# libconfig

A Library For Processing Structured Configuration Files

Version 1.4.9

28 September 2012

**Mark A. Lindner**

# Table of Contents

# 1 Introduction

*Libconfig* is a library for reading, manipulating, and writing structured configuration files. The library features a fully reentrant parser and includes bindings for both the C and C++ programming languages.

The library runs on modern POSIX-compilant systems, such as Linux, Solaris, and Mac OS X (Darwin), as well as on Microsoft Windows 2000/XP and later (with either Microsoft Visual Studio 2005 or later, or the GNU toolchain via the MinGW environment).

## 1.1 Why Another Configuration File Library?

There are several open-source configuration file libraries available as of this writing. This library was written because each of those libraries falls short in one or more ways. The main features of *libconfig* that set it apart from the other libraries are:

- A fully reentrant parser. Independent configurations can be parsed in concurrent threads at the same time.
- Both C *and* C++ bindings, as well as hooks to allow for the creation of wrappers in other languages.
- A simple, structured configuration file format that is more readable and compact than XML and more flexible than the obsolete but prevalent Windows "INI" file format.
- A low-footprint implementation (just 37K for the C library and 76K for the C++ library) that is suitable for memory-constrained systems.
- Proper documentation.

## 1.2 Using the Library from a C Program

To use the library from C code, include the following preprocessor directive in your source files:

```
#include <libconfig.h>
```

To link with the library, specify '`-lconfig`' as an argument to the linker.

## 1.3 Using the Library from a C++ Program

To use the library from C++, include the following preprocessor directive in your source files:

```
#include <libconfig.h++>
```

Or, alternatively:

```
#include <libconfig.hh>
```

The C++ API classes are defined in the namespace '`libconfig`', hence the following statement may optionally be used:

```
using namespace libconfig;
```

To link with the library, specify '`-lconfig++`' as an argument to the linker.

## 1.4 Multithreading Issues

*Libconfig* is fully *reentrant*; the functions in the library do not make use of global variables and do not maintain state between successive calls. Therefore two independent configurations may be safely manipulated concurrently by two distinct threads.

*Libconfig* is not *thread-safe*. The library is not aware of the presence of threads and knows nothing about the host system's threading model. Therefore, if an instance of a configuration is to be accessed from multiple threads, it must be suitably protected by synchronization mechanisms like read-write locks or mutexes; the standard rules for safe multithreaded access to shared data must be observed.

*Libconfig* is not *async-safe*. Calls should not be made into the library from signal handlers, because some of the C library routines that it uses may not be async-safe.

*Libconfig* is not guaranteed to be *cancel-safe*. Since it is not aware of the host system's threading model, the library does not contain any thread cancellation points. In most cases this will not be an issue for multithreaded programs. However, be aware that some of the routines in the library (namely those that read/write configurations from/to files or streams) perform I/O using C library routines which may potentially block; whether or not these C library routines are cancel-safe depends on the host system.

## 1.5 Internationalization Issues

*Libconfig* does not natively support Unicode configuration files, but string values may contain Unicode text encoded in UTF-8; such strings will be treated as ordinary 8-bit ASCII text by the library. It is the responsibility of the calling program to perform the necessary conversions to/from wide (`wchar_t`) strings using the wide string conversion functions such as `mbsrtowcs()` and `wcsrtombs()` or the `iconv()` function of the *libiconv* library.

The textual representation of a floating point value varies by locale. However, the *libconfig* grammar specifies that floating point values are represented using a period ('.') as the radix symbol; this is consistent with the grammar of most programming languages. When a configuration is read in or written out, *libconfig* temporarily changes the `LC_NUMERIC` category of the locale of the calling thread to the "C" locale to ensure consistent handling of floating point values regardless of the locale(s) in use by the calling program.

Note that the MinGW environment does not (as of this writing) provide functions for changing the locale of the calling thread. Therefore, when using *libconfig* in that environment, the calling program is responsible for changing the `LC_NUMERIC` category of the locale to the `"C"` locale before reading or writing a configuration.

## 1.6 Compiling Using pkg-config

On UNIX systems you can use the *pkg-config* utility (version 0.20 or later) to automatically select the appropriate compiler and linker switches for *libconfig*. Ensure that the environment variable `PKG_CONFIG_PATH` contains the absolute path to the '`lib/pkgconfig`' subdirectory of the *libconfig* installation. Then, you can compile and link C programs with *libconfig* as follows:

```
gcc `pkg-config --cflags libconfig` myprogram.c -o myprogram \
    `pkg-config --libs libconfig`
```

And similarly, for C++ programs:

```
g++ `pkg-config --cflags libconfig++` myprogram.cpp -o myprogram \
    `pkg-config --libs libconfig++`
```

Note the backticks in the above examples.

When using **autoconf**, the `PKG_CHECK_MODULES` m4 macro may be used to check for the presence of a given version of *libconfig*, and set the appropriate Makefile variables automatically. For example:

```
PKG_CHECK_MODULES([LIBCONFIGXX], [libconfig++ >= 1.4],,
  AC_MSG_ERROR([libconfig++ 1.4 or newer not found.])
)
```

In the above example, if *libconfig*++ version 1.4 or newer is found, the Makefile variables `LIBCONFIGXX_LIBS` and `LIBCONFIGXX_CFLAGS` will be set to the appropriate compiler and linker flags for compiling with *libconfig*, and if it is not found, the configure script will abort with an error to that effect.

## 1.7 Version Test Macros

The '`libconfig.h`' header declares the following macros:

`LIBCONFIG_VER_MAJOR` [Macro]
`LIBCONFIG_VER_MINOR` [Macro]
`LIBCONFIG_VER_REVISION` [Macro]

These macros represent the major version, minor version, and revision of the *libconfig* library. For example, in *libconfig* 1.4 these are defined as '`1`', '`4`', and '`0`', respectively. These macros can be used in preprocessor directives to determine which *libconfig* features and/or APIs are present. For example:

```
#if (((LIBCONFIG_VER_MAJOR == 1) && (LIBCONFIG_VER_MINOR >= 4)) \
     || (LIBCONFIG_VER_MAJOR > 1))
  /* use features present in libconfig 1.4 and later */
#endif
```

These macros were introduced in *libconfig* 1.4.

Similarly, the '`libconfig.h++`' header declares the following macros:

`LIBCONFIGXX_VER_MAJOR` [Macro]
`LIBCONFIGXX_VER_MINOR` [Macro]
`LIBCONFIGXX_VER_REVISION` [Macro]

These macros represent the major version, minor version, and revision of the *libconfig*++ library.

