
invocation.
Do not assume this behaviour: some
.Xr whatis 1
database generators are not smart enough to parse more than the line
```

```
arguments and will display macros verbatim.
.Pp
See also
.Sx \&Nm .
.Ss \&Nm
The name of the manual page, or \(em in particular in section 1, 6,
and 8 pages \(em of an additional command or feature documented in
the manual page.
When first invoked, the
.Sx \&Nm
macro expects a single argument, the name of the manual page.
Usually, the first invocation happens in the
.Em NAME
section of the page.
The specified name will be remembered and used whenever the macro is
called again without arguments later in the page.
The
.Sx \&Nm
macro uses
.Sx Block full-implicit
semantics when invoked as the first macro on an input line in the
.Em SYNOPSIS
section; otherwise, it uses ordinary
.Sx In-line
semantics.
.Pp
Examples:
.Bd -literal -offset indent
\&.Sh SYNOPSIS
\&.Nm cat
\&.Op Fl benstuv
\&.Op Ar
.Ed
.Pp
In the
.Em SYNOPSIS
of section 2, 3 and 9 manual pages, use the
.Sx \&Fn
macro rather than
.Sx \&Nm
to mark up the name of the manual page.
.Ss \&No
Normal text.
Closes the scope of any preceding in-line macro.
When used after physical formatting macros like
.Sx \&Em
or
.Sx \&Sy ,
switches back to the standard font face and weight.
Can also be used to embed plain text strings in macro lines
using semantic annotation macros.
.Pp
Examples:
.Dl ".Em italic , Sy bold , No and roman"
.Pp
.Bd -literal -offset indent -compact
\&.Sm off
\&.Cm :C No / Ar pattern No / Ar replacement No /
\&.Sm on
.Ed
.Pp
See also
.Sx \&Em ,
.Sx \&Li ,
and
.Sx \&Sy .
```



```
.Ss \&Ns
Suppress a space between the output of the preceding macro
and the following text or macro.
Following invocation, input is interpreted as normal text
just like after an
.Sx \&No
macro.
.Pp
This has no effect when invoked at the start of a macro line.
.Pp
Examples:
.Dl ".Ar name Ns = Ns Ar value"
.Dl ".Cm :M Ns Ar pattern"
.Dl ".Fl o Ns Ar output"
.Pp
See also
.Sx \&No
and
.Sx \&Sm .
.Ss \&Nx
Format the
.Nx
version provided as an argument, or a default value if
no argument is provided.
.Pp
Examples:
.Dl \&.Nx 5.01
.Dl \&.Nx
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Ox ,
and
.Sx \&Ux .
.Ss \&Oc
Close multi-line
.Sx \&Oo
context.
.Ss \&Oo
Multi-line version of
.Sx \&Op .
.Pp
Examples:
.Bd -literal -offset indent -compact
\&.Oo
\&.Op Fl flag Ns Ar value
\&.Oc
.Ed
.Ss \&Op
Optional part of a command line.
Prints the argument(s) in brackets.
This is most often used in the
.Em SYNOPSIS
section of section 1 and 8 manual pages.
.Pp
Examples:
.Dl \&.Op \&Fl a \&Ar b
.Dl \&.Op \&Ar a | b
.Pp
See also
.Sx \&Oo .
.Ss \&Os
```

```
Document operating system version.
This is the mandatory third macro of
any
.Nm
file.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Os Op Ar system Op Ar version
.Pp
The optional
.Ar system
parameter specifies the relevant operating system or environment.
Left unspecified, it defaults to the local operating system version.
This is the suggested form.
.Pp
Examples:
.Dl \&.Os
.Dl \&.Os KTH/CSC/TCS
.Dl \&.Os BSD 4.3
.Pp
See also
.Sx \&Dd
and
.Sx \&Dt .
.Ss \&Ot
This macro is obsolete and not implemented in
.Xr mandoc 1 .
.Pp
Historical
.Xr mdoc 5
packages described it as
.Dq "old function type (FORTRAN)" .
.Ss \&Ox
Format the
.Ox
version provided as an argument, or a default value
if no argument is provided.
.Pp
Examples:
.Dl \&.Ox 4.5
.Dl \&.Ox
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Nx ,
and
.Sx \&Ux .
.Ss \&Pa
An absolute or relative file system path, or a file or directory name.
If an argument is not provided, the character
.Sq \(ti
is used as a default.
.Pp
Examples:
.Dl \&.Pa /usr/bin/mandoc
.Dl \&.Pa /usr/share/man/man7/mdoc.7
.Pp
See also
.Sx \&Lk .
.Ss \&Pc
Close parenthesised context opened by
.Sx \&Po .
```



```
.Ss \&Pf
Removes the space between its argument
.Pq Dq prefix
and the following macro.
Its syntax is as follows:
.Pp
.D1 .Pf Ar prefix macro arguments ...
.Pp
This is equivalent to:
.Pp
.D1 .No Ar prefix No \&Ns Ar macro arguments ...
.Pp
Examples:
.Dl ".Pf $ Ar variable_name"
.Dl ".Pf 0x Ar hex_digits"
.Pp
See also
.Sx \&Ns
and
.Sx \&Sm .
.Ss \&Po
Multi-line version of
.Sx \&Pq .
.Ss \&Pp
Break a paragraph.
This will assert vertical space between prior and subsequent macros
and/or text.
.Pp
Paragraph breaks are not needed before or after
.Sx \&Sh
or
.Sx \&Ss
macros or before displays
.Pq Sx \&Bd
or lists
.Pq Sx \&Bl
unless the
.Fl compact
flag is given.
.Ss \&Pq
Parenthesised enclosure.
.Pp
See also
.Sx \&Po .
.Ss \&Qc
Close quoted context opened by
.Sx \&Qo .
.Ss \&Ql
Format a single-quoted literal.
See also
.Sx \&Qq
and
.Sx \&Sq .
.Ss \&Qo
Multi-line version of
.Sx \&Qq .
.Ss \&Qq
Encloses its arguments in
.Qq typewriter
double-quotes.
Consider using
.Sx \&Dq .
.Pp
See also
.Sx \&Dq ,
.Sx \&Sq ,
```

```
2108 and
2109 .Sx \&Qo .
2110 .Ss \&Re
2111 Close an
2112 .Sx \&Rs
2113 block.
2114 Does not have any tail arguments.
2115 .Ss \&Rs
2116 Begin a bibliographic
2117 .Pq Dq reference
2118 block.
2119 Does not have any head arguments.
2120 The block macro may only contain
2121 .Sx \&%A ,
2122 .Sx \&%B ,
2123 .Sx \&%C ,
2124 .Sx \&%D ,
2125 .Sx \&%I ,
2126 .Sx \&%J ,
2127 .Sx \&%N ,
2128 .Sx \&%O ,
2129 .Sx \&%P ,
2130 .Sx \&%Q ,
2131 .Sx \&%R ,
2132 .Sx \&%T ,
2133 .Sx \&%U ,
2134 and
2135 .Sx \&%V
2136 child macros (at least one must be specified).
2137 .Pp
2138 Examples:
2139 .Bd -literal -offset indent -compact
2140 \&.Rs
2141 \&.%A J. E. Hopcroft
2142 \&.%A J. D. Ullman
2143 \&.%B Introduction to Automata Theory, Languages, and Computation
2144 \&.%I Addison-Wesley
2145 \&.%C Reading, Massachusettes
2146 \&.%D 1979
2147 \&.Re
2148 .Ed
2149 .Pp
2150 If an
2151 .Sx \&Rs
2152 block is used within a SEE ALSO section, a vertical space is asserted
2153 before the rendered output, else the block continues on the current
2154 line.
2155 .Ss \&Rv
2156 Insert a standard sentence regarding a function call’s return value of 0
2157 on success and \-1 on error, with the
2158 .Va errno
2159 libc global variable set on error.
2160 Its syntax is as follows:
2161 .Pp
2162 .D1 Pf \. Sx \&Rv Fl std Op Ar function ...
2163 .Pp
2164 If
2165 .Ar function
2166 is not specified, the document’s name set by
2167 .Sx \&Nm
2168 is used.
2169 Multiple
2170 .Ar function
2171 arguments are treated as separate functions.
2172 .Pp
2173 See also
```



