The Configuration File Library

0.2.0

Generated by Doxygen 1.5.8

Mon Jan 24 01:13:09 2011

Contents

Chapter 1

C Environment Library: Configuration File Library

Version:

0.2.0

Author:

Jun Woong (woong.jun at gmail.com)

Date:

last modified on 2011-01-24

1.1 Introduction

This document specifies the Configuration File Library which belongs to the C Environment Library. This library reads an "ini-style" configuration file and allows its user to readily access to values set by the file. There is no de jure standard for "ini" files, but this library supports most of what the Wikipedia page for the "ini" file (`http://en.wikipedia.org/wiki/INI_file`) describes; sections (with no support for nested ones), line concatenation by a backslash, escape sequences and so on. Differently from other implementataions, this library supports a simple type system. This aids its users to retrieve values set by a configuration file without manual conversion to a desired type.

The Configuration File Library reserves identifiers starting with `conf_` and `CONF_`, and imports the Assertion Library (which requires the Exception Handling Library), the Memory Management Library, the Hash Library and the Table Library.

1.1.1 Concepts

There are several concepts used to specify the Configuration File Library.

"Configuration varaibles" (called simply "variables" hereafter) are variables managed by the library and set by a default setting, a configuration file or a program. Variables have names and types.

A set of variable can be grouped and defined to belong to a "configuration section" (called simply "section" hereafter). A section comprises a distinct namespace, thus two variables with the same name designate two different variables if they belong to different sections.

The "global section" is an unnamed section that always exists. How to designate the global section and when variables belong to it are described below. The "current section" is a section set by a user program so that variables with no section designation are assumed to belong to it; conf_section() is used to set a section to the current one.

The supported "types" are boolean, signed/unsigned integer, real and string. The string type is most general and the library provides facilities to convert that type to others.

A "congifuration description table" is an array that has a sequence of variable names and their properties including a default value. The table also specifies a set of supported sections and variables. If supplied, the library recognizes sections and variables only appeared in the table. Otherwise, all sections and variables mentioned in a configuration file are recognized.

1.2 How to Use The Library

The Configuration File Library reads an "ini-style" configuration file and set variables according to its contents. The library behaves differently depending on whether a configuration description table is given by conf_preset(). The table can be composed by creating an array of `conf_t` and passed to the library through conf_preset(). If conf_preset() is not invoked, conf_init() has to be used to initiate the library and to read a specified configuration file. If conf_init() is called after conf_preset(), the contents read from a configuration file override what the table sets.

A configuration description table gives a list of sections and variables that the library can recognize and any other section/variable names that appear in a configuration file (but not in the table) are treated as an error.

If no configuration description table given, the library recognizes all possible sections and variables during conf_init() reads a configuration file and since there is no way to prescribe the type of each variable, all variables are assumed to have the string type. Note that unless a configuration file is protected from a malicious user, the user can exploit it to interfere the normal starting of a program; lots of sections and variables require the library to consume lots of memory blocks and thus other parts of a program might fail to perform its job due to lack of memory. It is essential, thus, to restrict allowable section and variable names by letting conf_preset() provide a predefined set of names if a configuration file can be exposed to such a user.

The storage used to maintain a program configuration itself is managed by the library.

1.2.1 Configuration Description Tables

If ever invoked, conf_preset() has to be invoked before conf_init(). If a program need not read a configuration file and uses only predefined settings given through conf_preset(), it need not call conf_init() at all. A configuration description table is an array of `conf_t` and enumerates section/variable names, their types and default values. For more details including how to designate a section and variable in the table and what each field of the table means, see `conf_t`.

1.2.2 Configuration Files

A configuration file basically has the form of a so-called "ini" file. The file is consisted of variable-value pairs belonged to a certain section as follows:

```
[section_1]
var1 = value1
var2 = value2
```

A string between the square brackets specifies a section and variable-value pairs appear below are belonged to that section. Names for sections and variables have to be consisted of only digits, alphabets and an underscore (_). By default, names are case-insensitive (setting the `CONF_OPT_CASE` bit in the second argument to conf_preset() and conf_init() changes this behavior). They cannot have an embedded space. Digits, alphabets are here dependent on the locale where the library is used. If a program using the library changes its locale to other than "C" locale, characters that are allowed for section/variable names also change. Even if multibyte characters can appear in values, section and variable names cannot have them.

If the pairs are given before any section has not been specified, they belong to the "global" section. The global section also can be specified by an empty section name as shown below:

```
[]    # global section
var1 = value1
```

A section does not nest and variables belonging to a section need not be gethered.

```
var0 = value0    # belongs to the global section

[section_1]
var1 = value1    # belongs to section_1

[section_2]
var2 = value2    # belongs to section_2

[section_1]
var3 = value3    # belongs to section_1, now section_1 has two variables
```

Two different sections have the same variable and they are distinct variables.

```
[section_1]
```

```
var1 = value1    # var1 belonging to section_1

[section_2]
var1 = value1    # var1 belonging to section_2
```

If a variable appears with the same name as one that appeared first under the same section, the value is overwritten by the latter variable setting.

```
[section_1]
var1 = value1    # var1 has value1
var1 = value2    # var2 now has value2
```

Comments begin with a semicolon (;) or a number sign (#) and ends with a newline as you have shown in examples above.

