```
**********************************************************
   72665 Wed Mar 21 13:33:29 2012
new/usr/src/cmd/format/startup.c
574 Minor issues in usr/src/cmd/format/startup.c
**********************************************************
```

```
   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  */
  21 /*
  22  * Copyright (c) 2012 Nexenta System Inc.
  23  *
  24  * Copyright (c) 2011 Gary Mills
  25  *
  26  * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
  27  */

  29 /*
  30  * This file contains the code to perform program startup.  This
  31  * includes reading the data file and the search for disks.
  32  */
  33 #include "global.h"

  35 #include <ctype.h>
  36 #include <stdlib.h>
  37 #include <unistd.h>
  38 #include <string.h>
  39 #include <strings.h>
  40 #include <fcntl.h>
  41 #include <errno.h>
  42 #include <memory.h>
  43 #include <dirent.h>
  44 #include <sys/fcntl.h>
  45 #include <sys/param.h>
  46 #include <sys/stat.h>

  48 #include "startup.h"
  49 #include "param.h"
  50 #include "label.h"
  51 #include "misc.h"
  52 #include "menu_command.h"
  53 #include "partition.h"
  54 #include "ctlr_scsi.h"

  56 #include "auto_sense.h"

  58 extern  struct  ctlr_type ctlr_types[];
  59 extern  int     nctypes;
  60 extern  struct  ctlr_ops        genericops;
  61 extern  long    strtol();
```



```
  63 extern  int     errno;

  65 #ifdef __STDC__

  67 /* Function prototypes for ANSI C Compilers */
  68 static void     usage(void);
  69 static int      sup_prxfile(void);
  70 static void     sup_setpath(void);
  71 static void     sup_setdtype(void);
  72 static int      sup_change_spec(struct disk_type *, char *);
  73 static void     sup_setpart(void);
  74 static void     search_for_logical_dev(char *devname);
  75 static void     add_device_to_disklist(char *devname, char *devpath);
  76 static int      disk_is_known(struct dk_cinfo *dkinfo);
  77 static void     datafile_error(char *errmsg, char *token);
  78 static void     search_duplicate_dtypes(void);
  79 static void     search_duplicate_pinfo(void);
  80 static void     check_dtypes_for_inconsistency(struct disk_type *dp1,
  81                 struct disk_type *dp2);
  82 static void     check_pinfo_for_inconsistency(struct partition_info *pp1,
  83                 struct partition_info *pp2);
  84 static uint_t   str2blks(char *str);
  85 static int      str2cyls(char *str);
  86 static struct   chg_list *new_chg_list(struct disk_type *);
  87 static char     *get_physical_name(char *);
  88 static void     sort_disk_list(void);
  89 static int      disk_name_compare(const void *, const void *);
  90 static void     make_controller_list(void);
  91 static void     check_for_duplicate_disknames(char *arglist[]);

  93 #else   /* __STDC__ */

  95 /* Function prototypes for non-ANSI C Compilers */
  96 static void     usage();
  97 static int      sup_prxfile();
  98 static void     sup_setpath();
  99 static void     sup_setdtype();
 100 static int      sup_change_spec();
 101 static void     sup_setpart();
 102 static void     search_for_logical_dev();
 103 static void     add_device_to_disklist();
 104 static int      disk_is_known();
 105 static void     datafile_error();
 106 static void     search_duplicate_dtypes();
 107 static void     search_duplicate_pinfo();
 108 static void     check_dtypes_for_inconsistency();
 109 static void     check_pinfo_for_inconsistency();
 110 static uint_t   str2blks();
 111 static int      str2cyls();
 112 static struct   chg_list *new_chg_list();
 113 static char     *get_physical_name();
 114 static void     sort_disk_list();
 115 static int      disk_name_compare();
 116 static void     make_controller_list();
 117 static void     check_for_duplicate_disknames();

 119 #endif  /* __STDC__ */

 121 #if defined(sparc)
 122 static char *other_ctlrs[] = {
 123         "ata"
 124         };
 125 #define OTHER_CTLRS 1

 127 #elif defined(i386)
```

```
 128 static char *other_ctlrs[] = {
 129         "ISP-80"
 130         };
 131 #define OTHER_CTLRS 2

 133 #else
 134 #error No Platform defined.
 135 #endif


 138 /*
 139  * This global is used to store the current line # in the data file.
 140  * It must be global because the I/O routines are allowed to side
 141  * effect it to keep track of backslashed newlines.
 142  */
 143 int     data_lineno;                    /* current line # in data file */

 145 /*
 146  * Search path as defined in the format.dat files
 147  */
 148 static char     **search_path = NULL;


 151 static int name_represents_wholedisk(char *name);

 153 static void get_disk_name(int fd, char *disk_name);

 155 /*
 156  * This routine digests the options on the command line.  It returns
 157  * the index into argv of the first string that is not an option.  If
 158  * there are none, it returns -1.