```
2174 .Sx \&Ex .
2175 .Ss \&Sc
2176 Close single-quoted context opened by
2177 .Sx \&So .
2178 .Ss \&Sh
2179 Begin a new section.
2180 For a list of conventional manual sections, see
2181 .Sx MANUAL STRUCTURE .
2182 These sections should be used unless it’s absolutely necessary that
2183 custom sections be used.
2184 .Pp
2185 Section names should be unique so that they may be keyed by
2186 .Sx \&Sx .
2187 Although this macro is parsed, it should not consist of child node or it
2188 may not be linked with
2189 .Sx \&Sx .
2190 .Pp
2191 See also
2192 .Sx \&Pp ,
2193 .Sx \&Ss ,
2194 and
2195 .Sx \&Sx .
2196 .Ss \&Sm
2197 Switches the spacing mode for output generated from macros.
2198 Its syntax is as follows:
2199 .Pp
2200 .D1 Pf \. Sx \&Sm Cm on | off
2201 .Pp
2202 By default, spacing is
2203 .Cm on .
2204 When switched
2205 .Cm off ,
2206 no white space is inserted between macro arguments and between the
2207 output generated from adjacent macros, but text lines
2208 still get normal spacing between words and sentences.
2209 .Ss \&So
2210 Multi-line version of
2211 .Sx \&Sq .
2212 .Ss \&Sq
2213 Encloses its arguments in
2214 .Sq typewriter
2215 single-quotes.
2216 .Pp
2217 See also
2218 .Sx \&Dq ,
2219 .Sx \&Qq ,
2220 and
2221 .Sx \&So .
2222 .Ss \&Ss
2223 Begin a new subsection.
2224 Unlike with
2225 .Sx \&Sh ,
2226 there is no convention for the naming of subsections.
2227 Except
2228 .Em DESCRIPTION ,
2229 the conventional sections described in
2230 .Sx MANUAL STRUCTURE
2231 rarely have subsections.
2232 .Pp
2233 Sub-section names should be unique so that they may be keyed by
2234 .Sx \&Sx .
2235 Although this macro is parsed, it should not consist of child node or it
2236 may not be linked with
2237 .Sx \&Sx .
2238 .Pp
2239 See also
```

```
.Sx \&Pp ,
.Sx \&Sh ,
and
.Sx \&Sx .
.Ss \&St
Replace an abbreviation for a standard with the full form.
The following standards are recognised:
.Pp
.Bl -tag -width "-p1003.1g-2000X" -compact
.It \-p1003.1-88
.St -p1003.1-88
.It \-p1003.1-90
.St -p1003.1-90
.It \-p1003.1-96
.St -p1003.1-96
.It \-p1003.1-2001
.St -p1003.1-2001
.It \-p1003.1-2004
.St -p1003.1-2004
.It \-p1003.1-2008
.St -p1003.1-2008
.It \-p1003.1
.St -p1003.1
.It \-p1003.1b
.St -p1003.1b
.It \-p1003.1b-93
.St -p1003.1b-93
.It \-p1003.1c-95
.St -p1003.1c-95
.It \-p1003.1g-2000
.St -p1003.1g-2000
.It \-p1003.1i-95
.St -p1003.1i-95
.It \-p1003.2-92
.St -p1003.2-92
.It \-p1003.2a-92
.St -p1003.2a-92
.It \-p1387.2-95
.St -p1387.2-95
.It \-p1003.2
.St -p1003.2
.It \-p1387.2
.St -p1387.2
.It \-isoC
.St -isoC
.It \-isoC-90
.St -isoC-90
.It \-isoC-amd1
.St -isoC-amd1
.It \-isoC-tcor1
.St -isoC-tcor1
.It \-isoC-tcor2
.St -isoC-tcor2
.It \-isoC-99
.St -isoC-99
.It \-isoC-2011
.St -isoC-2011
.It \-iso9945-1-90
.St -iso9945-1-90
.It \-iso9945-1-96
.St -iso9945-1-96
.It \-iso9945-2-93
.St -iso9945-2-93
.It \-ansiC
.St -ansiC
.It \-ansiC-89
```



```
.St -ansiC-89
.It \-ansiC-99
.St -ansiC-99
.It \-ieee754
.St -ieee754
.It \-iso8802-3
.St -iso8802-3
.It \-iso8601
.St -iso8601
.It \-ieee1275-94
.St -ieee1275-94
.It \-xpg3
.St -xpg3
.It \-xpg4
.St -xpg4
.It \-xpg4.2
.St -xpg4.2
.It \-xpg4.3
.St -xpg4.3
.It \-xbd5
.St -xbd5
.It \-xcu5
.St -xcu5
.It \-xsh5
.St -xsh5
.It \-xns5
.St -xns5
.It \-xns5.2
.St -xns5.2
.It \-xns5.2d2.0
.St -xns5.2d2.0
.It \-xcurses4.2
.St -xcurses4.2
.It \-susv2
.St -susv2
.It \-susv3
.St -susv3
.It \-svid4
.St -svid4
.El
.Ss \&Sx
Reference a section or subsection in the same manual page.
The referenced section or subsection name must be identical to the
enclosed argument, including whitespace.
.Pp
Examples:
.Dl \&.Sx MANUAL STRUCTURE
.Pp
See also
.Sx \&Sh
and
.Sx \&Ss .
.Ss \&Sy
Format enclosed arguments in symbolic
.Pq Dq boldface .
Note that this is a presentation term and should not be used for
stylistically decorating technical terms.
.Pp
See also
.Sx \&Bf ,
.Sx \&Em ,
.Sx \&Li ,
and
.Sx \&No .
.Ss \&Ta
Table cell separator in
```

```
.Sx \&Bl Fl column
lists; can only be used below
.Sx \&It .
.Ss \&Tn
Format a tradename.
.Pp
Since this macro is often implemented to use a small caps font,
it has historically been used for acronyms (like ASCII) as well.
Such usage is not recommended because it would use the same macro
sometimes for semantical annotation, sometimes for physical formatting.
.Pp
Examples:
.Dl \&.Tn IBM
.Ss \&Ud
Prints out
.Dq currently under development.
.Ss \&Ux
Format the UNIX name.
Accepts no argument.
.Pp
Examples:
.Dl \&.Ux
.Pp
See also
.Sx \&At ,
.Sx \&Bsx ,
.Sx \&Bx ,
.Sx \&Dx ,
.Sx \&Fx ,
.Sx \&Nx ,
and
.Sx \&Ox .
.Ss \&Va
A variable name.
.Pp
Examples:
.Dl \&.Va foo
.Dl \&.Va const char *bar ;
.Ss \&Vt
A variable type.
This is also used for indicating global variables in the
.Em SYNOPSIS
section, in which case a variable name is also specified.
Note that it accepts
.Sx Block partial-implicit
syntax when invoked as the first macro on an input line in the
.Em SYNOPSIS
section, else it accepts ordinary
.Sx In-line
syntax.
In the former case, this macro starts a new output line,
and a blank line is inserted in front if there is a preceding
function definition or include directive.
.Pp
Note that this should not be confused with
.Sx \&Ft ,
which is used for function return types.
.Pp
Examples:
.Dl \&.Vt unsigned char
.Dl \&.Vt extern const char * const sys_signame[] \&;
.Pp
See also
.Sx MANUAL STRUCTURE
and
.Sx \&Va .
```



```
.Ss \&Xc
Close a scope opened by
.Sx \&Xo .
.Ss \&Xo
Extend the header of an
.Sx \&It
macro or the body of a partial-implicit block macro
beyond the end of the input line.
This macro originally existed to work around the 9-argument limit
of historic
.Xr roff 5 .
.Ss \&Xr
Link to another manual
.Pq Qq cross-reference .
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&Xr Ar name section
.Pp
The
.Ar name
and
.Ar section
are the name and section of the linked manual.
If
.Ar section
is followed by non-punctuation, an
.Sx \&Ns
is inserted into the token stream.
This behaviour is for compatibility with
GNU troff.
.Pp
Examples:
.Dl \&.Xr mandoc 1
.Dl \&.Xr mandoc 1 \&;
.Dl \&.Xr mandoc 1 \&Ns s behaviour
.Ss \&br
Emits a line-break.
This macro should not be used; it is implemented for compatibility with
historical manuals.
.Pp
Consider using
.Sx \&Pp
in the event of natural paragraph breaks.
.Ss \&sp
Emits vertical space.
This macro should not be used; it is implemented for compatibility with
historical manuals.
Its syntax is as follows:
.Pp
.D1 Pf \. Sx \&sp Op Ar height
.Pp
The
.Ar height
argument must be formatted as described in
.Sx Scaling Widths .
If unspecified,
.Sx \&sp
asserts a single vertical space.
.Sh MACRO SYNTAX
The syntax of a macro depends on its classification.
In this section,
.Sq \-arg
refers to macro arguments, which may be followed by zero or more
.Sq parm
parameters;
.Sq \&Yo
```