If the last character of a line is a backslash (\) without any trailing spaces, its following line is spliced to the line by eliminating the backslash and following newline. Any whitespaces preceding the backslash and any leading whitespaces in the following line are replaced with a space.

```
[section_1]
var1 = val\
ue             # var1 = value
var2 = value\
           2      # var2 = value 2
var3 = value    \
           3         # var3 = value 3
```

Values following an equal sign (=) after variables can have two forms, quoted and unquoted. Quoted values have to start with either a double-quote (") or single-quote (') and end with the matching character; that is, the whole value should be quoted. A semicolon (;) or number sign (#) in a quoted value does not start a comment.

```
[section_1]
var1 = "quoted value. ; or # starts no comment"  # now this is comment
var2 = 'quoted value again'
var3 = this is not a "quoted" value
```

The default behavior of the library recognizes no escape sequences, but if the `CONF_OPT_ESC` bit is set in the second argument to conf_preset() and conf_init(), they are recognized in a quoted value; an unquoted value supports no escape sequences. The supported sequences are:

```
\'    \"    \?    \\    \0
\a    \b    \f    \n    \r    \t    \v
```

with the same meanings as defined in C, and also include:

```
\;    \#    \=
```

that are replaced with a semicolon, number sign and equal sign respectively.

Any leading and trailing whitespaces are omitted from an unquoted value; thus only way to keep those spaces is to quote the value. Other whitespaces are kept unchanged.

1.3 Boilerplate Code

As already explained, using the library starts with invoking conf_preset() or conf_init(). If you desire to provide a predefined set of sections and variables with default values, call conf_preset() before calling conf_init() that reads a configuration file. It is decided when calling conf_preset() or conf_init() (if conf_preset() has not been invoked) whether names are case-sensitive and escape sequences are recognized in quoted values.

After reading a configuration file using conf_init(), a user program can freely inspects variables using conf_get(), conf_getbool(), conf_getint(), conf_getuint(), conf_getreal() and conf_getstr(). conf_get() retrieves the value of a given variable and interprets it as having the declared type of the variable. Other functions are useful when a variable from which a value is to be retrieved has the string type and a user code knows how to interpret it; when a configuration description table is not used, all variables are assumed to have the string type. If variables belonging to a specific section are frequently referred to, conf_section() that changes the current section to a given section helps.

If a function returns an error indicator, an immediate call to conf_errcode() returns the information about the error and conf_errstr() gives a string describing a given error code that is useful when constructing error or log messages for users.

This library works on top of the Memory Management Library and if any function that performs memory allocation fails to get necessary memory, an exception is raised.

conf.c contains an example designed to use as many facilities of the library as possible in a disabled part and a bolierplate code is given here:

```
#define CONFFILE "test.conf"

...

conf_t tab[] = {
    "VarBool",          CONF_TYPE_BOOL, "yes",
    "VarInt",           CONF_TYPE_INT,  "255",
    "VarUint",          CONF_TYPE_UINT, "0xFFFF",
    "VarReal",          CONF_TYPE_REAL, "3.14159",
    "VarStr",           CONF_TYPE_STR,  "Global VarStr Default",
    "Section1.VarBool", CONF_TYPE_BOOL, "FALSE",
    "Section1.VarStr",  CONF_TYPE_STR,  "Section1.VarStr Default",
    "Section2.VarBool", CONF_TYPE_BOOL, "true",
    "Section2.VarReal", CONF_TYPE_REAL, "314159e-5",
    NULL,
};
size_t line;
FILE *fp;

if (conf_preset(tab, CONF_OPT_CASE | CONF_OPT_ESC) != CONF_ERR_OK) {
    fprintf(stderr, "test: %s\n", conf_errstr(conf_errcode()));
    conf_free();
    conf_hashreset();
    exit(EXIT_FAILURE);
}

fp = fopen(CONFFILE, "r");
if (!fp) {
        fprintf(stderr, "test: failed to open %s for reading\n", CONFFILE);
        conf_free();
        conf_hashreset();
        exit(EXIT_FAILURE);
    }

    line = conf_init(fp, 0);
    fclose(fp);

    if (line != 0) {
        fprintf(stderr, "test:%s:%ld: %s\n", CONFFILE, (unsigned long)line,
                conf_errstr(conf_errcode()));
        conf_free();
        conf_hashreset();
        exit(EXIT_FAILURE);
    }

    ... sets an internal data structure properly
        according to what are read from configuration variables ...

    conf_free();
    conf_hashreset();

    ...
```

Even if this code defines the name of a configuration file as a macro, you may hard-code it or make it determined from a program argument.

An array of the `conf_t` type, `tab` is a configuration description table. It defines five variables in the global scope, each of which has the boolean, integer, unsigned integer, real and string type, respectively. It defines two more sections named "Section1" and "Section2", and four variables that belong to them. The last value in each row is a default value for each variable being defined. A null pointer terminates defining the table.

conf_preset() delivers the table to the library. If a problem occurs, conf_preset() returns an error code (that is not `CONF_ERR_OK`), and you can inspect it further using conf_errcode() and conf_errstr(). Do not forget that this library is based on data structures using the Memory Management Library that raises an exception if memory allocation fails.

conf_init() takes a stream (a `FILE` pointer), not a file name. This is because taking a stream allows its user to hand to conf_init() various kinds of files or file-like objects, for example, a string connected to a stream which has no file name.

