NAME

lc_cleanup − Clean up internal structures after processing data.

SYNOPSIS

#include <libconfig.h>

void lc_cleanup(void);

DESCRIPTION

The **lc_cleanup**(3) function cleans up the internal structures created by calling **lc_register_var**(3) or **lc_register_callback**(3) and returns the memory to the application. It is not strictly required, however memory concious programmers will still want to call this after finishing processing configuration files.

After you call **lc_cleanup**(3) calling **lc_process**(3) or **lc_process_file**(3) will generally cause errors since the registered variables and callbacks have been unregistered.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');

        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                        lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                        NULL);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                        %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterrstr**(3), **lc_geterrno**(3), **lc_process_file**(3), **lc_process**(3)

NAME

lc_geterrno − Retrieve a numeric error code.

SYNOPSIS

#include <libconfig.h>

lc_err_t lc_geterrno(void);

DESCRIPTION

The **lc_geterrno**(3) function returns the last numeric error code set as an *lc_err_t* which is an enumerated type and can be cast to any integer type.

The *lc_err_t* type specifies the following defined values:

LC_ERR_NONE
: No error was found. Success.

LC_ERR_INVCMD
: A command was specified for which there was no handler.

LC_ERR_INVSECTION
: A section was specified for which there was no handler.

LC_ERR_INVDATA
: A value that does not make sense was specified, such as a non-existant type to the **lc_process_file**(3) function.

LC_ERR_BADFORMAT
: An invalid format was detected, such as no argument where on was expected, or a value passed to a boolean-specified variable whose value was not one of: enable, true, yes, on, y, 1, disable, false, off, no, n, 0.

LC_ERR_CANTOPEN
: Unable to open a specified file.

LC_ERR_CALLBACK
: A callback function returned an error (LC_CBRET_ERROR).

LC_ERR_ENOMEM
: Memory could not be allocated for internal structures.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');

        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                        lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                        NULL);
```

```
        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                    %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process_file**(3), **lc_process**(3)

NAME

lc_geterrstr − Retrieve a human readable error message.

SYNOPSIS

#include <libconfig.h>

char *lc_geterrstr(void);

DESCRIPTION

The **lc_geterrstr**(3) function returns a string describing the last error code set.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');

        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %s.\n",
                        lc_geterrstr());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                        NULL);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                        %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterrno**(3), **lc_cleanup**(3), **lc_process_file**(3), **lc_process**(3)

NAME

lc_process − Begin processing configuration files.

SYNOPSIS

#include <libconfig.h>

int lc_process(int *argc***, char *****argv***, const char ****appname***, lc_conf_type_t** *type***, const char ****extra***);**

DESCRIPTION

The **lc_process**(3) function begins processing of the command line arguments, environment variables, and command line options. The *argc* and *argv* parameters should be in the same format as passed to the **main** function of your program. The *appname* parameter should be a reasonable form of the name of the application. The *type* parameter should describe the format of the configuration file (see below). The *extra* parameter should list any extra configuration files to process.

Valid type parameter values:

LC_CONF_SECTION

This type of configuration file is similar to the Windows INI-file style. An example configuration file:

```
[section]
variable = value
```

LC_CONF_APACHE

This type of configuration file is similar to the Apache configuration file. An example configuration file:

```
<Section argument>
        variable value
</Section>
```

LC_CONF_SPACE

This is a simple, flat configuration file. It has no section headers. An example configuration file:

```
variable value
```

RETURN VALUE

On success 0 is returned, otherwise -1 is returned.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');
        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                      lc_geterrno());
                return(EXIT_FAILURE);
        }
```

```
        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                NULL);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                    %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterrno**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process_file**(3)

NAME

lc_process_file − Process a specific file

SYNOPSIS

#include <libconfig.h>

int lc_process_file(const char * *appname***, const char *** *pathname***, lc_conf_type_t** *type***);**

DESCRIPTION

The **lc_process_file**(3) function processes exactly one configuration file. The file is specified by the *pathname* argument and should be in the format specified by the *type* argument. The *appname* argument should be a reasonable form of the name of the application.