```
opens the scope of a macro; and if specified,
.Sq \&Yc
closes it out.
.Pp
The
.Em Callable
column indicates that the macro may also be called by passing its name
as an argument to another macro.
For example,
.Sq \&.Op \&Fl O \&Ar file
produces
.Sq Op Fl O Ar file .
To prevent a macro call and render the macro name literally,
escape it by prepending a zero-width space,
.Sq \e& .
For example,
.Sq \&Op \e&Fl O
produces
.Sq Op \&Fl O .
If a macro is not callable but its name appears as an argument
to another macro, it is interpreted as opaque text.
For example,
.Sq \&.Fl \&Sh
produces
.Sq Fl \&Sh .
.Pp
The
.Em Parsed
column indicates whether the macro may call other macros by receiving
their names as arguments.
If a macro is not parsed but the name of another macro appears
as an argument, it is interpreted as opaque text.
.Pp
The
.Em Scope
column, if applicable, describes closure rules.
.Ss Block full-explicit
Multi-line scope closed by an explicit closing macro.
All macros contains bodies; only
.Sx \&Bf
and
.Pq optionally
.Sx \&Bl
contain a head.
.Bd -literal -offset indent
\&.Yo \(lB\-arg \(lBparm...\(rB\(rB \(lBhead...\(rB
\(lBbody...\(rB
\&.Yc
.Ed
.Bl -column "MacroX" "CallableX" "ParsedX" "closed by XXX" -offset indent
.It Em Macro Ta Em Callable Ta Em Parsed Ta Em Scope
.It Sx \&Bd  Ta    \&No     Ta    \&No     Ta    closed by Sx \&Ed
.It Sx \&Bf  Ta    \&No     Ta    \&No     Ta    closed by Sx \&Ef
.It Sx \&Bk  Ta    \&No     Ta    \&No     Ta    closed by Sx \&Ek
.It Sx \&Bl  Ta    \&No     Ta    \&No     Ta    closed by Sx \&El
.It Sx \&Ed  Ta    \&No     Ta    \&No     Ta    opened by Sx \&Bd
.It Sx \&Ef  Ta    \&No     Ta    \&No     Ta    opened by Sx \&Bf
.It Sx \&Ek  Ta    \&No     Ta    \&No     Ta    opened by Sx \&Bk
.It Sx \&El  Ta    \&No     Ta    \&No     Ta    opened by Sx \&Bl
.El
.Ss Block full-implicit
Multi-line scope closed by end-of-file or implicitly by another macro.
All macros have bodies; some
.Po
.Sx \&It Fl bullet ,
.Fl hyphen ,
```



```
.Fl dash ,
.Fl enum ,
.Fl item
.Pc
don't have heads; only one
.Po
.Sx \&It
in
.Sx \&Bl Fl column
.Pc
has multiple heads.
.Bd -literal -offset indent
\&.Yo \(lB\-arg \(lBparm...\(rB\(rB \(lBhead... \(lBTa head...\(rB\(rB
\(lBbody...\(rB
.Ed
.Bl -column "MacroX" "CallableX" "ParsedX" "closed by XXXXXXXXXX" -offset inden
.It Em Macro Ta Em Callable Ta Em Parsed Ta Em Scope
.It Sx \&It Ta \&No Ta Yes  Ta closed by Sx \&It , Sx \&El
.It Sx \&Nd Ta \&No Ta \&No Ta closed by Sx \&Sh
.It Sx \&Nm Ta \&No Ta Yes  Ta closed by Sx \&Nm , Sx \&Sh , Sx \&Ss
.It Sx \&Sh Ta \&No Ta Yes  Ta closed by Sx \&Sh
.It Sx \&Ss Ta \&No Ta Yes  Ta closed by Sx \&Sh , Sx \&Ss
.El
.Pp
Note that the
.Sx \&Nm
macro is a
.Sx Block full-implicit
macro only when invoked as the first macro
in a
.Em SYNOPSIS
section line, else it is
.Sx In-line .
.Ss Block partial-explicit
Like block full-explicit, but also with single-line scope.
Each has at least a body and, in limited circumstances, a head
.Po
.Sx \&Fo ,
.Sx \&Eo
.Pc
and/or tail
.Pq Sx \&Ec .
.Bd -literal -offset indent
\&.Yo \(lB\-arg \(lBparm...\(rB\(rB \(lBhead...\(rB
\(lBbody...\(rB
\&.Yc \(lBtail...\(rB

\&.Yo \(lB\-arg \(lBparm...\(rB\(rB \(lBhead...\(rB \
\(lBbody...\(rB \&Yc \(lBtail...\(rB
.Ed
.Bl -column "MacroX" "CallableX" "ParsedX" "closed by XXXX" -offset indent
.It Em Macro Ta Em Callable Ta Em Parsed Ta Em Scope
.It Sx \&Ac  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Ao
.It Sx \&Ao  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Ac
.It Sx \&Bc  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Bo
.It Sx \&Bo  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Bc
.It Sx \&Brc Ta    Yes      Ta    Yes      Ta    opened by Sx \&Bro
.It Sx \&Bro Ta    Yes      Ta    Yes      Ta    closed by Sx \&Brc
.It Sx \&Dc  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Do
.It Sx \&Do  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Dc
.It Sx \&Ec  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Eo
.It Sx \&Eo  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Ec
.It Sx \&Fc  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Fo
.It Sx \&Fo  Ta    \&No     Ta    \&No     Ta    closed by Sx \&Fc
.It Sx \&Oc  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Oo
.It Sx \&Oo  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Oc
```

```
.It Sx \&Pc  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Po
.It Sx \&Po  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Pc
.It Sx \&Qc  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Oo
.It Sx \&Qo  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Oc
.It Sx \&Re  Ta    \&No     Ta    \&No     Ta    opened by Sx \&Rs
.It Sx \&Rs  Ta    \&No     Ta    \&No     Ta    closed by Sx \&Re
.It Sx \&Sc  Ta    Yes      Ta    Yes      Ta    opened by Sx \&So
.It Sx \&So  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Sc
.It Sx \&Xc  Ta    Yes      Ta    Yes      Ta    opened by Sx \&Xo
.It Sx \&Xo  Ta    Yes      Ta    Yes      Ta    closed by Sx \&Xc
.El
.Ss Block partial-implicit
Like block full-implicit, but with single-line scope closed by the
end of the line.
.Bd -literal -offset indent
\&.Yo \(lB\-arg \(lBval...\(rB\(rB \(lBbody...\(rB \(lBres...\(rB
.Ed
.Bl -column "MacroX" "CallableX" "ParsedX" -offset indent
.It Em Macro Ta Em Callable Ta Em Parsed
.It Sx \&Aq  Ta    Yes      Ta    Yes
.It Sx \&Bq  Ta    Yes      Ta    Yes
.It Sx \&Brq Ta    Yes      Ta    Yes
.It Sx \&D1  Ta    \&No     Ta    \&Yes
.It Sx \&Dl  Ta    \&No     Ta    Yes
.It Sx \&Dq  Ta    Yes      Ta    Yes
.It Sx \&Op  Ta    Yes      Ta    Yes
.It Sx \&Pq  Ta    Yes      Ta    Yes
.It Sx \&Ql  Ta    Yes      Ta    Yes
.It Sx \&Qq  Ta    Yes      Ta    Yes
.It Sx \&Sq  Ta    Yes      Ta    Yes
.It Sx \&Vt  Ta    Yes      Ta    Yes
.El
.Pp
Note that the
.Sx \&Vt
macro is a
.Sx Block partial-implicit
only when invoked as the first macro
in a
.Em SYNOPSIS
section line, else it is
.Sx In-line .
.Ss Special block macro
The
.Sx \&Ta
macro can only be used below
.Sx \&It
in
.Sx \&Bl Fl column
lists.
It delimits blocks representing table cells;
these blocks have bodies, but no heads.
.Bl -column "MacroX" "CallableX" "ParsedX" "closed by XXXX" -offset indent
.It Em Macro Ta Em Callable Ta Em Parsed Ta Em Scope
.It Sx \&Ta  Ta    Yes      Ta    Yes    Ta closed by Sx \&Ta , Sx \&It
.El
.Ss In-line
Closed by the end of the line, fixed argument lengths,
and/or subsequent macros.
In-line macros have only text children.
If a number (or inequality) of arguments is
.Pq n ,
then the macro accepts an arbitrary number of arguments.
.Bd -literal -offset indent
\&.Yo \(lB\-arg \(lBval...\(rB\(rB \(lBargs...\(rB \(lBres...\(rB
```



```
\&.Yo \(lB\-arg \(lBval...\(rB\(rB \(lBargs...\(rB Yc...