Once conf_init() has done its job, the stream for the configuration file is no longer necessary, so fclose() closes it.

conf_init() returns 0 if nothing is wrong, or the line number (that is greater than 0) on which a problem occurs otherwise. You can use the return value when issueing an error message.

Note that if the hash table supported by the Hash Library is used for other purposes, it may not be desired to call conf_hashreset(). See conf_free() and conf_hashreset() for more details. If you feel uncomfortable with several instances of calls to conf_free() and conf_hashreset(), you can introduce a label before clean-up code and jump to that label whenever cleaning-up required.

1.4 Future Directions

1.4.1 Recoverable Errors

The current implementation does not provide a way to recover from errors like encountering unrecognized sections or variables. Recovering from them is sometimes necessary; for example, a programmer might want to issue a diagnostic message when a user uses an old version of the configuration file format, or to construct a certain part of the configuration file format dynamically depending on other parts of it.

1.4.2 Minor Changes

table_new() used by the Configuration File Library to create tables for storing configuration data takes a hint for the expected size of the table to create. Even if the performance is not a big issue in this library and granting a good hint improves the performance of operations on tables, providing a reasonable one to table_new() is necessary.

1.5 Contact Me

Visit `http://project.woong.org` to get the lastest version of this library. Only a small portion of my homepage (`http://www.woong.org`) is maintained in English, thus one who is not good at Korean would have difficulty when navigating most of other pages served in Korean. If you think the information you are looking for is on pages written in Korean you cannot read, do not hesitate to send me an email asking for help.

Any comments about the library are welcomed. If you have a proposal or question on the library just email me, and then I will reply as soon as possible.

1.6 Copyright

This package is a configuration file reader implementation by Jun Woong. The implementation was written so as to conform with the Standard C published by ISO 9899:1990 and ISO 9899:1999.

Chapter 2

Todo List

Global conf_init Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

Global conf_preset Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary;
- considering changes to the format of a configuration file as a program to accept it is upgraded, making it a recoverable error to encounter a non-preset section or variable name would be useful; this enables an old version of the program to accept a new configuration file with diagnostics.

Global conf_set Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 conf_t Struct Reference

represents an element of a configuration description table.

`#include <conf.h>`

Data Fields

- char ∗ var
- int type
- char ∗ defval

5.1.1 Detailed Description

represents an element of a configuration description table.

`conf_t` represents an element of a configuration description table that is used for a user program to specify a set of sections and variable including their types and default values; a configuration table is the only way to specify types of variables as having other than `CONF_TYPE_STR` (string type).

The `var` member specifies a section/variable name. The string has one of the following two forms:

```
variable
section . variable
```

where whitespaces are allowed before and/or after a section and variable name. The first form refers to a variable in the global section; there is no concept of the "current" section yet because conf_section() cannot be invoked before conf_preset() or conf_init(). To mark the end of a table, set the `var` member to a null pointer.

The `type` member specifies the type of a variable and should be one of `CONF_TYPE_BOOL` (boolean value, int), `CONF_TYPE_INT` (signed integer, long), `CONF_TYPE_UINT` (unsigned integer, unsigned long), `CONF_TYPE_REAL` (floating-point number, double), and `CONF_TYPE_STR` (string, char ∗). Once a variable is set to have a type, there is no way to change its type; thus, if a variable is supposed to have various types depending on the context, set to `CONF_TYPE_STR` and use conf_conv(). For `OPT_TYPE_INT` and `OPT_TYPE_UINT`, the conversion of a given value recognizes the C-style prefixes; numbers starting with 0 are treated as octal, and those with 0x or 0X are treated as hexadecimal.

The `defval` member specifies a default value for a variable that is used when a configuration file dose not set that variable to a value. It cannot be a null pointer but an empty string. Note that conf_preset() that accepts a configuration description table does not check if a default value has a proper form for the type of a variable.

See the commented-out example code given in the source file for more about a configuration description table.

5.1.2 Field Documentation

5.1.2.1 char∗ conf_t::defval

default value

5.1.2.2 int conf_t::type

type of variable

5.1.2.3 char∗ conf_t::var

section name and variable name

The documentation for this struct was generated from the following file:

- conf.h

Chapter 6

File Documentation

6.1 conf.c File Reference

Source for Configuration File Library (CEL).

```
#include <stddef.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include "cbl/assert.h"
#include "cbl/memory.h"
#include "cdsl/hash.h"
#include "cdsl/table.h"
#include "conf.h"
```

Include dependency graph for conf.c:

Data Structures

- struct **valnode_t**

Defines

- #define **BUFLEN** 80
- #define **VALID_CHR**(c) (isalpha(c) || isdigit(c) || (c) == '_')

Functions

- int() conf_preset (const conf_t ∗tab, int ctrl)

  constructs a default set for configuration variables.
- size_t() conf_init (FILE ∗fp, int ctrl)

  reads a configuration file and constructs the configuration data.
- const void ∗ conf_conv (const char ∗val, int type)

  converts a string based on a type.
- const void ∗() conf_get (const char ∗var)

  retrieves a value with a section/variable name.
- int() conf_getbool (const char ∗var, int errval)

  retrieves a boolean value with a section/variable name.
- long() conf_getint (const char ∗var, long errval)

  retrieves an integral value with a section/variable name.
- unsigned long() conf_getuint (const char ∗var, unsigned long errval)

  retrieves an unsigned integral value with a section/variable name.
- double() conf_getreal (const char ∗var, double errval)

  retrieves a real value with a section/variable name.
- const char ∗() conf_getstr (const char ∗var)

  retrieves a string with a section/variable name.
- int() conf_set (const char ∗secvar, const char ∗value)

  inserts or replaces a value associated with a variable.
- int() conf_section (const char ∗sec)

  sets the current section.
- void() conf_free (void)

  deallocates the stroage for the configuration data.
- void() conf_hashreset (void)

  resets the hash table using hash_reset().