# 2 Configuration Files

*Libconfig* supports structured, hierarchical configurations. These configurations can be read from and written to files and manipulated in memory.

A *configuration* consists of a group of *settings*, which associate names with values. A *value* can be one of the following:

- A *scalar value*: integer, 64-bit integer, floating-point number, boolean, or string
- An *array*, which is a sequence of scalar values, all of which must have the same type
- A *group*, which is a collection of settings
- A *list*, which is a sequence of values of any type, including other lists

Consider the following configuration file for a hypothetical GUI application, which illustrates all of the elements of the configuration file grammar.

```
# Example application configuration file

version = "1.0";

application:
{
  window:
  {
    title = "My Application";
    size = { w = 640; h = 480; };
    pos = { x = 350; y = 250; };
  };

  list = ( ( "abc", 123, true ), 1.234, ( /* an empty list */) );

  books = ( { title  = "Treasure Island";
              author = "Robert Louis Stevenson";
              price  = 29.95;
              qty    = 5; },
            { title  = "Snow Crash";
              author = "Neal Stephenson";
              price  = 9.99;
              qty    = 8; } );

  misc:
  {
    pi = 3.141592654;
    bigint = 9223372036854775807L;
    columns = [ "Last Name", "First Name", "MI" ];
    bitmask = 0x1FC3;
  };
};
```

Settings can be uniquely identified within the configuration by a *path*. The path is a dot-separated sequence of names, beginning at a top-level group and ending at the setting itself. Each name in the path is the name of a setting; if the setting has no name because it is an element in a list or array, an integer index in square brackets can be used as the name.

For example, in our hypothetical configuration file, the path to the `x` setting is `application.window.pos.x`; the path to the `version` setting is simply `version`; and the path to the `title` setting of the second book in the `books` list is `application.books.[1].title`.

The datatype of a value is determined from the format of the value itself. If the value is enclosed in double quotes, it is treated as a string. If it looks like an integer or floating point number, it is treated as such. If it is one of the values `TRUE`, `true`, `FALSE`, or `false` (or any other mixed-case version of those tokens, e.g., `True` or `FaLsE`), it is treated as a boolean. If it consists of a comma-separated list of values enclosed in square brackets, it is treated as an array. And if it consists of a comma-separated list of values enclosed in parentheses, it is treated as a list. Any value which does not meet any of these criteria is considered invalid and results in a parse error.

All names are case-sensitive. They may consist only of alphanumeric characters, dashes (‘`-`’), underscores (‘`_`’), and asterisks (‘`*`’), and must begin with a letter or asterisk. No other characters are allowed.

In C and C++, integer, 64-bit integer, floating point, and string values are mapped to the types `int`, `long long`, `double`, and `const char *`, respectively. The boolean type is mapped to `int` in C and `bool` in C++.

The following sections describe the elements of the configuration file grammar in additional detail.

## 2.1 Settings

A setting has the form:

*name* **=** *value* **;**

or:

*name* **:** *value* **;**

The trailing semicolon is optional. Whitespace is not significant.

The value may be a scalar value, an array, a group, or a list.

## 2.2 Groups

A group has the form:

**{** *settings ...* **}**

Groups can contain any number of settings, but each setting must have a unique name within the group.

## 2.3 Arrays

An array has the form:

**[** *value*, *value ...* **]**

An array may have zero or more elements, but the elements must all be scalar values of the same type.

## 2.4 Lists

A list has the form:

**(** *value***,** *value ...* **)**

A list may have zero or more elements, each of which can be a scalar value, an array, a group, or another list.

## 2.5 Integer Values

Integers can be represented in one of two ways: as a series of one or more decimal digits (‘`0`’ - ‘`9`’), with an optional leading sign character (‘`+`’ or ‘`-`’); or as a hexadecimal value consisting of the characters ‘`0x`’ followed by a series of one or more hexadecimal digits (‘`0`’ - ‘`9`’, ‘`A`’ - ‘`F`’, ‘`a`’ - ‘`f`’).

## 2.6 64-bit Integer Values

Long long (64-bit) integers are represented identically to integers, except that an ’L’ character is appended to indicate a 64-bit value. For example, ‘`0L`’ indicates a 64-bit integer value 0.

## 2.7 Floating Point Values

Floating point values consist of a series of one or more digits, one decimal point, an optional leading sign character (‘`+`’ or ‘`-`’), and an optional exponent. An exponent consists of the letter ‘`E`’ or ‘`e`’, an optional sign character, and a series of one or more digits.

## 2.8 Boolean Values

Boolean values may have one of the following values: ‘`true`’, ‘`false`’, or any mixed-case variation thereof.

## 2.9 String Values

String values consist of arbitrary text delimited by double quotes. Literal double quotes can be escaped by preceding them with a backslash: ‘`\"`’. The escape sequences ‘`\\`’, ‘`\f`’, ‘`\n`’, ‘`\r`’, and ‘`\t`’ are also recognized, and have the usual meaning.

In addition, the ‘`\x`’ escape sequence is supported; this sequence must be followed by *exactly two* hexadecimal digits, which represent an 8-bit ASCII value. For example, ‘`\xFF`’ represents the character with ASCII code 0xFF.

No other escape sequences are currently supported.

Adjacent strings are automatically concatenated, as in C/C++ source code. This is useful for formatting very long strings as sequences of shorter strings. For example, the following constructs are equivalent:

- `"The quick brown fox jumped over the lazy dog."`
- ```
  "The quick brown fox"
  " jumped over the lazy dog."
  ```
- ```
  "The quick" /* comment */ " brown fox " // another comment
  "jumped over the lazy dog."
  ```

## 2.10 Comments

Three types of comments are allowed within a configuration:

- Script-style comments. All text beginning with a '`#`' character to the end of the line is ignored.
- C-style comments. All text, including line breaks, between a starting '`/*`' sequence and an ending '`*/`' sequence is ignored.
- C++-style comments. All text beginning with a '`//`' sequence to the end of the line is ignored.

As expected, comment delimiters appearing within quoted strings are treated as literal text.

Comments are ignored when the configuration is read in, so they are not treated as part of the configuration. Therefore if the configuration is written back out to a stream, any comments that were present in the original configuration will be lost.

## 2.11 Include Directives

A configuration file may "include" the contents of another file using an *include directive*. This directive has the effect of inlining the contents of the named file at the point of inclusion.

An include directive must appear on its own line in the input. It has the form:

**@include** "*filename*"

Any backslashes or double quotes in the filename must be escaped as '`\\`' and '`\"`', respectively.