\&.Yo \(lB\-arg \(lBval...\(rB\(rB arg0 arg1 argN
.Ed
.Bl -column "MacroX" "CallableX" "ParsedX" "Arguments" -offset indent
.It Em Macro Ta Em Callable Ta Em Parsed Ta Em Arguments
.It Sx \&%A  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%B  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%C  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%D  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%I  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%J  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%N  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%O  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%P  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%Q  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%R  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%T  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%U  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&%V  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&Ad  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&An  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Ap  Ta    Yes      Ta    Yes      Ta    0
.It Sx \&Ar  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&At  Ta    Yes      Ta    Yes      Ta    1
.It Sx \&Bsx Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Bt  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Bx  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Cd  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Cm  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Db  Ta    \&No     Ta    \&No     Ta    1
.It Sx \&Dd  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Dt  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Dv  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Dx  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Em  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&En  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Er  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Es  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Ev  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Ex  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Fa  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Fd  Ta    \&No     Ta    \&No     Ta    >0
.It Sx \&Fl  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Fn  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Fr  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Ft  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Fx  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Hf  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Ic  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&In  Ta    \&No     Ta    \&No     Ta    1
.It Sx \&Lb  Ta    \&No     Ta    \&No     Ta    1
.It Sx \&Li  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Lk  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Lp  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Ms  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Mt  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Nm  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&No  Ta    Yes      Ta    Yes      Ta    0
.It Sx \&Ns  Ta    Yes      Ta    Yes      Ta    0
.It Sx \&Nx  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Os  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Ot  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Ox  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Pa  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Pf  Ta    Yes      Ta    Yes      Ta    1
```

```
.It Sx \&Pp  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Rv  Ta    \&No     Ta    \&No     Ta    n
.It Sx \&Sm  Ta    \&No     Ta    \&No     Ta    1
.It Sx \&St  Ta    \&No     Ta    Yes      Ta    1
.It Sx \&Sx  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Sy  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Tn  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Ud  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&Ux  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Va  Ta    Yes      Ta    Yes      Ta    n
.It Sx \&Vt  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&Xr  Ta    Yes      Ta    Yes      Ta    >0
.It Sx \&br  Ta    \&No     Ta    \&No     Ta    0
.It Sx \&sp  Ta    \&No     Ta    \&No     Ta    1
.El
.Ss Delimiters
When a macro argument consists of one single input character
considered as a delimiter, the argument gets special handling.
This does not apply when delimiters appear in arguments containing
more than one character.
Consequently, to prevent special handling and just handle it
like any other argument, a delimiter can be escaped by prepending
a zero-width space
.Pq Sq \e& .
In text lines, delimiters never need escaping, but may be used
as normal punctuation.
.Pp
For many macros, when the leading arguments are opening delimiters,
these delimiters are put before the macro scope,
and when the trailing arguments are closing delimiters,
these delimiters are put after the macro scope.
For example,
.Pp
.D1 Pf \. \&Aq "( [ word ] ) ."
.Pp
renders as:
.Pp
.D1 Aq ( [ word ] ) .
.Pp
Opening delimiters are:
.Pp
.Bl -tag -width Ds -offset indent -compact
.It \&(
left parenthesis
.It \&[
left bracket
.El
.Pp
Closing delimiters are:
.Pp
.Bl -tag -width Ds -offset indent -compact
.It \&.
period
.It \&,
comma
.It \&:
colon
.It \&;
semicolon
.It \&)
right parenthesis
.It \&]
right bracket
.It \&?
question mark
.It \&!
```



```
exclamation mark
.El
.Pp
Note that even a period preceded by a backslash
.Pq Sq \e.\&
gets this special handling; use
.Sq \e&.
to prevent that.
.Pp
Many in-line macros interrupt their scope when they encounter
delimiters, and resume their scope when more arguments follow that
are not delimiters.
For example,
.Pp
.D1 Pf \. \&Fl "a ( b | c \e*(Ba d ) e"
.Pp
renders as:
.Pp
.D1 Fl a ( b | c \*(Ba d ) e
.Pp
This applies to both opening and closing delimiters,
and also to the middle delimiter:
.Pp
.Bl -tag -width Ds -offset indent -compact
.It \&|
vertical bar
.El
.Pp
As a special case, the predefined string \e*(Ba is handled and rendered
in the same way as a plain
.Sq \&|
character.
Using this predefined string is not recommended in new manuals.
.Ss Font handling
In
.Nm
documents, usage of semantic markup is recommended in order to have
proper fonts automatically selected; only when no fitting semantic markup
is available, consider falling back to
.Sx Physical markup
macros.
Whenever any
.Nm
macro switches the
.Xr roff 5
font mode, it will automatically restore the previous font when exiting
its scope.
Manually switching the font using the
.Xr roff 5
.Ql \ef
font escape sequences is never required.
.Sh COMPATIBILITY
This section documents compatibility between mandoc and other other
troff implementations, at this time limited to GNU troff
.Pq Qq groff .
The term
.Qq historic groff
refers to groff versions before 1.17,
which featured a significant update of the
.Pa doc.tmac
file.
.Pp
Heirloom troff, the other significant troff implementation accepting
\-mdoc, is similar to historic groff.
.Pp
The following problematic behaviour is found in groff:
```

```
.ds hist (Historic groff only.)
.Pp
.Bl -dash -compact
.It
Display macros
.Po
.Sx \&Bd ,
.Sx \&Dl ,
and
.Sx \&D1
.Pc
may not be nested.
\*[hist]
.It
.Sx \&At
with unknown arguments produces no output at all.
\*[hist]
Newer groff and mandoc print
.Qq AT&T UNIX
and the arguments.
.It
.Sx \&Bl Fl column
does not recognise trailing punctuation characters when they immediately
precede tabulator characters, but treats them as normal text and
outputs a space before them.
.It
.Sx \&Bd Fl ragged compact
does not start a new line.
\*[hist]
.It
.Sx \&Dd
with non-standard arguments behaves very strangely.
When there are three arguments, they are printed verbatim.
Any other number of arguments is replaced by the current date,
but without any arguments the string
.Dq Epoch
is printed.
.It
.Sx \&Fl
does not print a dash for an empty argument.
\*[hist]
.It
.Sx \&Fn
does not start a new line unless invoked as the line macro in the
.Em SYNOPSIS
section.
\*[hist]
.It
.Sx \&Fo
with
.Pf non- Sx \&Fa
children causes inconsistent spacing between arguments.
In mandoc, a single space is always inserted between arguments.
.It
.Sx \&Ft
in the
.Em SYNOPSIS
causes inconsistent vertical spacing, depending on whether a prior
.Sx \&Fn
has been invoked.
See
.Sx \&Ft
and
.Sx \&Fn
for the normalised behaviour in mandoc.
.It
```



```
.Sx \&In
ignores additional arguments and is not treated specially in the
.Em SYNOPSIS .
\*[hist]
.It
.Sx \&It
sometimes requires a
.Fl nested
flag.
\*[hist]
In new groff and mandoc, any list may be nested by default and
.Fl enum
lists will restart the sequence only for the sub-list.
.It
.Sx \&Li
followed by a delimiter is incorrectly used in some manuals
instead of properly quoting that character, which sometimes works with
historic groff.
.It
.Sx \&Lk
only accepts a single link-name argument; the remainder is misformatted.
.It
.Sx \&Pa
does not format its arguments when used in the FILES section under
certain list types.
.It
.Sx \&Ta
can only be called by other macros, but not at the beginning of a line.
.It
.Sx \&%C
is not implemented.
.It
Historic groff only allows up to eight or nine arguments per macro input
line, depending on the exact situation.
Providing more arguments causes garbled output.
The number of arguments on one input line is not limited with mandoc.
.It
Historic groff has many un-callable macros.
Most of these (excluding some block-level macros) are callable
in new groff and mandoc.
.It
.Sq \(ba
(vertical bar) is not fully supported as a delimiter.
\*[hist]
.It
.Sq \ef
.Pq font face
and
.Sq \ef
.Pq font family face
.Sx Text Decoration
escapes behave irregularly when specified within line-macro scopes.
.It
Negative scaling units return to prior lines.
Instead, mandoc truncates them to zero.
.El
.Pp
The following features are unimplemented in mandoc:
.Pp
.Bl -dash -compact
.It
.Sx \&Bd
.Fl file Ar file .
.It
.Sx \&Bd
.Fl offset Ar center
```