- int() conf_errcode (void)

  returns an error code.

- const char ∗() conf_errstr (int code)

  returns an error message.

6.1.1 Detailed Description

Source for Configuration File Library (CEL).

6.1.2 Function Documentation

6.1.2.1 const void∗ conf_conv (const char ∗ *val*, int *type*)

converts a string based on a type.

conf_conv() converts a string to an integer or floating-point number as requested. `type` should be `CONF_TYPE_BOOL` (which recognizes some forms of boolean values), `CONF_TYPE_INT` (which indicates conversion to signed long int), `CONT_TYPE_UINT` (conversion to unsigned long int), `CONF_TYPE_REAL` (conversion to double) or `CONF_TYPE_STR` (no conversion necessary). The expected forms for `CONF_TYPE_INT`, `CONF_TYPE_UINT` and `CONF_TYPE_REAL` are respectively those for strtol(), strtoul() and strtod(). `CONF_TYPE_BOOL` gives 1 for a string starting with 't', 'T', 'y', 'Y', '1' and 0 for others. conf_conv() returns a pointer to the storage that contains the converted value (an integer, floating-point number or string) and its caller (user code) has to convert the pointer properly (to const int ∗, const long ∗, const unsigned long ∗, const double ∗ and const char ∗) before use. If the conversion fails, conf_conv() returns a null pointer and sets `CONF_ERR_TYPE` as an error code.

Warning:

> A subsequent call to conf_getbool(), conf_getint(), conf_getuint() and conf_getreal() may overwrite the contents of the buffer pointed by the resulting pointer. Similarly, a subsequent call to conf_conv() and conf_get() may overwrite the contents of the buffer pointed by the resulting pointer unless the type is `CONF_TYPE_STR`.

Parameters:

← ***val*** string to convert

← ***type*** type based on which conversion performed

Returns:

pointer to storage that contains result or null pointer

Return values:

non-null pointer to conversion result

NULL conversion failure

Here is the caller graph for this function:

6.1.2.2 int() conf_errcode (void)

returns an error code.

Every function in this library sets the internal error variable as it performs its operation. Unlike errno provided by <errno.h>, the error variable of this library is set to `CONF_ERR_OK` before starting an operation, thus a user code need not to clear it before calling a conf_ function.

When using a function returning an error code (of the int type), the returned value is the same as what conf_errcode() will return if there is no intervening call to a conf_ function between them. When using a function returning a pointer, the only way to get what the error has been occurred is to use conf_errcode().

The following code fragment shows an example for how to use conf_errcode() and conf_errstr():

```
fp = fopen(conf, "r");
if (!fp)
    fprintf(stderr, "%s:%s: %s\n", prg, conf, conf_errstr(CONF_ERR_FILE));
line = conf_init(fp, CONF_OPT_CASE | CONF_OPT_ESC);
if (line != 0)
    fprintf(stderr, "%s:%s:%lu: %s\n", prg, conf, line, conf_errstr(conf_errcode()));
```

Possible exceptions: none

Unchecked errors: none

Returns:

current error code

6.1.2.3 const char∗() conf_errstr (int *code*)

returns an error message.

conf_errstr() returns an error message for a given error code.

Possible exceptions: assert_exceptfail

Unchecked errors: none

Parameters:

← ***code*** error code for which error message returned

Returns:

error message

6.1.2.4 void() conf_free (void)

deallocates the stroage for the configuration data.

conf_free() deallocates storages for the configuration data. After conf_free() invoked, other conf_ functions should not be called without an intervening call to conf_preset() or conf_init().

Possible exceptions: assert_exceptfail

Unchecked errors: none

Warning:

conf_free() does not reset the hash table used internally since it may be used by other parts of the program. Invoking hash_reset() through conf_hashreset() before program termination cleans up storages occupied by the table.

Returns:

nothing

6.1.2.5 const void∗() conf_get (const char ∗ *var*)

retrieves a value with a section/variable name.

conf_get() retrieves a value with a section/variable name.

In a program (e.g., when using conf_get()), variables can be referred to using one of the following forms:

```
variable
. variable
section . variable
```

where whitespaces are optional before and after section and variable names. The first form refers to a variable belonging to the "current" section; the current section can be set by invoking conf_section(). The second form refers to a variable belonging to the global section. The last form refers to a variable belonging to a specific section.