For example, consider the following two configuration files:

```
# file: quote.cfg
quote = "Criticism may not be agreeable, but it is necessary."
        " It fulfils the same function as pain in the human"
        " body. It calls attention to an unhealthy state of"
        " things.\n"
        "\t--Winston Churchill";
```

```
# file: test.cfg
info: {
  name = "Winston Churchill";
  @include "quote.cfg"
  country = "UK";
};
```

Include files may be nested to a maximum of 10 levels; exceeding this limit results in a parse error.

Like comments, include directives are not part of the configuration file syntax. They are processed before the configuration itself is parsed. Therefore, they are not preserved when the configuration is written back out to a stream. There is presently no support for programmatically inserting include directives into a configuration.

# 3 The C API

This chapter describes the C library API. The type *config_t* represents a configuration, and the type *config_setting_t* represents a configuration setting.

The boolean values `CONFIG_TRUE` and `CONFIG_FALSE` are macros defined as `(1)` and `(0)`, respectively.

`void config_init` (*config_t * **config***) [Function]
`void config_destroy` (*config_t * **config***) [Function]

These functions initialize and destroy the configuration object *config*.

`config_init()` initializes the *config_t* structure pointed to by *config* as a new, empty configuration.

`config_destroy()` destroys the configuration *config*, deallocating all memory associated with the configuration, but does not attempt to deallocate the *config_t* structure itself.

`int config_read` (*config_t * **config**, FILE * **stream***) [Function]

This function reads and parses a configuration from the given *stream* into the configuration object *config*. It returns `CONFIG_TRUE` on success, or `CONFIG_FALSE` on failure; the `config_error_text()`, `config_error_file()`, `config_error_line()`, and `config_error_type()` functions, described below, can be used to obtain information about the error.

`int config_read_file` (*config_t * **config**, const char * **filename***) [Function]

This function reads and parses a configuration from the file named *filename* into the configuration object *config*. It returns `CONFIG_TRUE` on success, or `CONFIG_FALSE` on failure; the `config_error_text()` and `config_error_line()` functions, described below, can be used to obtain information about the error.

`int config_read_string` (*config_t * **config**, const char * **str***) [Function]

This function reads and parses a configuration from the string *str* into the configuration object *config*. It returns `CONFIG_TRUE` on success, or `CONFIG_FALSE` on failure; the `config_error_text()` and `config_error_line()` functions, described below, can be used to obtain information about the error.

`void config_write` (*const config_t * **config**, FILE * **stream***) [Function]

This function writes the configuration *config* to the given *stream*.

`int config_write_file` (*config_t * **config**, const char * **filename***) [Function]

This function writes the configuration *config* to the file named *filename*. It returns `CONFIG_TRUE` on success, or `CONFIG_FALSE` on failure.

`const char * config_error_text` (*const config_t * **config***) [Function]
`const char * config_error_file` (*const config_t * **config***) [Function]
`int config_error_line` (*const config_t * **config***) [Function]

These functions, which are implemented as macros, return the text, filename, and line number of the parse error, if one occurred during a call to `config_read()`, `config_read_string()`, or `config_read_file()`. Storage for the strings returned by `config_error_text()` and `config_error_file()` are managed by the library and released automatically when the configuration is destroyed; these strings must not be freed by the caller. If the error occurred in text that was read from a string or stream, `config_error_file()` will return NULL.

`config_error_t config_error_type` (*const config_t * **config***) [Function]

This function, which is implemented as a macro, returns the type of error that occurred during the last call to one of the read or write functions. The *config_error_t* type is an enumeration with the following values: `CONFIG_ERR_NONE`, `CONFIG_ERR_FILE_IO`, `CONFIG_ERR_PARSE`. These represent success, a file I/O error, and a parsing error, respectively.

`void config_set_include_dir` (*config_t *config, const char *include_dir*) [Function]

`const char * config_get_include_dir` (*const config_t *config*) [Function]

`config_set_include_dir()` specifies the include directory, *include_dir*, relative to which the files specified in '`@include`' directives will be located for the configuration *config*. By default, there is no include directory, and all include files are expected to be relative to the current working directory. If *include_dir* is `NULL`, the default behavior is reinstated.

For example, if the include directory is set to '`/usr/local/etc`', the include directive '`@include "configs/extra.cfg"`' would include the file '`/usr/local/etc/configs/extra.cfg`'.

`config_get_include_dir()` returns the current include directory for the configuration *config*, or `NULL` if none is set.

`void config_set_auto_convert` (*config_t *config, int flag*) [Function]

`int config_get_auto_convert` (*const config_t *config*) [Function]

`config_set_auto_convert()` enables number auto-conversion for the configuration *config* if *flag* is non-zero, and disables it otherwise. When this feature is enabled, an attempt to retrieve a floating point setting's value into an integer (or vice versa), or store an integer to a floating point setting's value (or vice versa) will cause the library to silently perform the necessary conversion (possibly leading to loss of data), rather than reporting failure. By default this feature is disabled.

`config_get_auto_convert()` returns `CONFIG_TRUE` if number auto-conversion is currently enabled for *config*; otherwise it returns `CONFIG_FALSE`.

`void config_set_default_format` (*config_t * config, short format*) [Function]

`short config_get_default_format` (*config_t * config*) [Function]

These functions, which are implemented as macros, set and get the default external format for settings in the configuration *config*. If a non-default format has not been set for a setting with `config_setting_set_format()`, this configuration-wide default format will be used instead when that setting is written to a file or stream.

`void config_set_tab_width` (*config_t * config, unsigned short width*) [Function]

`unsigned short config_get_tab_width` (*const config_t * config*) [Function]

These functions, which are implemented as macros, set and get the tab width for the configuration *config*. The tab width affects the formatting of the configuration when it is written to a file or stream: each level of nesting is indented by *width* spaces, or by a single tab character if *width* is 0. The tab width has no effect on parsing.

Valid tab widths range from 0 to 15. The default tab width is 2.

`int config_lookup_int` (*const config_t * **config**, const char * **path**, int * **value***) [Function]
`int config_lookup_int64` (*const config_t * **config**, const char * **path**, long long * **value***) [Function]
`int config_lookup_float` (*const config_t * **config**, const char * **path**, double * **value***) [Function]
`int config_lookup_bool` (*const config_t * **config**, const char * **path**, int * **value***) [Function]
`int config_lookup_string` (*const config_t * **config**, const char * **path**, const char ** **value***) [Function]

These functions look up the value of the setting in the configuration *config* specified by the path *path*. They store the value of the setting at *value* and return `CONFIG_TRUE` on success. If the setting was not found or if the type of the value did not match the type requested, they leave the data pointed to by *value* unmodified and return `CONFIG_FALSE`.