```
and
.Fl offset Ar right .
Groff does not implement centred and flush-right rendering either,
but produces large indentations.
.It
The
.Sq \eh
.Pq horizontal position ,
.Sq \ev
.Pq vertical position ,
.Sq \em
.Pq text colour ,
.Sq \eM
.Pq text filling colour ,
.Sq \ez
.Pq zero-length character ,
.Sq \ew
.Pq string length ,
.Sq \ek
.Pq horizontal position marker ,
.Sq \eo
.Pq text overstrike ,
and
.Sq \es
.Pq text size
escape sequences are all discarded in mandoc.
.It
The
.Sq \ef
scaling unit is accepted by mandoc, but rendered as the default unit.
.It
In quoted literals, groff allows pairwise double-quotes to produce a
standalone double-quote in formatted output.
This is not supported by mandoc.
.El
.Sh SEE ALSO
.Xr man 1 ,
.Xr mandoc 1 ,
.Xr eqn 5 ,
.Xr man 5 ,
.Xr mandoc_char 5 ,
.Xr roff 5 ,
.Xr tbl 5
.Sh HISTORY
The
.Nm
language first appeared as a troff macro package in
.Bx 4.4 .
It was later significantly updated by Werner Lemberg and Ruslan Ermilov
in groff-1.17.
The standalone implementation that is part of the
.Xr mandoc 1
utility written by Kristaps Dzonsons appeared in
.Ox 4.6 .
.Sh AUTHORS
The
.Nm
reference was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv .
```

```
**********************************************************
    7891 Tue Jul 15 13:48:11 2014
new/usr/src/man/man5/tbl.5
import complete (hopefully)
**********************************************************
```

```
.\"
.\" 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) 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.\"
.Dd Sep 3, 2011
.Dt TBL 5
.Os
.Sh NAME
.Nm tbl
.Nd tbl language reference for mandoc
.Sh DESCRIPTION
The
.Nm tbl
language is a table-formatting language.
It is used within
.Xr mdoc 5
and
.Xr man 5
.Ux
manual pages.
This manual describes the subset of the
.Nm
language accepted by the
.Xr mandoc 1
utility.
.Pp
Tables within
.Xr mdoc 5
or
.Xr man 5
are enclosed by the
.Sq TS
and
.Sq TE
macro tags, whose precise syntax is documented in
.Xr roff 5 .
Tables consist of a series of options on a single line, followed by the
table layout, followed by data.
.Pp
For example, the following creates a boxed table with digits centred in
the cells.
.Bd -literal -offset indent
\&.TS
tab(:) box;
c5 c5 c5.
1:2:3
4:5:6
\&.TE
```



```
.Ed
.Pp
When formatted, the following output is produced:
.Bd -filled -offset indent -compact
.TS
tab(:) box;
c5 c5 c5.
1:2:3
4:5:6
.TE
.Ed
.Pp
The
.Nm
implementation in
.Xr mandoc 1
is
.Ud
.Sh TABLE STRUCTURE
Tables are enclosed by the
.Sq TS
and
.Sq TE
.Xr roff 5
macros.
A table consists of an optional single line of table
.Sx Options
terminated by a semicolon, followed by one or more lines of
.Sx Layout
specifications terminated by a period, then
.Sx Data .
All input must be 7-bit ASCII.
Example:
.Bd -literal -offset indent
\&.TS
box tab(:);
c | c
| c | c.
1:2
3:4
\&.TE
.Ed
.Pp
Table data is
.Em pre-processed ,
that is, data rows are parsed then inserted into the underlying stream
of input data.
This allows data rows to be interspersed by arbitrary
.Xr roff 5 ,
.Xr mdoc 5 ,
and
.Xr man 5
macros such as
.Bd -literal -offset indent
\&.TS
tab(:);
c c c.
1:2:3
\&.Ao
3:2:1
\&.Ac
\&.TE
.Ed
.Pp
in the case of
.Xr mdoc 5
```

```
or
.Bd -literal -offset indent
\&.TS
tab(:);
c c c.
\&.ds ab 2
1:\e*(ab:3
\&.I
3:2:1
\&.TE
.Ed
.Pp
in the case of
.Xr man 5 .
.Ss Options
The first line of a table consists of space-separated option keys and
modifiers terminated by a semicolon.
If the first line does not have a terminating semicolon, it is assumed
that no options are specified and instead a
.Sx Layout
is processed.
Some options accept arguments enclosed by parenthesis.
The following case-insensitive options are available:
.Bl -tag -width Ds
.It Cm center
This option is not supported by
.Xr mandoc 1 .
This may also be invoked with
.Cm centre .
.It Cm delim
Accepts a two-character argument.
This option is not supported by
.Xr mandoc 1 .
.It Cm expand
This option is not supported by
.Xr mandoc 1 .
.It Cm box
Draw a single-line box around the table.
This may also be invoked with
.Cm frame .
.It Cm doublebox
Draw a double-line box around the table.
This may also be invoked with
.Cm doubleframe .
.It Cm allbox
This option is not supported by
.Xr mandoc 1 .
.It Cm tab
Accepts a single-character argument.
This character is used as a delimiter between data cells, which otherwise
defaults to the tab character.
.It Cm linesize
Accepts a natural number (all digits).
This option is not supported by
.Xr mandoc 1 .
.It Cm nokeep
This option is not supported by
.Xr mandoc 1 .
.It Cm decimalpoint
Accepts a single-character argument.
This character will be used as the decimal point with the
.Cm n
layout key.
.It Cm nospaces
This option is not supported by
.Xr mandoc 1 .
```



```
.El
.Ss Layout
The table layout follows
.Sx Options
or a
.Sq \&T&
macro invocation.
Layout specifies how data rows are displayed on output.
Each layout line corresponds to a line of data; the last layout line
applies to all remaining data lines.
Layout lines may also be separated by a comma.
Each layout cell consists of one of the following case-insensitive keys:
.Bl -tag -width Ds
.It Cm c
Centre a literal string within its column.
.It Cm r
Right-justify a literal string within its column.
.It Cm l
Left-justify a literal string within its column.
.It Cm n
Justify a number around its last decimal point.
If the decimal point is not found on the number, it’s assumed to trail
the number.
.It Cm s
Horizontally span columns from the last
.No non- Ns Cm s
data cell.
It is an error if spanning columns follow a
.Cm \-
or
.Cm \(ba
cell, or come first.
This option is not supported by
.Xr mandoc 1 .
.It Cm a
Left-justify a literal string and pad with one space.
.It Cm ^
Vertically span rows from the last
.No non- Ns Cm ^
data cell.
It is an error to invoke a vertical span on the first layout row.
Unlike a horizontal spanner, you must specify an empty cell (if it not
empty, the data is discarded) in the corresponding data cell.
.It Cm \-
Replace the data cell (its contents will be lost) with a single
horizontal line.
This may also be invoked with
.Cm _ .
.It Cm =
Replace the data cell (its contents will be lost) with a double
horizontal line.
.It Cm \(ba
Emit a vertical bar instead of data.
.It Cm \(ba\(ba
Emit a double-vertical bar instead of data.
.El
.Pp
Keys may be followed by a set of modifiers.
A modifier is either a modifier key or a natural number for specifying
the minimum width of a column.
The following case-insensitive modifier keys are available:
.Cm z ,
.Cm u ,
.Cm e ,
.Cm t ,
.Cm d ,
```

```
.Cm b ,
.Cm i ,
.Cm r ,
and
.Cm f
.Po
followed by
.Cm b ,
.Cm i ,
.Cm r ,
.Cm 3 ,
.Cm 2 ,
or
.Cm 1
.Pc .
All of these are ignored by
.Xr mandoc 1 .
.Pp
For example, the following layout specifies a centre-justified column of
minimum width 10, followed by vertical bar, followed by a left-justified
column of minimum width 10, another vertical bar, then a column
justified about the decimal point in numbers:
.Pp
.Dl c10 | l10 | n
.Ss Data
The data section follows the last layout row.
By default, cells in a data section are delimited by a tab.
This behaviour may be changed with the
.Cm tab
option.
If
.Cm _
or
.Cm =
is specified, a single or double line, respectively, is drawn across the
data field.
If
.Cm \e-
or
.Cm \e=
is specified, a line is drawn within the data field (i.e. terminating
within the cell and not draw to the border).
If the last cell of a line is
.Cm T{ ,
all subsequent lines are included as part of the cell until
.Cm T}
is specified as its own data cell.
It may then be followed by a tab
.Pq or as designated by Cm tab
or an end-of-line to terminate the row.
.Sh COMPATIBILITY
This section documents compatibility between mandoc and other
.Nm
implementations, at this time limited to GNU tbl.
.Pp
.Bl -dash -compact
.It
In GNU tbl, comments and macros are disallowed prior to the data block
of a table.
The
.Xr mandoc 1
implementation allows them.
.El
.Sh SEE ALSO
.Xr mandoc 1 ,
.Xr man 5 ,
```