Warning:

A subsequent call to conf_conv() and conf_get() may overwrite the contents of the buffer pointed by the resulting pointer unless the type is `CONF_TYPE_STR`. Similarly, a subsequent call to conf_getbool(), conf_getint(), conf_getuint() and conf_getreal() may overwrite the contents of the buffer pointed by the resulting pointer.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

Returns:

pointer to storage that contains value or null pointer

Return values:

non-null value retrieved

NULL failure

Here is the call graph for this function:

6.1.2.6 int() conf_getbool (const char * *var*, int *errval*)

retrieves a boolean value with a section/variable name.

conf_getbool() retrieves a boolean value with a section/variable name. Every value for a variable is stored in a string form, and conf_getbool() converts it to a boolean value; the result is 1 (indicating true) if the string starts with 't', 'T', 'y', 'Y' or '1' ignoring any leading spaces and 0 (indicating false) otherwise. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_BOOL`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name
← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.1.2.7 long() conf_getint (const char * *var*, long *errval*)

retrieves an integral value with a section/variable name.

conf_getint() retrieves an integral value with a section/variable name. Every value for a variable is stored in a string form, and conf_getint() converts it to an integer using strtol() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_INT`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name
← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.1.2.8 double() conf_getreal (const char * *var*, double *errval*)

retrieves a real value with a section/variable name.

conf_getreal() retrieves a real value with a section/variable name. Every value for a variable is stored in a string form, and conf_getreal() converts it to a floating-point number using strtod() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_REAL`, the value of `errval` is returned; `HUGE_VAL` defined <math.h> would be a nice choice for `errval`.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name
← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.1.2.9 const char∗() conf_getstr (const char ∗ *var*)

retrieves a string with a section/variable name.

conf_getstr() retrieves a string with a section/variable name. Every value for a variable is stored in a string form, thus conf_getstr() performs no conversion. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_STR`, a null pointer is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

Returns:

string or null pointer

Return values:

non-null string retrieved
NULL failure

Here is the call graph for this function:

6.1.2.10 unsigned long() conf_getuint (const char * *var*, unsigned long *errval*)

retrieves an unsigned integral value with a section/variable name.

conf_getuint() retrieves an unsigned integral value with a section/variable name. Every value for a variable is stored in a string form, and conf_getuint() converts it to an unsigned integer using strtoul() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_UINT`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name
← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.1.2.11 void() conf_hashreset (void)

resets the hash table using hash_reset().

conf_hashreset() simply calls hash_reset() to reset the hash table. As explained in conf_free(), conf_free() does not invoke hash_reset() because the single hash table may be used by other parts of a user program. Since requiring a reference to hash_reset() when using the Configuration File Library is inconsistent and inconvenient (e.g., a user code is obliged to include "hash.h"), conf_hashreset() is provided as a wrapper for hash_reset().

Warning:

Do not forget that the effect on the hash table caused by conf_hashreset() is not limited to eliminating only what conf_ functions adds to the table; it completely cleans up the entire hash table.

Possible exceptions: none

Unchecked errors: none

Returns:

nothing

6.1.2.12 size_t() conf_init (FILE ∗ *fp*, int *ctrl*)

reads a configuration file and constructs the configuration data.

conf_init() reads a configuration file and constructs the configuration data by analyzing the file. For how conf_init() interacts with conf_preset(), see conf_preset().

The default behavior of the library is that names are not case-insensitive and that escape sequences are not recognized. This behavior can be changed by setting the `CONF_OPT_CASE` and `CONF_OPT_ESC` bits in `ctrl`, respectively; see also conf_preset().

If the control mode that can be set through `ctrl` has been already set by conf_preset(), conf_init() ignores `ctrl`.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: invalid file pointer given for `fp`

Parameters:

← ***fp*** file pointer from which configuration data read

← ***ctrl*** control code

Returns:

success/failure indicator

Return values:

0 success

positive line number on which error occurred

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

6.1.2.13 int() conf_preset (const conf_t ∗ *tab*, int *ctrl*)

constructs a default set for configuration variables.

A user program can specify the default set of configuration variables (including sections to which they belong and their types) with conf_preset(). The table (an array, in fact) containing necessary information have the `conf_t` type and called a "configuration description table." For a detailed explanation and examples, see `conf_t`. conf_preset(), if invoked, has to be called before conf_init(). conf_init() is not necessarily invoked if conf_preset() is used.

If invoked, conf_preset() remembers names that need to be recognized as sections and variables, types of variables, and their default values. When conf_init() processes a configuration file, a sections or variable that is not given via conf_preset() is considered an error. Using conf_preset() and a configuration description table is the only way to let variables have other types than `CONF_TYPE_STR` (string type).

If not invoked, conf_init() accepts any section and variable name (if they have a valid form) and all of variables are assumed to be of `CONF_TYPE_STR` type.

conf_preset() also takes `ctrl` for controling some behaviors of the library, especially handling section/variable names and values. If the `CONF_OPT_CASE` bit is set in `ctrl` (that is, `CONF_OPT_CASE` & `ctrl` is not 0), section and variable names are case-sensitive. If the `CONF_OPT_ESC` bit is set in `ctrl`, some forms of escape sequences are supported in a quoted value. The default behavior is that section and variable names are case-insensitive and no escape sequences are supported.

Warning:

conf_preset() does not warn that a default value for a variable does not have an expected form for the variable's type. It is to be treated as an error when retrieving the value by conf_get() or similar functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***tab*** configuration description table

← ***ctrl*** control code

Returns:

success/failure indicator

Return values:

CONF_ERR_OK success

others failure

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary;
- considering changes to the format of a configuration file as a program to accept it is upgraded, making it a recoverable error to encounter a non-preset section or variable name would be useful; this enables an old version of the program to accept a new configuration file with diagnostics.

6.1.2.14 int() conf_section (const char ∗ *sec*)

sets the current section.

conf_section() sets the current section to a given section. The global section can be set as the current section by giving an empty string "" to conf_section(). conf_section() affects how conf_get(), conf_getbool(), conf_getint(), confgetuint(), confgetreal(), confgetstr() and conf_set() work.