Storage for the string returned by `config_lookup_string()` is managed by the library and released automatically when the setting is destroyed or when the setting's value is changed; the string must not be freed by the caller.

`config_setting_t * config_lookup` (*const config_t * **config**, const char * **path***) [Function]

This function locates the setting in the configuration *config* specified by the path *path*. It returns a pointer to the `config_setting_t` structure on success, or `NULL` if the setting was not found.

`int config_setting_get_int` (*const config_setting_t * **setting***) [Function]
`long long config_setting_get_int64` (*const config_setting_t * **setting***) [Function]
`double config_setting_get_float` (*const config_setting_t * **setting***) [Function]
`int config_setting_get_bool` (*const config_setting_t * **setting***) [Function]
`const char * config_setting_get_string` (*const config_setting_t * **setting***) [Function]

These functions return the value of the given *setting*. If the type of the setting does not match the type requested, a 0 or `NULL` value is returned. Storage for the string returned by `config_setting_get_string()` is managed by the library and released automatically when the setting is destroyed or when the setting's value is changed; the string must not be freed by the caller.

`int config_setting_set_int` (*config_setting_t * **setting**, int **value***) [Function]
`int config_setting_set_int64` (*config_setting_t * **setting**, long long **value***) [Function]
`int config_setting_set_float` (*config_setting_t * **setting**, double **value***) [Function]

int **config_setting_set_bool** (*config_setting_t * **setting**, int **value***) [Function]

int **config_setting_set_string** (*config_setting_t * **setting**, const char * **value***) [Function]

These functions set the value of the given *setting* to *value*. On success, they return `CONFIG_TRUE`. If the setting does not match the type of the value, they return `CONFIG_FALSE`. `config_setting_set_string()` makes a copy of the passed string *value*, so it may be subsequently freed or modified by the caller without affecting the value of the setting.

int **config_setting_lookup_int** (*const config_setting_t * **setting**, const char * **name**, int * **value***) [Function]

int **config_setting_lookup_int64** (*const config_setting_t * **setting**, const char * **name**, long long * **value***) [Function]

int **config_setting_lookup_float** (*const config_setting_t * **setting**, const char * **name**, double * **value***) [Function]

int **config_setting_lookup_bool** (*const config_setting_t * **setting**, const char * **name**, int * **value***) [Function]

int **config_setting_lookup_string** (*const config_setting_t * **setting**, const char * **name**, const char ** **value***) [Function]

These functions look up the value of the child setting named *name* of the setting *setting*. They store the value at *value* and return `CONFIG_TRUE` on success. If the setting was not found or if the type of the value did not match the type requested, they leave the data pointed to by *value* unmodified and return `CONFIG_FALSE`.

Storage for the string returned by `config_setting_lookup_string()` is managed by the library and released automatically when the setting is destroyed or when the setting's value is changed; the string must not be freed by the caller.

short **config_setting_get_format** (*config_setting_t * **setting***) [Function]

int **config_setting_set_format** (*config_setting_t * **setting**, short **format***) [Function]

These functions get and set the external format for the setting *setting*.

The *format* must be one of the constants `CONFIG_FORMAT_DEFAULT` or `CONFIG_FORMAT_HEX`. All settings support the `CONFIG_FORMAT_DEFAULT` format. The `CONFIG_FORMAT_HEX` format specifies hexadecimal formatting for integer values, and hence only applies to settings of type `CONFIG_TYPE_INT` and `CONFIG_TYPE_INT64`. If *format* is invalid for the given setting, it is ignored.

If a non-default format has not been set for the setting, `config_setting_get_format()` returns the default format for the configuration, as set by `config_set_default_format()`.

`config_setting_set_format()` returns `CONFIG_TRUE` on success and `CONFIG_FALSE` on failure.

config_setting_t * **config_setting_get_member** (*config_setting_t * **setting**, const char * **name***) [Function]

This function fetches the child setting named *name* from the group *setting*. It returns the requested setting on success, or `NULL` if the setting was not found or if *setting* is not a group.

**config_setting_t * config_setting_get_elem** [Function]
(*const config_setting_t * **setting**, unsigned int **index***)

This function fetches the element at the given index *index* in the setting *setting*, which must be an array, list, or group. It returns the requested setting on success, or NULL if *index* is out of range or if *setting* is not an array, list, or group.

**int config_setting_get_int_elem** [Function]
(*const config_setting_t * **setting**, int **index***)

**long long config_setting_get_int64_elem** [Function]
(*const config_setting_t * **setting**, int **index***)

**double config_setting_get_float_elem** [Function]
(*const config_setting_t * **setting**, int **index***)

**int config_setting_get_bool_elem** [Function]
(*const config_setting_t * **setting**, int **index***)

**const char * config_setting_get_string_elem** [Function]
(*const config_setting_t * **setting**, int **index***)

These functions return the value at the specified index *index* in the setting *setting*. If the setting is not an array or list, or if the type of the element does not match the type requested, or if *index* is out of range, they return 0 or NULL. Storage for the string returned by `config_setting_get_string_elem()` is managed by the library and released automatically when the setting is destroyed or when its value is changed; the string must not be freed by the caller.

**config_setting_t * config_setting_set_int_elem** [Function]
(*config_setting_t * **setting**, int **index**, int **value***)

**config_setting_t * config_setting_set_int64_elem** [Function]
(*config_setting_t * **setting**, int **index**, long long **value***)

**config_setting_t * config_setting_set_float_elem** [Function]
(*config_setting_t * **setting**, int **index**, double **value***)

**config_setting_t * config_setting_set_bool_elem** [Function]
(*config_setting_t * **setting**, int **index**, int **value***)

**config_setting_t * config_setting_set_string_elem** [Function]
(*config_setting_t * **setting**, int **index**, const char * **value***)

These functions set the value at the specified index *index* in the setting *setting* to *value*. If *index* is negative, a new element is added to the end of the array or list. On success, these functions return a pointer to the setting representing the element. If the setting is not an array or list, or if the setting is an array and the type of the array does not match the type of the value, or if *index* is out of range, they return NULL. `config_setting_set_string_elem()` makes a copy of the passed string *value*, so it may be subsequently freed or modified by the caller without affecting the value of the setting.

**config_setting_t * config_setting_add** [Function]
(*config_setting_t * **parent**, const char * **name**, int **type***)

This function adds a new child setting or element to the setting *parent*, which must be a group, array, or list. If *parent* is an array or list, the *name* parameter is ignored and may be NULL.