```
.Xr mandoc_char 5 ,
.Xr mdoc 5 ,
.Xr roff 5
.Rs
.%A M. E. Lesk
.%T Tbl\(emA Program to Format Tables
.%D June 11, 1976
.Re
.Sh HISTORY
The tbl utility, a preprocessor for troff, was originally written by M.
E. Lesk at Bell Labs in 1975.
The GNU reimplementation of tbl, part of the groff package, was released
in 1990 by James Clark.
A standalone tbl implementation was written by Kristaps Dzonsons in
2010.
This formed the basis of the implementation that is part of the
.Xr mandoc 1
utility.
.Sh AUTHORS
This
.Nm
reference was written by
.An Kristaps Dzonsons ,
.Mt kristaps@bsd.lv .
```

```
**********************************************************
    2381 Tue Jul 15 13:48:11 2014
new/usr/src/pkg/manifests/system-man.mf
Finish integration.  Use mandoc_preconv, etc.
import complete (hopefully)
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

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

  17 set name=pkg.fmri value=pkg:/system/man@$(PKGVERS)
  18 set name=pkg.description \
  19     value="utilities for display and formatting of reference manual pages"
  20 set name=pkg.summary value="Reference Manual Pages Tools"
  21 set name=info.classification \
  22     value="org.opensolaris.category.2008:System/Text Tools"
  23 set name=variant.arch value=$(ARCH)
  24 dir path=usr/bin
  25 dir path=usr/lib
  26 dir path=usr/share
  27 dir path=usr/share/man
  28 dir path=usr/share/man/man1
  29 dir path=usr/share/man/man5
  30 file path=usr/bin/man mode=0555
  31 file path=usr/bin/mandoc mode=0555
  32 file path=usr/lib/mandoc_preconv mode=0555
  33 file path=usr/share/man/man1/apropos.1
  34 file path=usr/share/man/man1/man.1
  35 file path=usr/share/man/man1/mandoc.1
  36 file path=usr/share/man/man5/eqn.5
  37 file path=usr/share/man/man5/man.5
  38 file path=usr/share/man/man5/mandoc_char.5
  39 file path=usr/share/man/man5/mandoc_roff.5
  40 file path=usr/share/man/man5/mdoc.5
  41 file path=usr/share/man/man5/tbl.5
  42 hardlink path=usr/bin/apropos target=../../usr/bin/man
  43 hardlink path=usr/bin/whatis target=../../usr/bin/man
  44 license lic_CDDL license=lic_CDDL
  45 license usr/src/cmd/man/THIRDPARTYLICENSE \
  46     license=usr/src/cmd/man/THIRDPARTYLICENSE
  47 license usr/src/cmd/mandoc/THIRDPARTYLICENSE \
  48     license=usr/src/cmd/mandoc/THIRDPARTYLICENSE
  49 link path=usr/man target=./share/man
  50 link path=usr/share/man/man1/whatis.1 target=apropos.1
  51 # arguably we also need lp, for man -t support, but really we don't
  52 # want to make this mandatory, so we don't express teh dependency here.
  53 # gzcat/bzcat are used for displaying compressed manpages.  However,
  54 # as we don't format such pages this way by default, lets leave the
  55 # dependency out.
  56 #depend fmri=compress/bzip2 type=require
  57 #depend fmri=compress/gzip type=require
  58 # less is the default (per user environment) pager.  We really should just
  59 # import this into illumos-gate.
  60 depend fmri=text/less type=require
```

```
**********************************************************
   16215 Tue Jul 15 13:48:11 2014
new/usr/src/pkg/manifests/text-doctools.mf
Finish integration.  Use mandoc_preconv, etc.
import complete (hopefully)
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  26 #

  28 # man moved to system/man
  29 set name=pkg.fmri value=pkg:/text/doctools@$(PKGVERS)
  30 set name=pkg.description \
  31     value="utilities and fonts for development, display, and production of docum
  32 set name=pkg.summary value="Documentation Tools"
  33 set name=info.classification \
  34     value="org.opensolaris.category.2008:System/Text Tools"
  35 set name=variant.arch value=$(ARCH)
  36 dir path=usr group=sys
  37 dir path=usr/bin
  38 dir path=usr/lib
  39 dir path=usr/lib/font
  40 dir path=usr/lib/font/devpost group=lp
  41 dir path=usr/lib/font/devpost/charlib group=lp
  42 dir path=usr/lib/font/devpost/fontmaps group=lp
  43 dir path=usr/lib/refer
  44 dir path=usr/lib/refer/papers
  45 dir path=usr/share
  46 dir path=usr/share/lib
  47 dir path=usr/share/lib/nterm
  48 dir path=usr/share/lib/pub
  49 dir path=usr/share/lib/tmac
  50 dir path=usr/share/man
  51 dir path=usr/share/man/man1
  52 dir path=usr/share/man/man1m
  53 dir path=usr/share/man/man5
  54 file path=usr/bin/addbib mode=0555
  53 file path=usr/bin/apropos mode=0555
  55 file path=usr/bin/checkeq mode=0555
  56 file path=usr/bin/checknr mode=0555
  57 file path=usr/bin/deroff mode=0555
  58 file path=usr/bin/diffmk mode=0555
  59 file path=usr/bin/eqn mode=0555
```



```
  60 file path=usr/bin/indxbib mode=0555
  61 file path=usr/bin/lookbib mode=0555
  62 file path=usr/bin/neqn mode=0555
  63 file path=usr/bin/nroff mode=0555
  64 file path=usr/bin/refer mode=0555
  65 file path=usr/bin/roffbib mode=0555
  66 file path=usr/bin/soelim mode=0555
  67 file path=usr/bin/sortbib mode=0555
  68 file path=usr/bin/ta mode=0555
  69 file path=usr/bin/tbl mode=0555
  70 file path=usr/bin/troff mode=0555
  71 file path=usr/bin/ul mode=0555
  72 file path=usr/bin/vgrind mode=0555
  73 file path=usr/lib/font/devpost/AB group=lp mode=0444
  74 file path=usr/lib/font/devpost/AB.name group=lp mode=0444
  75 file path=usr/lib/font/devpost/AB.out group=lp mode=0444
  76 file path=usr/lib/font/devpost/AI group=lp mode=0444
  77 file path=usr/lib/font/devpost/AI.name group=lp mode=0444
  78 file path=usr/lib/font/devpost/AI.out group=lp mode=0444
  79 file path=usr/lib/font/devpost/AR group=lp mode=0444
  80 file path=usr/lib/font/devpost/AR.name group=lp mode=0444
  81 file path=usr/lib/font/devpost/AR.out group=lp mode=0444
  82 file path=usr/lib/font/devpost/AX group=lp mode=0444
  83 file path=usr/lib/font/devpost/AX.name group=lp mode=0444
  84 file path=usr/lib/font/devpost/AX.out group=lp mode=0444
  85 file path=usr/lib/font/devpost/B group=lp mode=0444
  86 file path=usr/lib/font/devpost/B.name group=lp mode=0444
  87 file path=usr/lib/font/devpost/B.out group=lp mode=0444
  88 file path=usr/lib/font/devpost/BI group=lp mode=0444
  89 file path=usr/lib/font/devpost/BI.name group=lp mode=0444
  90 file path=usr/lib/font/devpost/BI.out group=lp mode=0444
  91 file path=usr/lib/font/devpost/CB group=lp mode=0444
  92 file path=usr/lib/font/devpost/CB.name group=lp mode=0444
  93 file path=usr/lib/font/devpost/CB.out group=lp mode=0444
  94 file path=usr/lib/font/devpost/CI group=lp mode=0444
  95 file path=usr/lib/font/devpost/CI.name group=lp mode=0444
  96 file path=usr/lib/font/devpost/CI.out group=lp mode=0444
  97 file path=usr/lib/font/devpost/CO group=lp mode=0444
  98 file path=usr/lib/font/devpost/CO.name group=lp mode=0444
  99 file path=usr/lib/font/devpost/CO.out group=lp mode=0444
 100 file path=usr/lib/font/devpost/CW group=lp mode=0444
 101 file path=usr/lib/font/devpost/CW.name group=lp mode=0444
 102 file path=usr/lib/font/devpost/CW.out group=lp mode=0444
 103 file path=usr/lib/font/devpost/CX group=lp mode=0444
 104 file path=usr/lib/font/devpost/CX.name group=lp mode=0444
 105 file path=usr/lib/font/devpost/CX.out group=lp mode=0444
 106 file path=usr/lib/font/devpost/DESC group=lp mode=0444
 107 file path=usr/lib/font/devpost/DESC.out group=lp mode=0444
 108 file path=usr/lib/font/devpost/G.out group=lp mode=0444
 109 file path=usr/lib/font/devpost/GI.out group=lp mode=0444
 110 file path=usr/lib/font/devpost/GR group=lp mode=0444
 111 file path=usr/lib/font/devpost/GR.name group=lp mode=0444
 112 file path=usr/lib/font/devpost/GR.out group=lp mode=0444
 113 file path=usr/lib/font/devpost/H group=lp mode=0444
 114 file path=usr/lib/font/devpost/H.name group=lp mode=0444
 115 file path=usr/lib/font/devpost/H.out group=lp mode=0444
 116 file path=usr/lib/font/devpost/HB group=lp mode=0444
 117 file path=usr/lib/font/devpost/HB.name group=lp mode=0444
 118 file path=usr/lib/font/devpost/HB.out group=lp mode=0444
 119 file path=usr/lib/font/devpost/HI group=lp mode=0444
 120 file path=usr/lib/font/devpost/HI.name group=lp mode=0444
 121 file path=usr/lib/font/devpost/HI.out group=lp mode=0444
 122 file path=usr/lib/font/devpost/HK.out group=lp mode=0444
 123 file path=usr/lib/font/devpost/HL.out group=lp mode=0444
 124 file path=usr/lib/font/devpost/HM.out group=lp mode=0444
 125 file path=usr/lib/font/devpost/HX group=lp mode=0444
```