RETURN VALUE

On success 0 is returned, otherwise -1 is returned.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');
        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                      lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process_file("example", "/data/extra.conf",
                        LC_CONF_APACHE);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration file: \
                      %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterrno**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process**(3)

NAME

lc_register_callback − Register a function for callback in config processing.

SYNOPSIS

#include <libconfig.h>

int lc_register_callback(const char *_var_, char _opt_, lc_var_type_t _type_, int (*_callback_)(const char *, const char *, const char *, const char *, lc_flags_t, void *), void *_extra_);

DESCRIPTION

The **lc_register_callback**(3) function registers a function to be called when *var* is encounted in a configuration file, command line, or environment variable. The parameters are as follows:

*const char *var*

The *var* parameter indicates the name of the variable to register for a callback when encountered in a configuration file, the environment, or as a long option. The *var* may be prefixed with "*." to indicate that the object can occur in any section or subsection.

const char opt

The *opt* parameter indicates the single charectar short option to use from the command line to invoke the register callback. A value of 0 indicates that no short option is acceptable.

lc_var_type_t type

The *type* parameter indicates the type of values that are acceptable for this callback. A value of LC_VAR_NONE means that the command will accept no arguments, while a value of LC_VAR_UNKNOWN indicates that it's not known whether or not an argument is applicable, this will also disable command line processing. Any other value is currently ignored.

*int (*callback)(...)*

The *callback* parameter indicates the name of the function to invoke when the above parameters are met. The specified function should take 6 parameters, see below for more information. This value may not be NULL.

*void *extra*

The *extra* parameter is a pointer that can be used to pass data to the callback upon invocation, it will not be mangled or examined by any function.

The arguments to the function specified as *callback* are as follows:

*const char *shortvar*

The *shortvar* parameter is the local variable name, provided as a convience. It is the portion of the variable name after the first "dot" (.) in the fully qualified variable name. The "dot" (.) value in the fully qualified variable name indicates a section or subsection that the variable belongs to. This may be **NULL** if the *var* parameter to **lc_register_callback**(3) was **NULL** too.

*const char *var*

The *var* parameter is the fully qualified variable name. It includes in the prefix any sections and subsections that contain this variable. This may be **NULL** if the *var* parameter to **lc_register_callback**(3) was **NULL** too.

*const char *arguments*

The *arguments* parameter provides the arguments passed to the variable, currently only sections may have arguments. This may be **NULL** if there were no arguments specified, or if arguments were not applicable.

*const char *value*
The *value* parameter provides the value of the variable specified. This may be **NULL** if no value was specified. Values are required if the *type* parameter to **lc_register_callback**(3) was not specified as one of LC_VAR_NONE, LC_VAR_SECTION, LC_VAR_SECTIONSTART, or LC_VAR_SECTIONEND.

lc_flags_t flags
The flags parameter provides information about the type of command being called. The valid values are:

LC_FLAGS_VAR
To indicate a regular variable in a configuration file.

LC_FLAGS_CMDLINE
To indicate a command line option has been used to invoke this option.

LC_FLAGS_SECTIONSTART
To indicate that this command represents the beginning of a section.

LC_FLAGS_SECTIONEND
To indicate that this command represents the end of a section.

*void *extra*
The *extra* parameter is just a copy of the *extra* parameter passed to **lc_register_callback**(3) when the callback was registered.

The *callback* function should return one of three values:

LC_CBRET_IGNORESECTION
Returning LC_CBRET_IGNORESECTION from a callback that begins a section causes the entire section to be ignored without generating an error.

LC_CBRET_OKAY
Returning LC_CBRET_OKAY from a callback indicates that all went well and further processing may continue.