Parameters:

← ***sec*** section name to set as current section

Returns:

success/failure indicator

Return values:

CONF_ERR_OK success

others failure

6.1.2.15 int() conf_set (const char ∗ *secvar*, const char ∗ *value*)

inserts or replaces a value associated with a variable.

conf_set() inserts or replaces a value associated with a variable. If conf_preset() has been invoked, conf_set() is able to only replace a value associated with an existing variable, which means an error code is returned when a user tries to insert a new variable and its value (possibly with a new section). conf_set() is allowed to insert a new variable-value pair otherwise.

For how to refer to variables in a program, see conf_get().

Warning:

When conf_preset() invoked, conf_set() does not check if a given value is appropriate to the preset type of a variable. That mismatch is to be detected when conf_get() or similar functions called later for the variable.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***secvar*** section/variable name

← ***value*** value to store

Returns:

success/failure indicator

Return values:

CONF_ERR_OK success

others failure

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

6.2 conf.h File Reference

Header for Configuration File Library (CEL).

This graph shows which files directly or indirectly include this file:

Data Structures

- struct conf_t

  represents an element of a configuration description table.

Enumerations

- enum {

  CONF_TYPE_NO, CONF_TYPE_BOOL, CONF_TYPE_INT, CONF_TYPE_UINT,

  CONF_TYPE_REAL, CONF_TYPE_STR }

  defines enum constants for types of values.

- enum {

  CONF_ERR_OK, CONF_ERR_FILE, CONF_ERR_IO, CONF_ERR_SPACE,

  CONF_ERR_CHAR, CONF_ERR_LINE, CONF_ERR_BSLASH, CONF_ERR_SEC,

  CONF_ERR_VAR, CONF_ERR_TYPE, **CONF_ERR_MAX** }

  defines enum constants for error codes.

- enum { CONF_OPT_CASE = 0x01, CONF_OPT_ESC = CONF_OPT_CASE << 1 }

  defines masks for control options.

Functions

configuration initializing functions:

- int conf_preset (const conf_t ∗, int)

constructs a default set for configuration variables.

- size_t conf_init (FILE ∗, int)
  reads a configuration file and constructs the configuration data.
- void conf_free (void)
  deallocates the stroage for the configuration data.
- void conf_hashreset (void)
  resets the hash table using hash_reset().

configuration data-handling functions:

- const void ∗ conf_conv (const char ∗, int)
  converts a string based on a type.
- const void ∗ conf_get (const char ∗)
  retrieves a value with a section/variable name.
- int conf_getbool (const char ∗, int)
  retrieves a boolean value with a section/variable name.
- long conf_getint (const char ∗, long)
  retrieves an integral value with a section/variable name.
- unsigned long conf_getuint (const char ∗, unsigned long)
  retrieves an unsigned integral value with a section/variable name.
- double conf_getreal (const char ∗, double)
  retrieves a real value with a section/variable name.
- const char ∗ conf_getstr (const char ∗)
  retrieves a string with a section/variable name.
- int conf_set (const char ∗, const char ∗)
  inserts or replaces a value associated with a variable.
- int conf_section (const char ∗)
  sets the current section.

error handling functions:

- int conf_errcode (void)
  returns an error code.
- const char ∗ conf_errstr (int)
  returns an error message.

6.2.1 Detailed Description

Header for Configuration File Library (CEL).

Documentation for Configuration File Library (CEL).

6.2.2 Enumeration Type Documentation

6.2.2.1 anonymous enum

defines enum constants for types of values.

Enumerator:

CONF_TYPE_BOOL	has boolean (int) type
CONF_TYPE_INT	has integer (long) type
CONF_TYPE_UINT	has unsigned integer (unsigned long) type
CONF_TYPE_REAL	has floating-point (double) type
CONF_TYPE_STR	has string (char ∗) type

6.2.2.2 anonymous enum

defines enum constants for error codes.

Enumerator:

CONF_ERR_OK	everything is okay
CONF_ERR_FILE	file not found
CONF_ERR_IO	I/O error occurred
CONF_ERR_SPACE	space in section/variable name
CONF_ERR_CHAR	invalid character encountered
CONF_ERR_LINE	invalid line encountered
CONF_ERR_BSLASH	no following line for slicing
CONF_ERR_SEC	section not found
CONF_ERR_VAR	variable not found
CONF_ERR_TYPE	data type mismatch

6.2.2.3 anonymous enum

defines masks for control options.

Enumerator:

CONF_OPT_CASE	case-sensitive variable/section name
CONF_OPT_ESC	supports escape sequence in quoted value

6.2.3 Function Documentation

6.2.3.1 const void∗ conf_conv (const char ∗ *val*, int *type*)

converts a string based on a type.

conf_conv() converts a string to an integer or floating-point number as requested. `type` should be `CONF_TYPE_BOOL` (which recognizes some forms of boolean values), `CONF_TYPE_INT` (which indicates conversion to signed long int), `CONT_TYPE_UINT` (conversion to unsigned long int), `CONF_TYPE_REAL` (conversion to double) or `CONF_TYPE_STR` (no conversion necessary). The expected forms for `CONF_TYPE_INT`, `CONF_TYPE_UINT` and `CONF_TYPE_REAL` are respectively those for strtol(), strtoul() and strtod(). `CONF_TYPE_BOOL` gives 1 for a string starting with 't', 'T', 'y', 'Y', '1' and 0 for others. conf_conv() returns a pointer to the storage that contains the converted value (an integer, floating-point number or string) and its caller (user code) has to convert the pointer properly (to const int ∗, const long ∗, const unsigned long ∗, const double ∗ and const char ∗) before use. If the conversion fails, conf_conv() returns a null pointer and sets `CONF_ERR_TYPE` as an error code.