The function returns the new setting on success, or `NULL` if *parent* is not a group, array, or list; or if there is already a child setting of *parent* named *name*; or if *type* is invalid. If *type* is a scalar type, the new setting will have a default value of 0, 0.0, `false`, or `NULL`, as appropriate.

`int config_setting_remove` (*config_setting_t * **parent**, const char * **name***) [Function]

This function removes and destroys the setting named *name* from the parent setting *parent*, which must be a group. Any child settings of the setting are recursively destroyed as well.

The function returns `CONFIG_TRUE` on success. If *parent* is not a group, or if it has no setting with the given name, it returns `CONFIG_FALSE`.

`int config_setting_remove_elem` (*config_setting_t * **parent**, unsigned int **index***) [Function]

This function removes the child setting at the given index *index* from the setting *parent*, which must be a group, list, or array. Any child settings of the removed setting are recursively destroyed as well.

The function returns `CONFIG_TRUE` on success. If *parent* is not a group, list, or array, or if *index* is out of range, it returns `CONFIG_FALSE`.

`config_setting_t * config_root_setting` (*const config_t * **config***) [Function]

This function returns the root setting for the configuration *config*. The root setting is a group.

`const char * config_setting_name` (*const config_setting_t * **setting***) [Function]

This function returns the name of the given *setting*, or `NULL` if the setting has no name. Storage for the returned string is managed by the library and released automatically when the setting is destroyed; the string must not be freed by the caller.

`config_setting_t * config_setting_parent` (*const config_setting_t * **setting***) [Function]

This function returns the parent setting of the given *setting*, or `NULL` if *setting* is the root setting.

`int config_setting_is_root` (*const config_setting_t * **setting***) [Function]

This function returns `CONFIG_TRUE` if the given *setting* is the root setting, and `CONFIG_FALSE` otherwise.

`int config_setting_index` (*const config_setting_t * **setting***) [Function]

This function returns the index of the given *setting* within its parent setting. If *setting* is the root setting, this function returns -1.

`int config_setting_length` (*const config_setting_t * **setting***) [Function]

This function returns the number of settings in a group, or the number of elements in a list or array. For other types of settings, it returns 0.

**int config_setting_type** (*const config_setting_t * setting*) [Function]

This function returns the type of the given *setting*. The return value is one of the constants `CONFIG_TYPE_INT`, `CONFIG_TYPE_INT64`, `CONFIG_TYPE_FLOAT`, `CONFIG_TYPE_STRING`, `CONFIG_TYPE_BOOL`, `CONFIG_TYPE_ARRAY`, `CONFIG_TYPE_LIST`, or `CONFIG_TYPE_GROUP`.

**int config_setting_is_group** (*const config_setting_t * setting*) [Function]
**int config_setting_is_array** (*const config_setting_t * setting*) [Function]
**int config_setting_is_list** (*const config_setting_t * setting*) [Function]

These convenience functions, which are implemented as macros, test if the setting *setting* is of a given type. They return `CONFIG_TRUE` or `CONFIG_FALSE`.

**int config_setting_is_aggregate** [Function]
(*const config_setting_t * setting*)
**int config_setting_is_scalar** (*const config_setting_t * setting*) [Function]
**int config_setting_is_number** (*const config_setting_t * setting*) [Function]

These convenience functions, which are implemented as macros, test if the setting *setting* is of an aggregate type (a group, array, or list), of a scalar type (integer, 64-bit integer, floating point, boolean, or string), and of a number (integer, 64-bit integer, or floating point), respectively. They return `CONFIG_TRUE` or `CONFIG_FALSE`.

**const char * config_setting_source_file** [Function]
(*const config_setting_t * setting*)

This function returns the name of the file from which the setting *setting* was read, or NULL if the setting was not read from a file. This information is useful for reporting application-level errors. Storage for the returned string is managed by the library and released automatically when the configuration is destroyed; the string must not be freed by the caller.

**unsigned int config_setting_source_line** [Function]
(*const config_setting_t * setting*)

This function returns the line number of the configuration file or stream at which the setting *setting* was read, or 0 if no line number is available. This information is useful for reporting application-level errors.

**void config_setting_set_hook** (*config_setting_t * setting*, [Function]
*void * hook*)
**void * config_setting_get_hook** (*const config_setting_t * setting*) [Function]

These functions make it possible to attach arbitrary data to each setting structure, for instance a “wrapper” or “peer” object written in another programming language. The destructor function, if one has been supplied via a call to `config_set_destructor()`, will be called by the library to dispose of this data when the setting itself is destroyed. There is no default destructor.

**void config_set_destructor** (*config_t * config*, [Function]
*void (* destructor)(void *)*)

This function assigns the destructor function *destructor* for the configuration *config*. This function accepts a single `void *` argument and has no return value. See `config_setting_set_hook()` above for more information.

# 4 The C++ API

This chapter describes the C++ library API. The class `Config` represents a configuration, and the class `Setting` represents a configuration setting. Note that by design, neither of these classes provides a public copy constructor or assignment operator. Therefore, instances of these classes may only be passed between functions via references or pointers.

The library defines a group of exceptions, all of which extend the common base exception `ConfigException`.

A `SettingTypeException` is thrown when the type of a setting's value does not match the type requested.

A `SettingNotFoundException` is thrown when a setting is not found.

A `SettingNameException` is thrown when an attempt is made to add a new setting with a non-unique or invalid name.

A `ParseException` is thrown when a parse error occurs while reading a configuration from a stream.

A `FileIOException` is thrown when an I/O error occurs while reading/writing a configuration from/to a file.

`SettingTypeException`, `SettingNotFoundException`, and `SettingNameException` all extend the common base exception `SettingException`, which provides the following method:

`const char * getPath ()` [Method on `SettingException`]

Returns the path to the setting associated with the exception, or `NULL` if there is no applicable path.

The remainder of this chapter describes the methods for manipulating configurations and configuration settings.

`Config ()` [Method on `Config`]
`~Config ()` [Method on `Config`]

These methods create and destroy `Config` objects.

`void read (`*FILE ** ***stream***`)` [Method on `Config`]
`void write (`*FILE ** ***stream***`)` [Method on `Config`]

The `read()` method reads and parses a configuration from the given *stream*. A `ParseException` is thrown if a parse error occurs.

The `write()` method writes the configuration to the given *stream*.

`void readFile (`*const char ** ***filename***`)` [Method on `Config`]
`void writeFile (`*const char ** ***filename***`)` [Method on `Config`]

The `readFile()` method reads and parses a configuration from the file named *filename*. A `ParseException` is thrown if a parse error occurs. A `FileIOException` is thrown if the file cannot be read.