```
126 file path=usr/lib/font/devpost/HX.name group=lp mode=0444
127 file path=usr/lib/font/devpost/HX.out group=lp mode=0444
128 file path=usr/lib/font/devpost/I group=lp mode=0444
129 file path=usr/lib/font/devpost/I.name group=lp mode=0444
130 file path=usr/lib/font/devpost/I.out group=lp mode=0444
131 file path=usr/lib/font/devpost/JB group=lp mode=0444
132 file path=usr/lib/font/devpost/JB.name group=lp mode=0444
133 file path=usr/lib/font/devpost/JB.out group=lp mode=0444
134 file path=usr/lib/font/devpost/JI group=lp mode=0444
135 file path=usr/lib/font/devpost/JI.name group=lp mode=0444
136 file path=usr/lib/font/devpost/JI.out group=lp mode=0444
137 file path=usr/lib/font/devpost/JR group=lp mode=0444
138 file path=usr/lib/font/devpost/JR.name group=lp mode=0444
139 file path=usr/lib/font/devpost/JR.out group=lp mode=0444
140 file path=usr/lib/font/devpost/JX group=lp mode=0444
141 file path=usr/lib/font/devpost/JX.name group=lp mode=0444
142 file path=usr/lib/font/devpost/JX.out group=lp mode=0444
143 file path=usr/lib/font/devpost/KB group=lp mode=0444
144 file path=usr/lib/font/devpost/KB.name group=lp mode=0444
145 file path=usr/lib/font/devpost/KB.out group=lp mode=0444
146 file path=usr/lib/font/devpost/KI group=lp mode=0444
147 file path=usr/lib/font/devpost/KI.name group=lp mode=0444
148 file path=usr/lib/font/devpost/KI.out group=lp mode=0444
149 file path=usr/lib/font/devpost/KR group=lp mode=0444
150 file path=usr/lib/font/devpost/KR.name group=lp mode=0444
151 file path=usr/lib/font/devpost/KR.out group=lp mode=0444
152 file path=usr/lib/font/devpost/KX group=lp mode=0444
153 file path=usr/lib/font/devpost/KX.name group=lp mode=0444
154 file path=usr/lib/font/devpost/KX.out group=lp mode=0444
155 file path=usr/lib/font/devpost/NB group=lp mode=0444
156 file path=usr/lib/font/devpost/NB.name group=lp mode=0444
157 file path=usr/lib/font/devpost/NB.out group=lp mode=0444
158 file path=usr/lib/font/devpost/NI group=lp mode=0444
159 file path=usr/lib/font/devpost/NI.name group=lp mode=0444
160 file path=usr/lib/font/devpost/NI.out group=lp mode=0444
161 file path=usr/lib/font/devpost/NR group=lp mode=0444
162 file path=usr/lib/font/devpost/NR.name group=lp mode=0444
163 file path=usr/lib/font/devpost/NR.out group=lp mode=0444
164 file path=usr/lib/font/devpost/NX group=lp mode=0444
165 file path=usr/lib/font/devpost/NX.name group=lp mode=0444
166 file path=usr/lib/font/devpost/NX.out group=lp mode=0444
167 file path=usr/lib/font/devpost/PA group=lp mode=0444
168 file path=usr/lib/font/devpost/PA.name group=lp mode=0444
169 file path=usr/lib/font/devpost/PA.out group=lp mode=0444
170 file path=usr/lib/font/devpost/PB group=lp mode=0444
171 file path=usr/lib/font/devpost/PB.name group=lp mode=0444
172 file path=usr/lib/font/devpost/PB.out group=lp mode=0444
173 file path=usr/lib/font/devpost/PI group=lp mode=0444
174 file path=usr/lib/font/devpost/PI.name group=lp mode=0444
175 file path=usr/lib/font/devpost/PI.out group=lp mode=0444
176 file path=usr/lib/font/devpost/PX group=lp mode=0444
177 file path=usr/lib/font/devpost/PX.name group=lp mode=0444
178 file path=usr/lib/font/devpost/PX.out group=lp mode=0444
179 file path=usr/lib/font/devpost/R group=lp mode=0444
180 file path=usr/lib/font/devpost/R.name group=lp mode=0444
181 file path=usr/lib/font/devpost/R.out group=lp mode=0444
182 file path=usr/lib/font/devpost/S group=lp mode=0444
183 file path=usr/lib/font/devpost/S.name group=lp mode=0444
184 file path=usr/lib/font/devpost/S.out group=lp mode=0444
185 file path=usr/lib/font/devpost/S1 group=lp mode=0444
186 file path=usr/lib/font/devpost/S1.name group=lp mode=0444
187 file path=usr/lib/font/devpost/S1.out group=lp mode=0444
188 file path=usr/lib/font/devpost/VB group=lp mode=0444
189 file path=usr/lib/font/devpost/VB.name group=lp mode=0444
190 file path=usr/lib/font/devpost/VB.out group=lp mode=0444
191 file path=usr/lib/font/devpost/VI group=lp mode=0444
```



```
192 file path=usr/lib/font/devpost/VI.name group=lp mode=0444
193 file path=usr/lib/font/devpost/VI.out group=lp mode=0444
194 file path=usr/lib/font/devpost/VR group=lp mode=0444
195 file path=usr/lib/font/devpost/VR.name group=lp mode=0444
196 file path=usr/lib/font/devpost/VR.out group=lp mode=0444
197 file path=usr/lib/font/devpost/VX group=lp mode=0444
198 file path=usr/lib/font/devpost/VX.name group=lp mode=0444
199 file path=usr/lib/font/devpost/VX.out group=lp mode=0444
200 file path=usr/lib/font/devpost/ZD group=lp mode=0444
201 file path=usr/lib/font/devpost/ZD.name group=lp mode=0444
202 file path=usr/lib/font/devpost/ZD.out group=lp mode=0444
203 file path=usr/lib/font/devpost/ZI group=lp mode=0444
204 file path=usr/lib/font/devpost/ZI.name group=lp mode=0444
205 file path=usr/lib/font/devpost/ZI.out group=lp mode=0444
206 file path=usr/lib/font/devpost/charlib/12 group=lp mode=0444
207 file path=usr/lib/font/devpost/charlib/14 group=lp mode=0444
208 file path=usr/lib/font/devpost/charlib/34 group=lp mode=0444
209 file path=usr/lib/font/devpost/charlib/BRACKETS_NOTE group=lp mode=0444
210 file path=usr/lib/font/devpost/charlib/Fi group=lp mode=0444
211 file path=usr/lib/font/devpost/charlib/Fl group=lp mode=0444
212 file path=usr/lib/font/devpost/charlib/L1 group=lp mode=0444
213 file path=usr/lib/font/devpost/charlib/L1.map group=lp mode=0444
214 file path=usr/lib/font/devpost/charlib/Lb group=lp mode=0444
215 file path=usr/lib/font/devpost/charlib/Lb.map group=lp mode=0444
216 file path=usr/lib/font/devpost/charlib/README group=lp mode=0444
217 file path=usr/lib/font/devpost/charlib/Sl group=lp mode=0444
218 file path=usr/lib/font/devpost/charlib/bx group=lp mode=0444
219 file path=usr/lib/font/devpost/charlib/ci group=lp mode=0444
220 file path=usr/lib/font/devpost/charlib/ff group=lp mode=0444
221 file path=usr/lib/font/devpost/charlib/lc group=lp mode=0444
222 file path=usr/lib/font/devpost/charlib/lf group=lp mode=0444
223 file path=usr/lib/font/devpost/charlib/lh group=lp mode=0444
224 file path=usr/lib/font/devpost/charlib/ob group=lp mode=0444
225 file path=usr/lib/font/devpost/charlib/rc group=lp mode=0444
226 file path=usr/lib/font/devpost/charlib/rf group=lp mode=0444
227 file path=usr/lib/font/devpost/charlib/rh group=lp mode=0444
228 file path=usr/lib/font/devpost/charlib/sq group=lp mode=0444
229 file path=usr/lib/font/devpost/charlib/~ group=lp mode=0444
230 file path=usr/lib/font/devpost/fontmaps/post group=lp mode=0444
231 file path=usr/lib/font/makedev group=lp mode=0555
231 file path=usr/lib/getNAME mode=0555
232 file path=usr/lib/makewhatis mode=0555
232 file path=usr/lib/refer/hunt mode=0555
233 file path=usr/lib/refer/inv mode=0555
234 file path=usr/lib/refer/mkey mode=0555
235 file path=usr/lib/refer/papers/Rbstjissue
236 file path=usr/lib/refer/papers/Rv7man
237 file path=usr/lib/refer/papers/runinv mode=0755
238 file path=usr/lib/vfontedpr mode=0555
239 file path=usr/lib/vgrindefs mode=0444
240 file path=usr/share/lib/nterm/tab.2631
241 file path=usr/share/lib/nterm/tab.2631-c
242 file path=usr/share/lib/nterm/tab.2631-e
243 file path=usr/share/lib/nterm/tab.300
244 file path=usr/share/lib/nterm/tab.300-12
245 file path=usr/share/lib/nterm/tab.300S
246 file path=usr/share/lib/nterm/tab.300S-12
247 file path=usr/share/lib/nterm/tab.37
248 file path=usr/share/lib/nterm/tab.382
249 file path=usr/share/lib/nterm/tab.4000A
250 file path=usr/share/lib/nterm/tab.450
251 file path=usr/share/lib/nterm/tab.450-12
252 file path=usr/share/lib/nterm/tab.832
253 file path=usr/share/lib/nterm/tab.8510
254 file path=usr/share/lib/nterm/tab.X
255 file path=usr/share/lib/nterm/tab.lp
```