Warning:

A subsequent call to conf_getbool(), conf_getint(), conf_getuint() and conf_getreal() may overwrite the contents of the buffer pointed by the resulting pointer. Similarly, a subsequent call to conf_conv() and conf_get() may overwrite the contents of the buffer pointed by the resulting pointer unless the type is `CONF_TYPE_STR`.

Parameters:

← ***val*** string to convert

← ***type*** type based on which conversion performed

Returns:

pointer to storage that contains result or null pointer

Return values:

non-null pointer to conversion result

NULL conversion failure

Here is the caller graph for this function:

6.2.3.2 int conf_errcode (void)

returns an error code.

Every function in this library sets the internal error variable as it performs its operation. Unlike errno provided by <errno.h>, the error variable of this library is set to `CONF_ERR_OK` before starting an operation, thus a user code need not to clear it before calling a conf_ function.

When using a function returning an error code (of the int type), the returned value is the same as what conf_errcode() will return if there is no intervening call to a conf_ function between them. When using a function returning a pointer, the only way to get what the error has been occurred is to use conf_errcode().

The following code fragment shows an example for how to use conf_errcode() and conf_errstr():

```
fp = fopen(conf, "r");
if (!fp)
    fprintf(stderr, "%s:%s: %s\n", prg, conf, conf_errstr(CONF_ERR_FILE));
line = conf_init(fp, CONF_OPT_CASE | CONF_OPT_ESC);
if (line != 0)
    fprintf(stderr, "%s:%s:%lu: %s\n", prg, conf, line, conf_errstr(conf_errcode()));
```

Possible exceptions: none

Unchecked errors: none

Returns:

current error code

6.2.3.3 const char∗ conf_errstr (int *code*)

returns an error message.

conf_errstr() returns an error message for a given error code.

Possible exceptions: assert_exceptfail

Unchecked errors: none

Parameters:

← ***code*** error code for which error message returned

Returns:

error message

6.2.3.4 void conf_free (void)

deallocates the stroage for the configuration data.

conf_free() deallocates storages for the configuration data. After conf_free() invoked, other conf_ functions should not be called without an intervening call to conf_preset() or conf_init().

Possible exceptions: assert_exceptfail

Unchecked errors: none

Warning:

conf_free() does not reset the hash table used internally since it may be used by other parts of the program. Invoking hash_reset() through conf_hashreset() before program termination cleans up storages occupied by the table.

Returns:

nothing

6.2.3.5 const void∗ conf_get (const char ∗ *var*)

retrieves a value with a section/variable name.

conf_get() retrieves a value with a section/variable name.

In a program (e.g., when using conf_get()), variables can be referred to using one of the following forms:

```
variable
. variable
section . variable
```

where whitespaces are optional before and after section and variable names. The first form refers to a variable belonging to the "current" section; the current section can be set by invoking conf_section(). The second form refers to a variable belonging to the global section. The last form refers to a variable belonging to a specific section.

Warning:

> A subsequent call to conf_conv() and conf_get() may overwrite the contents of the buffer pointed by the resulting pointer unless the type is `CONF_TYPE_STR`. Similarly, a subsequent call to conf_getbool(), conf_getint(), conf_getuint() and conf_getreal() may overwrite the contents of the buffer pointed by the resulting pointer.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

Returns:

pointer to storage that contains value or null pointer

Return values:

non-null value retrieved

NULL failure

Here is the call graph for this function:

6.2.3.6 int conf_getbool (const char ∗ *var*, int *errval*)

retrieves a boolean value with a section/variable name.

conf_getbool() retrieves a boolean value with a section/variable name. Every value for a variable is stored in a string form, and conf_getbool() converts it to a boolean value; the result is 1 (indicating true) if the string starts with ’t’, ’T’, ’y’, ’Y’ or ’1’ ignoring any leading spaces and 0 (indicating false) otherwise. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_BOOL`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.2.3.7 long conf_getint (const char ∗ *var*, long *errval*)

retrieves an integral value with a section/variable name.

conf_getint() retrieves an integral value with a section/variable name. Every value for a variable is stored in a string form, and conf_getint() converts it to an integer using strtol() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_INT`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.2.3.8 double conf_getreal (const char ∗ *var*, double *errval*)

retrieves a real value with a section/variable name.

conf_getreal() retrieves a real value with a section/variable name. Every value for a variable is stored in a string form, and conf_getreal() converts it to a floating-point number using strtod() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_REAL`, the value of `errval` is returned; `HUGE_VAL` defined <math.h> would be a nice choice for `errval`.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.2.3.9 const char∗ conf_getstr (const char ∗ *var*)

retrieves a string with a section/variable name.

conf_getstr() retrieves a string with a section/variable name. Every value for a variable is stored in a string form, thus conf_getstr() performs no conversion. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_STR`, a null pointer is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

Returns:

string or null pointer

Return values:

non-null string retrieved

NULL failure

Here is the call graph for this function:

6.2.3.10 unsigned long conf_getuint (const char $*$ *var*, unsigned long *errval*)

retrieves an unsigned integral value with a section/variable name.

conf_getuint() retrieves an unsigned integral value with a section/variable name. Every value for a variable is stored in a string form, and conf_getuint() converts it to an unsigned integer using strtoul() declared in <stdlib.h>. If there is no variable with the given name or the preset type of the variable is not `CONF_TYPE_UINT`, the value of `errval` is returned.