LC_CBRET_ERROR
Returnning LC_CBRET_ERROR from a callback indicates that the command failed for some reason, the error will be passed back down the chain back to the **lc_process**(3) call that began processing the configuration data. If LC_CBRET_ERROR is returned from a callback that begins a section, the entire section is ignored.

RETURN VALUE

On success 0 is returned, otherwise -1 is returned.

EXAMPLE

```
#include <libconfig.h>
#include <strings.h>
#include <stdlib.h>
#include <stdio.h>

int callback_ifmodule(const char *shortvar, const char *var,
        const char *arguments, const char *value,
        lc_flags_t flags, void *extra) {
    if (flags == LC_FLAGS_SECTIONEND) {
        return(LC_CBRET_OKAY);
    }

    if (flags != LC_FLAGS_SECTIONSTART) {
        fprintf(stderr, "IfModule can only be used as a \
            section.\n");
        return(LC_CBRET_ERROR);
```

```
        }
        if (arguments == NULL) {
                fprintf(stderr, "You must specify an argument to \
                    IfModule.\n");
                return(LC_CBRET_ERROR);
        }

        printf("IfModule %s\n", arguments);

        if (strcasecmp(arguments, "MyModule") == 0) {
                return(LC_CBRET_IGNORESECTION);
        }

        return(LC_CBRET_OKAY);
}

int main(int argc, char **argv) {
        int lc_rc_ret = 0, lc_p_ret;

        lc_rc_ret = lc_register_callback("*.IfModule", 0, LC_VAR_NONE,
                        callback_ifmodule, NULL);

        if (lc_rc_ret != 0) {
                fprintf(stderr, "Error registering callback.\n");
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                    NULL);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                    %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        return(EXIT_SUCCESS);
}
```

ERRORS

ENOMEM

Memory could not be allocated to create the needed internal structures.

SEE ALSO

lc_register_var(3), **lc_geterrno**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process_file**(3), **lc_process**(3)

NAME

lc_register_var − Register a variable for automatic processing.

SYNOPSIS

#include <libconfig.h>

int lc_register_var(const char * *var* **, lc_var_type_t** *type***, void *** *data***, char** *opt***);**

DESCRIPTION

The **lc_register_var**(3) function registers a variable for automatic processing. The *var* parameter specifies the variable name for processing. This name can exist in a configuration file, an environment variable, or on the command line. The *opt* parameter specifies the single letter short option that can be specified on the command line to change the value of the variable specified by the *data* parameter. A value of ' ' can be specified for no short option.

The *type* parameter is of type *lc_var_type_t* which specifies the type of the *data* parameter. Valid values for *type* are:

LC_VAR_STRING
: For a string type variable. The data passed should be of type "char **". The data will be set to a region of memory that has been allocated with malloc() and can be released be free()'d.

LC_VAR_LONG_LONG
: For a "long long" integer type variable. The data passed should be of type "long long *".

LC_VAR_LONG
: For a "long" integer type variable. The data passed should be of type "long *".

LC_VAR_INT
: For a "int" integer type variable. The data passed should be of type "int *".

LC_VAR_SHORT
: For a "short" integer type variable. The data passed should be of type "short *".

LC_VAR_BOOL
: For a boolean type variable. The data passed should be of type "int *". When a true value is specified the variable is set to 1. When a false value is specified the variable is set to 0. Any other value sets the variable to -1. Valid true values are: enable, true, yes, on, y, and 1. Valid false values are: disable, false, off, no, n, and 0.

LC_VAR_FILENAME
: Not implemented.

LC_VAR_DIRECTORY
: Not implemented.

LC_VAR_SIZE_LONG_LONG
: For a "long long" integer type that can have size modifiers, such as 'G' or gigabytes, 'M' for megabytes, 'K' for kilobytes. The data passed should be of type "long long *".

LC_VAR_SIZE_LONG
: For a "long" integer type that can have size modifiers, such as 'G' or gigabytes, 'M' for megabytes, 'K' for kilobytes. The data passed should be of type "long *".

