NAME

getflags − collect flag arguments from command-line

SYNOPSIS

#include "mjsu.h"

CHAR *getflags(INT *pac, CHAR ***pav, CHAR *fmt, CHAR *usagetail, ...);

DESCRIPTION

getflags() sets variables (pointed-to by the arguments ...) from the flags in the command-line passed to the process (represented by **pac* and **pav*), according to the specifications of the format string *fmt*.

usagetail is either NULL or a text-string used to construct a fatal error-message, see RETURNS.

The command-line is given by the array of strings **pav*; The number of strings in the array is given by **pac*. It is assumed that the first string is the process name, and is to be skipped. Each succeeding string is taken to represent one or more flags if it begins with '-' and is not the string "-" or "--". Processing of flags stops when the first non-flag string is encountered.

getflags() updates **pac* and **pav* to reflect the consumption of recognised flags. The string "--" is a flag terminator which is consumed, whereas the string "-" is assumed to be a potential filename argument and is thus not consumed.

fmt is a sequence of flag-descriptors, each of which describes how to interpret a flag. The order of the flag variables in ... must match the order of flag-descriptors in *fmt*.

Each flag-descriptor consists of a sequence of alphabetic flag-name characters, optionally followed by a type-indicator, terminated by a ',' or the end of the format string.

For each of the strings at **pav*, the descriptors are tested in sequence until a matching flag-name is encountered or the end of the format-string is encountered. When a match is found, if the flag does not require a value, the remainder of the string is scanned for further flag-names; otherwise the remainder of the string, if any, or all of the next string, is taken as the flag's "value" and is interpreted as specified by the type-indicator.

The type-indicators are:

"#" the value is interpreted as a number and is converted to a **SHORT**. The value is interpreted as decimal if it does not start with "0", as heximadecimal if it starts with "0x" or "0X", or as octal otherwise. The corresponding argument is taken to be the address of a **SHORT**, which is set to the result of the conversion.

"##" is treated the same as '#', except that the value is converted to a **LONG**, and the corresponding argument is assumed to be the address of a **LONG** instead.

"*" the value is not specially interpreted. The corresponding argument is taken as the address of a pointer to **CHAR**, which is set to point at the value string.

"" a null type-indicator precludes a value. The corresponding argument is taken to be the address of a **BOOL**, which is set to YES.

"#^"

"##^"

"*^" are similar to "#", "##" and "*" respectively, except that the corresponding argument is taken to be the address of a structure in the form:

```
struct
        {
        UTINY maxnum;
        UTINY num;
        TYPE val[255];
        };
```

where *TYPE* is either **SHORT**, **LONG** or pointer to **CHAR**, as appropriate. If (*num* < *maxnum*), the value is delivered to *val[num]* and *num* is incremented. Otherwise an error is indicated. This allows multiple instances of a flag, each with possibly distinct values, to be remembered. *Note:* the *num* and *maxnum* members of the structure must be set properly *before* the call to **getflags()**.

An error is indicated if no "value" is present when one is expected, and vice versa. An error is also indicated if a command-line string starts with '-' but is not the name of a flag as specified in the format string.

Note: The above specifications impose a constraint on how flag-descriptors are ordered in the format string. Any flag whose name is a prefix of the name of another flag must appear in the format string after the longer one. This also implies that unnamed flags such as "-#" or "-*" must be given last.

RETURNS

In all cases, **getflags()** updates **pac* and **pav* to reflect the number of command-line arguments successfully consumed. If all command-line flags are valid, **getflags()** returns NULL. Otherwise, if *usagetail* is NULL, **getflags()** returns a pointer to the offending command-line flag. Otherwise, **getflags()** emits a "usage" message and aborts the calling process by making the call:

```
exit(EXIT_FAILURE);
```

The usage message is of the form:

usage: *process-name* [*flags*] *usagetail*

where *flags* is a list of the valid flags. Conventionally, *usagetail* is a string describing any non-flag arguments the program accepts, eg: "text-files".

EXAMPLE

All the above may seem terribly complicated, but the following example shows how simple it is to use **getflags()**...

To accept the command-line:

```
myprog -f filename -s -check -b512 file1 file2 file3
```

one might write:

```
BOOL silent = NO;
BOOL checking = NO;
CHAR *fname = "default";
LONG biasval = 256L;

INT do_file(char *, char *, long);

INT main(INT ac, CHAR **av)
        {
        INT ok;

        getflags(&ac, &av, "b##,check,f*,s", "object-files",
                &biasval, &checking, &fname, &silent);

        for ( ; ac && *av; --ac, ++av)
                ok += do_file(*av, fname, biasval);

        exit ((checking && !ok) ? EXIT_FAILURE : EXIT_SUCCESS);
        }
```

In this case, an unrecognised command-line flag would provoke the following fatal error message:

```
usage: myprog [-b## -check -f* -s] object-files
```

SEE ALSO

error(3), **remark(3)**, **warning(3)**, **mjsu(7)**.

AVAILABILITY

getflags() is written in C, conforming to ANSI X3.159-1989 (hosted program environment).