 159  */
 160 int
 161 do_options(int argc, char *argv[])
 162 {
 163         char    *ptr;
 164         int     i;
 165         int     next;

 167         /*
 168          * Default is no extended messages.  Can be enabled manually.
 169          */
 170         option_msg = 0;
 171         diag_msg = 0;
 172         expert_mode = 0;
 173         need_newline = 0;
 174         dev_expert = 0;

 176         /*
 177          * Loop through the argument list, incrementing each time by
 178          * an amount determined by the options found.
 179          */
 180         for (i = 1; i < argc; i = next) {
 181                 /*
 182                  * Start out assuming an increment of 1.
 183                  */
 184                 next = i + 1;
 185                 /*
 186                  * As soon as we hit a non-option, we're done.
 187                  */
 188                 if (*argv[i] != '-')
 189                         return (i);
 190                 /*
 191                  * Loop through all the characters in this option string.
 192                  */
 193                 for (ptr = argv[i] + 1; *ptr != '\0'; ptr++) {
```



```
 194                         /*
 195                          * Determine each option represented.  For options
 196                          * that use a second string, increase the increment
 197                          * of the main loop so they aren't re-interpreted.
 198                          */
 199                         switch (*ptr) {
 200                         case 's':
 201                         case 'S':
 202                                 option_s = 1;
 203                                 break;
 204                         case 'f':
 205                         case 'F':
 206                                 option_f = argv[next++];
 207                                 if (next > argc)
 208                                         goto badopt;
 209                                 break;
 210                         case 'l':
 211                         case 'L':
 212                                 option_l = argv[next++];
 213                                 if (next > argc)
 214                                         goto badopt;
 215                                 break;
 216                         case 'x':
 217                         case 'X':
 218                                 option_x = argv[next++];
 219                                 if (next > argc)
 220                                         goto badopt;
 221                                 break;
 222                         case 'd':
 223                         case 'D':
 224                                 option_d = argv[next++];
 225                                 if (next > argc)
 226                                         goto badopt;
 227                                 break;
 228                         case 't':
 229                         case 'T':
 230                                 option_t = argv[next++];
 231                                 if (next > argc)
 232                                         goto badopt;
 233                                 break;
 234                         case 'p':
 235                         case 'P':
 236                                 option_p = argv[next++];
 237                                 if (next > argc)
 238                                         goto badopt;
 239                                 break;
 240                         case 'm':
 241                                 option_msg = 1;
 242                                 break;
 243                         case 'M':
 244                                 option_msg = 1;
 245                                 diag_msg = 1;
 246                                 break;
 247                         case 'e':
 248                                 expert_mode = 1;
 249                                 break;
 250 #ifdef DEBUG
 251                         case 'z':
 252                                 dev_expert = 1;
 253                                 break;
 254 #endif
 255                         default:
 256 badopt:
 257                                 usage();
 258                                 break;
 259                         }
```

```
 260                 }
 261         }
 262         /*
 263          * All the command line strings were options.  Return that fact.
 264          */
 265         return (-1);
 266 }
______unchanged_portion_omitted_

2960 #define DISK_PREFIX     "/dev/rdsk/"

2962 /*
2963  * This Function checks if the non-conventional name is a a link to
2964  * one of the conventional whole disk name.
2965  */
2966 static int
2967 name_represents_wholedisk(char  *name)
2965 name_represents_wholedisk(name)
2966 char    *name;
2968 {
2969         char    symname[MAXPATHLEN];
2970         char    localname[MAXPATHLEN];
2971         char    *nameptr;
2972         ssize_t symname_size;
2973         size_t disk_prefix_len = strlen(DISK_PREFIX);

2975         if (strlcpy(localname, name, MAXPATHLEN) >= MAXPATHLEN)
2976                 return (1);     /* buffer overflow, reject this name */

2978         while ((symname_size = readlink(localname, symname, MAXPATHLEN - 1)) !=
2979                 symname[symname_size] = '\0';
2973         (void) memset(symname, 0, MAXPATHLEN);
2974         (void) memset(localname, 0, MAXPATHLEN);
2975         (void) strcpy(localname, name);

2977         while (readlink(localname, symname, MAXPATHLEN) != -1) {
2980                 nameptr = symname;
2981                 if (strncmp(symname, DISK_PREFIX, disk_prefix_len) == 0)
2982                         nameptr += disk_prefix_len;
2983                 if (conventional_name(nameptr))
2984                         return (!whole_disk_name(nameptr));
2979                 if (strncmp(symname, DISK_PREFIX, strlen(DISK_PREFIX)) == 0)
2980                         nameptr += strlen(DISK_PREFIX);
2981                 if (conventional_name(nameptr)) {
2982                         if (whole_disk_name(nameptr))
2983                                 return (0);
2984                         else
2985                                 return (1);
2986                 }
2985                 (void) strcpy(localname, symname);
2988                 (void) memset(symname, 0, MAXPATHLEN);
2986         }
2987         return (0);
2988 }
______unchanged_portion_omitted_
```