-- --

IEEE P1003.2 Draft 11.2 – September 1991

B.7 C Binding for Command Option Parsing

Function: *getopt* ()

B.7.1 Synopsis

```
#include <unistd.h>
int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```
1

B.7.2 Description

The *getopt* () function is a command-line parser that can be used by applications that follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9, and 10 in 2.10.2. The remaining guidelines are not addressed by *getopt* () and are the responsibility of the application.

The parameters *argc* and *argv* are the argument count and argument array as passed to *main* (). The argument *optstring* is a string of recognized option characters; if a character is followed by a colon, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as an extension.

The variable *optind* is the index of the next element of the *argv* [] vector to be processed. It is initialized to 1 by the system, and *getopt* () updates it when it finishes with each element of *argv* []. When an element of *argv* [] contains multiple option characters, it is unspecified how *getopt* () determines which options have already been processed.

The *getopt* () function shall return the next option character from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes 1
an argument, *getopt* () shall set the variable *optarg* to point to the option-argument as follows:

(1) If the option was the last character in the string pointed to by an element of *argv*, then *optarg* contains the next element of *argv*, and *optind* shall be incremented by 2. If the resulting value of *optind* is not less than *argc*, this indicates a missing option argument, and *getopt* () shall return an error indication.

(2) Otherwise, *optarg* points to the string following the option character in that element of *argv*, and *optind* shall be incremented by 1.

If, when *getopt* () is called, *argv* [*optind*] is **NULL**, **argv* [*optind*] is not the character −, or *argv* [*optind*] points to the string `"-"`, *getopt* () shall return −1 without changing *optind*. If *argv* [*optind*] points to the string `"--"`, *getopt* () shall return −1 after incrementing *optind*.

If *getopt* () encounters an option character that is not contained in *optstring* , it shall return the question-mark (`?`) character. If it detects a missing option argument, it shall return the colon character (`:`) if the first character of *optstring* was a colon, or a question-mark character otherwise. In either case, *getopt* () shall set the variable *optopt* to the option character that caused the error. If the application has not set the variable *opterr* to zero and the first character of *optstring* is not a colon, *getopt* () shall also print a diagnostic message to standard error using the formatting rules specified for the `getopts` utility (see 4.27.6.2).

B.7.3 Returns

The *getopt* () function shall return the next option character specified on the command line. The value −1 shall be returned when all command line options have been parsed.

B.7.4 Errors

If an invalid option is encountered, *getopt* () shall return a question-mark character. If an option with a missing option argument is encountered, *getopt* () shall return either a question-mark or a colon, as described previously.

B.7.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The *getopt* () function is only required to support option characters included in Guideline 3. Many historical implementations of *getopt* () support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multibyte option characters is only possible when such characters can be represented as type *int* .

The code fragment in Figure B-4 shows how one might process the arguments for a utility that can take the mutually exclusive options `a` and `b` and the options `f` and `o`, both of which require arguments.

The code in Figure B-4 accepts any of the following as equivalent:

```
cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path
```

History of Decisions Made

Support for the *optopt* variable was added in Draft 9. This documents historical practice, and allows the application to obtain the identity of the invalid option.

```
#include <unistd.h>
int main (int argc, char *argv[ ])
{
        int c, bflg, aflg, errflg = 0;
        char *ifile, *ofile;
        extern char *optarg;
        extern int optind, optopt;
        . . .
        while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg = 1;
                        else
                                aflg = 1;
                        break;
                case 'b':
                        if (aflg)
                                errflg = 1;
                        else
                                bflg = 1;
                                bproc( );
                        break;
                case 'f':
                        ifile = optarg;
                        break;
                case 'o':
                        ofile = optarg;
                        break;
                case ':':       /* -f or -o without option-arg */
                        fprintf (stderr,
                                "Option -%c requires an option-argument\n",
                                optopt);
                        errflg = 1;
                        break;
                case '?':
                        fprintf (stderr,
                                "Unrecognized option: -%c\n", optopt);
                        errflg = 1;
                        break;
                }
        }
        if (errflg) {
                fprintf(stderr, "usage: . . . ");
                exit(2);
        }
        for ( ; optind < argc; optind++) {
                if (access(argv[optind], R_OK)) {
        . . .
}
```

Figure B-4 – Argument Processing with *getopt* ()

The description was extensively rewritten in Draft 9 to be more explicit about how *optarg* and *optind* are set, and to recognize that this routine deals with a vector of string pointers, not directly with a shell command line.

The description was modified in Draft 9 to make it clear that *getopt* (), like the `getopts` utility, shall deal with option-arguments whether separated from the option by `<blank>`s or not. Note that the requirements on *getopt* () and `getopts` are more stringent than the Utility Syntax Guidelines.

The *getopt* () function has been changed to return `−1`, rather than `EOF`, so that *<stdio.h>* is not required.

The special significance of a colon as the first character of *optstring* was added in Draft 11 to make *getopt* () consistent with the `getopts` utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

B.8 C Binding for Generate Pathnames Matching a Pattern

Functions: *glob* (), *globfree* ()

B.8.1 Synopsis

```
#include <glob.h>

int glob(const char *pattern , int flags ,
        int (*errfunc )(const char *epath , int eerrno ), glob_t *pglob);

void globfree(glob_t *pglob);
```

B.8.2 Description

The *glob* () function is a pathname generator that implements the rules defined in 3.13, with optional support for rule (3) in 3.13.3.

The header `<glob.h>` defines the structure type *glob_t* , which includes at least the members shown in Table B-12.

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob* () function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob* () requires search permission on every component of a path except the last and read permission on each directory of any filename component of *pattern* that contains any of the special characters `*`, `?` or `[`. The *glob* () function stores the number of matched pathnames into *pglob->gl_pathc* and a pointer to a list of pointers to pathnames into *pglob->gl_pathv* . The pathnames are in sort order as defined by 2.2.2.30. The first pointer after the last pathname shall be **NULL** . If the pattern does not match any pathnames, the returned number of matched paths is set to zero.