The `writeFile()` method writes the configuration to the file named *filename*. A `FileIOException` is thrown if the file cannot be written.

`void readString (const char * str)` [Method on `Config`]
`void readString (const std::string &str)` [Method on `Config`]

These methods read and parse a configuration from the string *str*. A `ParseException` is thrown if a parse error occurs.

`const char * getError ()` [Method on `ParseException`]
`const char * getFile ()` [Method on `ParseException`]
`int getLine ()` [Method on `ParseException`]

If a call to `readFile()`, `readString()`, or `read()` resulted in a `ParseException`, these methods can be called on the exception object to obtain the text, filename, and line number of the parse error. Storage for the strings returned by `getError()` and `getFile()` are managed by the library; the strings must not be freed by the caller.

`void setIncludeDir (const char *includeDir)` [Method on `Config`]
`const char * getIncludeDir ()` [Method on `Config`]

`setIncludeDir()` specifies the include directory, *includeDir*, relative to which the files specified in '`@include`' directives will be located for the configuration. By default, there is no include directory, and all include files are expected to be relative to the current working directory. If *includeDir* is `NULL`, the default behavior is reinstated.

For example, if the include directory is set to '`/usr/local/etc`', the include directive '`@include "configs/extra.cfg"`' would include the file '`/usr/local/etc/configs/extra.cfg`'.

`getIncludeDir()` returns the current include directory for the configuration, or `NULL` if none is set.

`void setAutoConvert (bool flag)` [Method on `Config`]
`bool getAutoConvert ()` [Method on `Config`]

`setAutoConvert()` enables number auto-conversion for the configuration if *flag* is `true`, and disables it otherwise. When this feature is enabled, an attempt to assign a floating point setting to an integer (or vice versa), or assign an integer to a floating point setting (or vice versa) will cause the library to silently perform the necessary conversion (possibly leading to loss of data), rather than throwing a `SettingTypeException`. By default this feature is disabled.

`getAutoConvert()` returns `true` if number auto-conversion is currently enabled for the configuration; otherwise it returns `false`.

`void setDefaultFormat (Setting::Format format)` [Method on `Config`]
`Setting::Format getDefaultFormat ()` [Method on `Config`]

These methods set and get the default external format for settings in the configuration. If a non-default format has not been set for a setting with `Setting::setFormat()`, this configuration-wide default format will be used instead when that setting is written to a file or stream.

`void setTabWidth (unsigned short width)` [Method on `Config`]
`unsigned short getTabWidth ()` [Method on `Config`]

These methods set and get the tab width for the configuration. The tab width affects the formatting of the configuration when it is written to a file or stream: each level of nesting is indented by *width* spaces, or by a single tab character if *width* is 0. The tab width has no effect on parsing.

Valid tab widths range from 0 to 15. The default tab width is 2.

`Setting & getRoot ()` [Method on `Config`]

This method returns the root setting for the configuration, which is a group.

`Setting & lookup` (*const std::string &*`path`) [Method on `Config`]
`Setting & lookup` (*const char ** `path`) [Method on `Config`]

These methods locate the setting specified by the path *path*. If the requested setting is not found, a `SettingNotFoundException` is thrown.

`bool exists` (*const std::string &*`path`) [Method on `Config`]
`bool exists` (*const char **`path`) [Method on `Config`]

These methods test if a setting with the given *path* exists in the configuration. They return `true` if the setting exists, and `false` otherwise. These methods do not throw exceptions.

`bool lookupValue` (*const char **`path`, *bool &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *bool &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *int &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *int &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *unsigned int &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *unsigned int &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *long long &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *long long &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *float &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *float &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *double &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *double &*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *const char *&*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *const char *&*`value`) [Method on `Config`]
`bool lookupValue` (*const char **`path`, *std::string &*`value`) [Method on `Config`]
`bool lookupValue` (*const std::string &*`path`, *std::string &*`value`) [Method on `Config`]

These are convenience methods for looking up the value of a setting with the given *path*. If the setting is found and is of an appropriate type, the value is stored in *value* and the method returns `true`. Otherwise, *value* is left unmodified and the method returns `false`. These methods do not throw exceptions.

Storage for *const char ** values is managed by the library and released automatically when the setting is destroyed or when its value is changed; the string must not be freed by the caller. For safety and convenience, always assigning string values to a `std::string` is suggested.

Since these methods have boolean return values and do not throw exceptions, they can be used within boolean logic expressions. The following example presents a concise way to look up three values at once and perform error handling if any of them are not found or are of the wrong type:

```
int var1;
double var2;
const char *var3;

if(config.lookupValue("values.var1", var1)
   && config.lookupValue("values.var2", var2)
   && config.lookupValue("values.var3", var3))
{
  // use var1, var2, var3
}
else
{
  // error handling here
}
```

This approach also takes advantage of the short-circuit evaluation rules of C++, e.g., if the first lookup fails (returning `false`), the remaining lookups are skipped entirely.

`operator bool ()` [Method on `Setting`]
`operator int ()` [Method on `Setting`]
`operator unsigned int ()` [Method on `Setting`]
`operator long ()` [Method on `Setting`]
`operator unsigned long ()` [Method on `Setting`]
`operator long long ()` [Method on `Setting`]
`operator unsigned long long ()` [Method on `Setting`]
`operator float ()` [Method on `Setting`]
`operator double ()` [Method on `Setting`]
`operator const char * ()` [Method on `Setting`]
`operator std::string ()` [Method on `Setting`]
`const char * c_str ()` [Method on `Setting`]

These cast operators allow a `Setting` object to be assigned to a variable of type *bool* if it is of type `TypeBoolean`; *int*, *unsigned int*; `long long` or `unsigned long long` if it is of type `TypeInt64`, *float* or *double* if it is of type `TypeFloat`; or *const char ** or *std::string* if it is of type `TypeString`.

Values of type `TypeInt` or `TypeInt64` may be assigned to variables of type *long*, or *unsigned long*, depending on the sizes of those types on the host system.

Storage for *const char ** return values is managed by the library and released automatically when the setting is destroyed or when its value is changed; the string must not be freed by the caller. For safety and convenience, always assigning string return values to a `std::string` is suggested.

The following examples demonstrate this usage:

```
long width = config.lookup("application.window.size.w");

bool splashScreen = config.lookup("application.splash_screen");

std::string title = config.lookup("application.window.title");
```

Note that certain conversions can lead to loss of precision or clipping of values, e.g., assigning a negative value to an *unsigned int* (in which case the value will be treated as 0), or a double-precision value to a *float*. The library does not treat these lossy conversions as errors.

