NAME

configfile, config_read, config_delete, config_renewed, config_length, config_issub, config_isatom, config_isstring − generic configuration file functions

SYNOPSIS

#include <configfile.h>

config_t *config_read(const char ***file*****, int** ***flags*****, config_t** ***cfg*****)**
void config_delete(config_t ***cfg*****)**
int config_renewed(config_t ***cfg*****)**
size_t config_length(config_t ***cfg*****)**
int config_issub(config_t ***cfg*****)**
int config_isatom(config_t ***cfg*****)**
int config_isstring(config_t ***cfg*****)**

DESCRIPTION

The **configfile** routines operate on a generic configuration file that follows the syntax described in **configfile**(5).

The interface presented by the functions above uses the following type and definitions from <configfile.h>:

```
typedef const struct config {
    config_t      *next;      /* Next configuration file thing. */
    config_t      *list;      /* For a { sublist }. */
    const char    *file;      /* File and line where this is found. */
    unsigned      line;
    int           flags;      /* Special flags. */
    char          word[];     /* Payload. */
} config_t;

#define  CFG_CLONG    0x0001   /* strtol(word, &end, 0) is valid. */
#define  CFG_OLONG    0x0002   /* strtol(word, &end, 010). */
#define  CFG_DLONG    0x0004   /* strtol(word, &end, 10). */
#define  CFG_XLONG    0x0008   /* strtol(word, &end, 0x10). */
#define  CFG_CULONG   0x0010   /* strtoul(word, &end, 0). */
#define  CFG_OULONG   0x0020   /* strtoul(word, &end, 010). */
#define  CFG_DULONG   0x0040   /* strtoul(word, &end, 10). */
#define  CFG_XULONG   0x0080   /* strtoul(word, &end, 0x10). */
#define  CFG_STRING   0x0100   /* The word is enclosed in quotes. */
#define  CFG_SUBLIST  0x0200   /* This is a sublist, so no word. */
#define  CFG_ESCAPED  0x0400   /* Escapes are still marked with \. */
```

In memory a configuration file is represented as a list of **config_t** cells linked together with the **next** field ending with a null pointer. A sublist between braces is attached to a cell at the **list** field. Words and strings are put in the **word** field, a null terminated string. The **flags** field records the type and features of a cell. The **CFG_*LONG** flags are set if a word is a number according to one of the **strtol** or **strtoul** calls. Purely a number, no quotes or trailing garbage. The **CFG_STRING** flag is set if the object was enclosed in double quotes. Lastly **CFG_SUBLIST** tells if the cell is only a pointer to a sublist in braces.

Characters in a word or string may have been formed with the \ escape character. They have been parsed and expanded, but the \ is still present if **CFG_ESCAPED** is set. The **word** array may be changed, as long as it doesn't grow longer, so one may remove the \s like this:

```
if (cfg->flags & CFG_ESCAPED) {
    char *p, *q;
    p= q= cfg->word;
    for (;;) {
        if ((*q = *p) == '\\') *q = *++p;
```

```
                if (*q == 0) break;
                p++;
                q++;
            }
        }
```

The low level syntax of a config file is checked when it is read. If an error is encountered a message is printed and the program exits with exit code 1. What the data means is not checked, that should be done by the program using the data. Only the atom **include** at the beginning of a list is special. It should be followed by a string. The string is seen as the name of a file, that is opened, read, and inserted in place of the **include**. Unless the name of the file starts with a **/**, it is sought relative to the directory the current file is found in. Nonexistent files are treated as being empty.

The **file** and **line** fields in each cell tell where the cell was read.

Functions

A configuration file is read with **config_read**. The first argument is the file to read. The second is either **0** or **CFG_ESCAPED** to tell whether \ escapes should be fully expanded without leaving a trace, or if they should still be marked with a \ so that the caller knows where the excapes are. The third argument, *cfg*, should be a null pointer on the first call. If you want to reread a config file that may have changed then *cfg* should be what you previously read.

With **config_delete** one can free up the memory that has been acquired with **malloc**(3) to hold the contents of the configuration file.

To determine if the contents of configuration file has changed when reread one uses **config_renewed** after **config_read**. It returns a "changed" flag that is set when the configuration file changed and then clears that flag. It returns true on the very first call. For the function to work you need to feed the old data back into **config_read**, not delete and reread.

The length of a series of config structures is told by **config_length**. It follows the **next** fields, so a sublist between braces counts as one extra.

The **config_issub**, **config_isatom** and **config_isstring** functions are just pretty macros to test if a cell references a sublist, is a word/string, or is just a string. **CFG_SUBLIST** and **CFG_STRING** tell the same story.

FILES

***/etc/*.conf** Several files in several **etc** directories.

SEE ALSO

configfile(5).

NOTES

The syntax of a config file puts some constraints on what you find in memory. The top level list consists entirely of sublist cells. These point to lists that start with at least an atom, followed by a mix of atoms and sublist cells. These sublists in turn point to a list of only sublist cells (recurse now.)

The struct config shown above is not exactly proper C to aid readability, read <configfile.h> itself to see why.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)