```
256 file path=usr/share/lib/nterm/tab.tn300
257 file path=usr/share/lib/pub/ascii
258 file path=usr/share/lib/pub/eqnchar
259 file path=usr/share/lib/pub/greek
260 file path=usr/share/lib/pub/iso
261 file path=usr/share/lib/tmac/acm.me
262 file path=usr/share/lib/tmac/an
263 file path=usr/share/lib/tmac/ansun
264 file path=usr/share/lib/tmac/ansun.tbl
265 file path=usr/share/lib/tmac/bib
266 file path=usr/share/lib/tmac/chars.me
267 file path=usr/share/lib/tmac/deltext.me
268 file path=usr/share/lib/tmac/e
269 file path=usr/share/lib/tmac/eqn.me
270 file path=usr/share/lib/tmac/float.me
271 file path=usr/share/lib/tmac/footnote.me
272 file path=usr/share/lib/tmac/index.me
273 file path=usr/share/lib/tmac/local.me
274 file path=usr/share/lib/tmac/m
275 file path=usr/share/lib/tmac/mmn
276 file path=usr/share/lib/tmac/mmt
277 file path=usr/share/lib/tmac/ms.acc
278 file path=usr/share/lib/tmac/ms.cov
279 file path=usr/share/lib/tmac/ms.eqn
280 file path=usr/share/lib/tmac/ms.ref
281 file path=usr/share/lib/tmac/ms.tbl
282 file path=usr/share/lib/tmac/ms.ths
283 file path=usr/share/lib/tmac/ms.toc
284 file path=usr/share/lib/tmac/null.me
285 file path=usr/share/lib/tmac/refer.me
286 file path=usr/share/lib/tmac/s
287 file path=usr/share/lib/tmac/sh.me
288 file path=usr/share/lib/tmac/tbl.me
289 file path=usr/share/lib/tmac/thesis.me
290 file path=usr/share/lib/tmac/tmac.bib
291 file path=usr/share/lib/tmac/tmac.vgrind
292 file path=usr/share/lib/tmac/tz.map
293 file path=usr/share/lib/tmac/v
294 file path=usr/share/lib/tmac/vgrind
295 file path=usr/share/man/man1/addbib.1
297 file path=usr/share/man/man1/apropos.1
296 file path=usr/share/man/man1/checknr.1
297 file path=usr/share/man/man1/deroff.1
298 file path=usr/share/man/man1/diffmk.1
299 file path=usr/share/man/man1/eqn.1
300 file path=usr/share/man/man1/indxbib.1
301 file path=usr/share/man/man1/lookbib.1
304 file path=usr/share/man/man1/man.1
302 file path=usr/share/man/man1/nroff.1
303 file path=usr/share/man/man1/refer.1
304 file path=usr/share/man/man1/roffbib.1
305 file path=usr/share/man/man1/soelim.1
306 file path=usr/share/man/man1/sortbib.1
307 file path=usr/share/man/man1/tbl.1
308 file path=usr/share/man/man1/troff.1
309 file path=usr/share/man/man1/ul.1
310 file path=usr/share/man/man1/vgrind.1
314 file path=usr/share/man/man1/whatis.1
315 file path=usr/share/man/man1m/catman.1m
311 file path=usr/share/man/man5/eqnchar.5
317 file path=usr/share/man/man5/man.5
318 file path=usr/share/man/man5/mansun.5
312 file path=usr/share/man/man5/me.5
313 file path=usr/share/man/man5/mm.5
314 file path=usr/share/man/man5/ms.5
315 file path=usr/share/man/man5/vgrindefs.5
```



```
323 hardlink path=usr/bin/catman target=../../usr/bin/apropos
324 hardlink path=usr/bin/man target=../../usr/bin/apropos
325 hardlink path=usr/bin/whatis target=../../usr/bin/apropos
316 hardlink path=usr/lib/font/devpost/charlib/~= target=./~
317 hardlink path=usr/share/lib/nterm/tab.300s \
318     target=../../../../usr/share/lib/nterm/tab.300S
319 hardlink path=usr/share/lib/nterm/tab.300s-12 \
320     target=../../../../usr/share/lib/nterm/tab.300S-12
321 hardlink path=usr/share/lib/nterm/tab.4000a \
322     target=../../../../usr/share/lib/nterm/tab.4000A
323 legacy pkg=SUNWdoc \
324     desc="utilities and fonts for development, display, and production of docume
325     name="Documentation Tools"
326 license cr_Sun license=cr_Sun
327 license usr/src/cmd/checkeq/THIRDPARTYLICENSE \
328     license=usr/src/cmd/checkeq/THIRDPARTYLICENSE
329 license usr/src/cmd/checknr/THIRDPARTYLICENSE \
330     license=usr/src/cmd/checknr/THIRDPARTYLICENSE
331 license usr/src/cmd/eqn/THIRDPARTYLICENSE \
332     license=usr/src/cmd/eqn/THIRDPARTYLICENSE
343 license usr/src/cmd/man/src/THIRDPARTYLICENSE \
344     license=usr/src/cmd/man/src/THIRDPARTYLICENSE
333 license usr/src/cmd/refer/THIRDPARTYLICENSE \
334     license=usr/src/cmd/refer/THIRDPARTYLICENSE
335 license usr/src/cmd/soelim/THIRDPARTYLICENSE \
336     license=usr/src/cmd/soelim/THIRDPARTYLICENSE
337 license usr/src/cmd/tbl/THIRDPARTYLICENSE \
338     license=usr/src/cmd/tbl/THIRDPARTYLICENSE
339 license usr/src/cmd/ul/THIRDPARTYLICENSE \
340     license=usr/src/cmd/ul/THIRDPARTYLICENSE
341 license usr/src/cmd/vgrind/THIRDPARTYLICENSE \
342     license=usr/src/cmd/vgrind/THIRDPARTYLICENSE
343 link path=usr/lib/tmac target=../share/lib/tmac
356 link path=usr/man target=./share/man
344 link path=usr/share/man/man1/checkeq.1 target=eqn.1
345 link path=usr/share/man/man1/neqn.1 target=eqn.1
346 depend fmri=system/man type=require
359 # lp is used by man -t and -r
360 depend fmri=print/lp/print-client-commands type=require
361 # awk is used by catman (via makewhatis)
362 depend fmri=system/extended-system-utilities type=require
363 # gpic is used as a preprocessor by the apropos command
364 depend fmri=text/groff type=require
365 # less is the default (per user environment) pager
366 depend fmri=text/less type=require
```