Perhaps surprisingly, the following code in particular will cause a compiler error:

```
std::string title;
.
.
.
title = config.lookup("application.window.title");
```

This is because the assignment operator of `std::string` is being invoked with a `Setting &` as an argument. The compiler is unable to make an implicit conversion because both the `const char *` and the `std::string` cast operators of `Setting` are equally appropriate. This is not a bug in *libconfig*; providing only the `const char *` cast operator would resolve this particular ambiguity, but would cause assignments to `std::string` like the one in the previous example to produce a compiler error. (To understand why, see section 11.4.1 of *The C++ Programming Language*.)

The solution to this problem is to use an explicit conversion that avoids the construction of an intermediate `std::string` object, as follows:

```
std::string title;
.
.
.
title = (const char *)config.lookup("application.window.title");
```

Or, alternatively, use the `c_str()` method, which has the same effect:

```
std::string title;
.
.
.
title = config.lookup("application.window.title").c_str();
```

If the assignment is invalid due to a type mismatch, a `SettingTypeException` is thrown.

`Setting & operator=` (*bool* `value`) [Method on `Setting`]
`Setting & operator=` (*int* `value`) [Method on `Setting`]
`Setting & operator=` (*long* `value`) [Method on `Setting`]
`Setting & operator=` (*const long long &*`value`) [Method on `Setting`]
`Setting & operator=` (*float* `value`) [Method on `Setting`]

**Setting & operator=** (*const double &*`value`) [Method on `Setting`]
**Setting & operator=** (*const char **`value`) [Method on `Setting`]
**Setting & operator=** (*const std::string &*`value`) [Method on `Setting`]

These assignment operators allow values of type *bool*, *int*, *long*, *long long*, *float*, *double*, *const char **, and *std::string* to be assigned to a setting. In the case of strings, the library makes a copy of the passed string *value*, so it may be subsequently freed or modified by the caller without affecting the value of the setting.

The following example code looks up a (presumably) integer setting and changes its value:

```
Setting &setting = config.lookup("application.window.size.w");
setting = 1024;
```

If the assignment is invalid due to a type mismatch, a `SettingTypeException` is thrown.

**Setting & operator[]** (*int* `index`) [Method on `Setting`]
**Setting & operator[]** (*const std::string &*`name`) [Method on `Setting`]
**Setting & operator[]** (*const char **`name`) [Method on `Setting`]

A `Setting` object may be subscripted with an integer index *index* if it is an array or list, or with either a string *name* or an integer index *index* if it is a group. For example, the following code would produce the string '`Last Name`' when applied to the example configuration in Chapter 2 [Configuration Files], page 5.

```
Setting& setting = config.lookup("application.misc");
const char *s = setting["columns"][0];
```

If the setting is not an array, list, or group, a `SettingTypeException` is thrown. If the subscript (*index* or *name*) does not refer to a valid element, a `SettingNotFoundException` is thrown.

Iterating over a group's child settings with an integer index will return the settings in the same order that they appear in the configuration.

**bool lookupValue** (*const char **`name`, *bool &*`value`) [Method on `Setting`]
**bool lookupValue** (*const std::string &*`name`, *bool &*`value`) [Method on `Setting`]
**bool lookupValue** (*const char **`name`, *int &*`value`) [Method on `Setting`]
**bool lookupValue** (*const std::string &*`name`, *int &*`value`) [Method on `Setting`]
**bool lookupValue** (*const char **`name`, *unsigned int &*`value`) [Method on `Setting`]
**bool lookupValue** (*const std::string &*`name`, *unsigned int &*`value`) [Method on `Setting`]
**bool lookupValue** (*const char **`name`, *long long &*`value`) [Method on `Setting`]
**bool lookupValue** (*const std::string &*`name`, *long long &*`value`) [Method on `Setting`]
**bool lookupValue** (*const char **`name`, *unsigned long long &*`value`) [Method on `Setting`]
**bool lookupValue** (*const std::string &*`name`, *unsigned long long &*`value`) [Method on `Setting`]

**bool lookupValue** (*const char *name, float &value*) [Method on Setting]
**bool lookupValue** (*const std::string &name, float &value*) [Method on Setting]
**bool lookupValue** (*const char *name, double &value*) [Method on Setting]
**bool lookupValue** (*const std::string &name, double &value*) [Method on Setting]
**bool lookupValue** (*const char *name, const char *&value*) [Method on Setting]
**bool lookupValue** (*const std::string &name, const char *&value*) [Method on Setting]
**bool lookupValue** (*const char *name, std::string &value*) [Method on Setting]
**bool lookupValue** (*const std::string &name, std::string &value*) [Method on Setting]

These are convenience methods for looking up the value of a child setting with the given *name*. If the setting is found and is of an appropriate type, the value is stored in *value* and the method returns `true`. Otherwise, *value* is left unmodified and the method returns `false`. These methods do not throw exceptions.

Storage for *const char ** values is managed by the library and released automatically when the setting is destroyed or when its value is changed; the string must not be freed by the caller. For safety and convenience, always assigning string values to a `std::string` is suggested.

Since these methods have boolean return values and do not throw exceptions, they can be used within boolean logic expressions. The following example presents a concise way to look up three values at once and perform error handling if any of them are not found or are of the wrong type:

```
int var1;
double var2;
const char *var3;

if(setting.lookupValue("var1", var1)
   && setting.lookupValue("var2", var2)
   && setting.lookupValue("var3", var3))
{
  // use var1, var2, var3
}
else
{
  // error handling here
}
```

This approach also takes advantage of the short-circuit evaluation rules of C++, e.g., if the first lookup fails (returning `false`), the remaining lookups are skipped entirely.

**Setting & add** (*const std::string &name, Setting::Type type*) [Method on Setting]
**Setting & add** (*const char *name, Setting::Type type*) [Method on Setting]

These methods add a new child setting with the given *name* and *type* to the setting, which must be a group. They return a reference to the new setting. If the setting already has a child setting with the given name, or if the name is invalid, a `SettingNameException` is thrown. If the setting is not a group, a `SettingTypeException` is thrown.

Once a setting has been created, neither its name nor type can be changed.

`Setting & add` (*Setting::Type* ***type***) [Method on `Setting`]

This method adds a new element to the setting, which must be of type `TypeArray` or `TypeList`. If the setting is an array which currently has zero elements, the *type* parameter (which must be `TypeInt`, `TypeInt64`, `TypeFloat`, `TypeBool`, or `TypeString`) determines the type for the array; otherwise it must match the type of the existing elements in the array.