For how to refer to variables in a program, see conf_get().

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***var*** section/variable name

← ***errval*** value returned as error

Returns:

converted result or `errval`

Here is the call graph for this function:

6.2.3.11 void conf_hashreset (void)

resets the hash table using hash_reset().

conf_hashreset() simply calls hash_reset() to reset the hash table. As explained in conf_free(), conf_free() does not invoke hash_reset() because the single hash table may be used by other parts of a user program. Since requiring a reference to hash_reset() when using the Configuration File Library is inconsistent and inconvenient (e.g., a user code is obliged to include "hash.h"), conf_hashreset() is provided as a wrapper for hash_reset().

Warning:

Do not forget that the effect on the hash table caused by conf_hashreset() is not limited to eliminating only what conf_ functions adds to the table; it completely cleans up the entire hash table.

Possible exceptions: none

Unchecked errors: none

Returns:

nothing

6.2.3.12 size_t conf_init (FILE $*$ *fp*, int *ctrl*)

reads a configuration file and constructs the configuration data.

conf_init() reads a configuration file and constructs the configuration data by analyzing the file. For how conf_init() interacts with conf_preset(), see conf_preset().

The default behavior of the library is that names are not case-insensitive and that escape sequences are not recognized. This behavior can be changed by setting the `CONF_OPT_CASE` and `CONF_OPT_ESC` bits in `ctrl`, respectively; see also conf_preset().

If the control mode that can be set through `ctrl` has been already set by conf_preset(), conf_init() ignores `ctrl`.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: invalid file pointer given for `fp`

Parameters:

← ***fp*** file pointer from which configuration data read

← ***ctrl*** control code

Returns:

success/failure indicator

Return values:

0 success

positive line number on which error occurred

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

6.2.3.13 int conf_preset (const conf_t $*$ *tab*, int *ctrl*)

constructs a default set for configuration variables.

A user program can specify the default set of configuration variables (including sections to which they belong and their types) with conf_preset(). The table (an array, in fact) containing necessary information have the `conf_t` type and called a "configuration description table." For a detailed explanation and examples, see `conf_t`. conf_preset(), if invoked, has to be called before conf_init(). conf_init() is not necessarily invoked if conf_preset() is used.

If invoked, conf_preset() remembers names that need to be recognized as sections and variables, types of variables, and their default values. When conf_init() processes a configuration file, a sections or variable that is not given via conf_preset() is considered an error. Using conf_preset() and a configuration description table is the only way to let variables have other types than `CONF_TYPE_STR` (string type).

If not invoked, conf_init() accepts any section and variable name (if they have a valid form) and all of variables are assumed to be of `CONF_TYPE_STR` type.

conf_preset() also takes `ctrl` for controling some behaviors of the library, especially handling section/variable names and values. If the `CONF_OPT_CASE` bit is set in `ctrl` (that is, `CONF_OPT_CASE & ctrl` is not 0), section and variable names are case-sensitive. If the `CONF_OPT_ESC` bit is set in `ctrl`, some forms of escape sequences are supported in a quoted value. The default behavior is that section and variable names are case-insensitive and no escape sequences are supported.

Warning:

conf_preset() does not warn that a default value for a variable does not have an expected form for the variable's type. It is to be treated as an error when retrieving the value by conf_get() or similar functions.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***tab*** configuration description table
← ***ctrl*** control code

Returns:

success/failure indicator

Return values:

CONF_ERR_OK	success
others	failure

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary;
- considering changes to the format of a configuration file as a program to accept it is upgraded, making it a recoverable error to encounter a non-preset section or variable name would be useful; this enables an old version of the program to accept a new configuration file with diagnostics.

6.2.3.14 int conf_section (const char * *sec*)

sets the current section.

conf_section() sets the current section to a given section. The global section can be set as the current section by giving an empty string "" to conf_section(). conf_section() affects how conf_get(), conf_getbool(), conf_getint(), confgetuint(), confgetreal(), confgetstr() and conf_set() work.

Parameters:

← ***sec*** section name to set as current section

Returns:

success/failure indicator

Return values:

CONF_ERR_OK success

others failure

6.2.3.15 int conf_set (const char $*$ *secvar*, const char $*$ *value*)

inserts or replaces a value associated with a variable.

conf_set() inserts or replaces a value associated with a variable. If conf_preset() has been invoked, conf_set() is able to only replace a value associated with an existing variable, which means an error code is returned when a user tries to insert a new variable and its value (possibly with a new section). conf_set() is allowed to insert a new variable-value pair otherwise.

For how to refer to variables in a program, see conf_get().

Warning:

When conf_preset() invoked, conf_set() does not check if a given value is appropriate to the preset type of a variable. That mismatch is to be detected when conf_get() or similar functions called later for the variable.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters:

← ***secvar*** section/variable name

← ***value*** value to store

Returns:

success/failure indicator

Return values:

CONF_ERR_OK success

others failure

Todo

Improvements are possible and planned:

- some adjustment on arguments to table_new() is necessary.

Index