The method returns the new setting on success. If *type* is a scalar type, the new setting will have a default value of 0, 0.0, `false`, or `NULL`, as appropriate.

The method throws a `SettingTypeException` if the setting is not an array or list, or if *type* is invalid.

`void remove` (*const std::string &***name**) [Method on `Setting`]
`void remove` (*const char ****name**) [Method on `Setting`]

These methods remove the child setting with the given *name* from the setting, which must be a group. Any child settings of the removed setting are recursively destroyed as well.

If the setting is not a group, a `SettingTypeException` is thrown. If the setting does not have a child setting with the given name, a `SettingNotFoundException` is thrown.

`void remove` (*unsigned int* ***index***) [Method on `Setting`]

This method removes the child setting at the given index *index* from the setting, which must be a group, list, or array. Any child settings of the removed setting are recursively destroyed as well.

If the setting is not a group, list, or array, a `SettingTypeException` is thrown. If *index* is out of range, a `SettingNotFoundException` is thrown.

`const char * getName` () [Method on `Setting`]

This method returns the name of the setting, or `NULL` if the setting has no name. Storage for the returned string is managed by the library and released automatically when the setting is destroyed; the string must not be freed by the caller. For safety and convenience, consider assigning the return value to a `std::string`.

`std::string getPath` () [Method on `Setting`]

This method returns the complete dot-separated path to the setting. Settings which do not have a name (list and array elements) are represented by their index in square brackets.

`Setting & getParent` () [Method on `Setting`]

This method returns the parent setting of the setting. If the setting is the root setting, a `SettingNotFoundException` is thrown.

`bool isRoot` () [Method on `Setting`]

This method returns `true` if the setting is the root setting, and `false` otherwise.

`int getIndex ()` [Method on `Setting`]

This method returns the index of the setting within its parent setting. When applied to the root setting, this method returns -1.

`Setting::Type getType ()` [Method on `Setting`]

This method returns the type of the setting. The `Setting::Type` enumeration consists of the following constants: `TypeInt`, `TypeInt64`, `TypeFloat`, `TypeString`, `TypeBoolean`, `TypeArray`, `TypeList`, and `TypeGroup`.

`Setting::Format getFormat ()` [Method on `Setting`]
`void setFormat` (*Setting::Format* ***format***) [Method on `Setting`]

These methods get and set the external format for the setting.

The *Setting::Format* enumeration consists of the following constants: `FormatDefault` and `FormatHex`. All settings support the `FormatDefault` format. The `FormatHex` format specifies hexadecimal formatting for integer values, and hence only applies to settings of type `TypeInt` and `TypeInt64`. If *format* is invalid for the given setting, it is ignored.

`bool exists` (*const std::string &*`name`) [Method on `Setting`]
`bool exists` (*const char **`name`) [Method on `Setting`]

These methods test if the setting has a child setting with the given *name*. They return `true` if the setting exists, and `false` otherwise. These methods do not throw exceptions.

`int getLength ()` [Method on `Setting`]

This method returns the number of settings in a group, or the number of elements in a list or array. For other types of settings, it returns 0.

`bool isGroup ()` [Method on `Setting`]
`bool isArray ()` [Method on `Setting`]
`bool isList ()` [Method on `Setting`]

These convenience methods test if a setting is of a given type.

`bool isAggregate ()` [Method on `Setting`]
`bool isScalar ()` [Method on `Setting`]
`bool isNumber ()` [Method on `Setting`]

These convenience methods test if a setting is of an aggregate type (a group, array, or list), of a scalar type (integer, 64-bit integer, floating point, boolean, or string), and of a number (integer, 64-bit integer, or floating point), respectively.

`const char * getSourceFile ()` [Method on `Setting`]

This function returns the name of the file from which the setting was read, or NULL if the setting was not read from a file. This information is useful for reporting application-level errors. Storage for the returned string is managed by the library and released automatically when the configuration is destroyed; the string must not be freed by the caller.

`unsigned int getSourceLine ()` [Method on `Setting`]

This function returns the line number of the configuration file or stream at which the setting *setting* was read, or 0 if no line number is available. This information is useful for reporting application-level errors.

# 5 Example Programs

Practical example programs that illustrate how to use *libconfig* from both C and C++ are included in the '`examples`' subdirectory of the distribution. These examples include:

'`examples/c/example1.c`'
: An example C program that reads a configuration from an existing file '`example.cfg`' (also located in '`examples/c`') and displays some of its contents.

'`examples/c++/example1.cpp`'
: The C++ equivalent of '`example1.c`'.

'`examples/c/example2.c`'
: An example C program that reads a configuration from an existing file '`example.cfg`' (also located in '`examples/c`'), adds new settings to the configuration, and writes the updated configuration to another file.

'`examples/c++/example2.cpp`'
: The C++ equivalent of '`example2.c`'

'`examples/c/example3.c`'
: An example C program that constructs a new configuration in memory and writes it to a file.

'`examples/c++/example3.cpp`'
: The C++ equivalent of '`example3.c`'

# 6 Configuration File Grammar

Below is the BNF grammar for configuration files. Comments and include directives are not part of the grammar, so they are not included here.

```
configuration = setting-list | empty

setting-list = setting | setting-list setting

setting = name (":" | "=") value (";" | "," | empty)

value = scalar-value | array | list | group

value-list = value | value-list "," value

scalar-value = boolean | integer | integer64 | hex | hex64 | float
               | string

scalar-value-list = scalar-value | scalar-value-list "," scalar-value

array = "[" (scalar-value-list | empty) "]"

list = "(" (value-list | empty) ")"

group = "{" (setting-list | empty) "}"

empty =
```

Terminals are defined below as regular expressions:

| | |
|---|---|
| boolean | `([Tt][Rr][Uu][Ee])|([Ff][Aa][Ll][Ss][Ee])` |
| string | `\"([^\"\\]|\\.)*\"` |
| name | `[A-Za-z\*][-A-Za-z0-9_\*]*` |
| integer | `[-+]?[0-9]+` |
| integer64 | `[-+]?[0-9]+L(L)?` |
| hex | `0[Xx][0-9A-Fa-f]+` |
| hex64 | `0[Xx][0-9A-Fa-f]+L(L)?` |
| float | `([-+]?([0-9]*)?\.[0-9]*([eE][-+]?[0-9]+)?)|([-+]([0-9]+)(\.[0-9]*)?[eE][-+]?[0-9]+)` |

# Appendix A License



[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

   c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

   d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights,

from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

# Function Index

# Type Index

## C



## F



## P



## S

# Concept Index