LC_VAR_SIZE_INT
: For a "int" integer type that can have size modifiers, such as 'G' or gigabytes, 'M' for megabytes, 'K' for kilobytes. The data passed should be of type "int *".

LC_VAR_SIZE_SHORT
: For a "short" integer type that can have size modifiers, such as 'G' or gigabytes, 'M' for megabytes, 'K' for kilobytes. The data passed should be of type "short *".

LC_VAR_SIZE_SIZE_T
: For a "size_t" data type that can have size modifiers, such as ’G’ or gigabytes, ’M’ for megabytes, ’K’ for kilobytes. The data passed should be of type "size_t *".

LC_VAR_TIME
: Not implemented.

LC_VAR_DATE
: Not implemented.

LC_VAR_BOOL_BY_EXISTANCE
: This type of variable takes no arguments, it is set to true (1) by its existance in a configuration file, environment variable, or on the command line. If it is not specified, the value of the data passed is not changed. The data passed should be of type "int *".

LC_VAR_CIDR
: This type of variable accepts a CIDR format netmask and IP. This is not yet implemented. (XXX)

LC_VAR_IP
: This type of variable accepts an IP address in decimal-dot format. The value is stored in a uint32_t.

LC_VAR_ADDR
: This type of variable accepts an address in either host or decimal-dot format. The value is stored in a uint32_t. This is not yet implemented. (XXX)

RETURN VALUE

On success 0 is returned, otherwise -1 is returned.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;
        long int counter = -1;

        lc_rv_ret = lc_register_var("Begin", LC_VAR_LONG,
                        &counter, ’c’);
        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                        lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, ’f’);
        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                        lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                        NULL);

        lc_cleanup();
```

```
        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                    %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
        }

        if (counter != -1) {
                printf("Counter was specified as: %ld\n", counter);
        } else {
                printf("Counter was not specified.\n");
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_callback(3), **lc_geterrno**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process_file**(3), **lc_process**(3)

NAME

libconfig − Consistent configuration library.

SYNOPSIS

#include <libconfig.h>

int lc_register_callback(const char *_var_, char _opt_, lc_var_type_t _type_, int (*_callback_)(const char *, const char *, const char *, const char *, lc_flags_t, void *), void *_extra_);

int lc_register_var(const char *_var_, lc_var_type_t _type_, void *_data_, char _opt_);

int lc_process(int _argc_, char **_argv_, const char *_appname_, lc_conf_type_t _type_, const char *_extra_);

lc_err_t lc_geterrno(void);

char *lc_geterrstr(void);

DESCRIPTION

Libconfig is a library to provide easy access to configuration data in a consistent and logical manner. Variables (registered through **lc_register_var**(3) or **lc_register_callback**(3)) are processed with the **lc_process**(3) and **lc_process_file**(3) functions. Errors can be examined through **lc_geterrno**(3) and **lc_geterrstr**(3). Clean-up may be performed using the **lc_cleanup**(3) function.

EXAMPLE

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

int main(int argc, char **argv) {
        int lc_p_ret, lc_rv_ret;
        char *filename = NULL;

        lc_rv_ret = lc_register_var("File", LC_VAR_STRING,
                        &filename, 'f');

        if (lc_rv_ret != 0) {
                fprintf(stderr, "Error registering variable: %i.\n",
                        lc_geterrno());
                return(EXIT_FAILURE);
        }

        lc_p_ret = lc_process(argc, argv, "example", LC_CONF_APACHE,
                        NULL);

        lc_cleanup();

        if (lc_p_ret != 0) {
                fprintf(stderr, "Error processing configuration: \
                        %s\n", lc_geterrstr());
                return(EXIT_FAILURE);
        }

        if (filename != NULL) {
                printf("File specified was: %s\n", filename);
        } else {
                printf("No filename specified.\n");
```

```
        }

        return(EXIT_SUCCESS);
}
```

SEE ALSO

lc_register_var(3), **lc_register_callback**(3), **lc_geterr**(3), **lc_geterrstr**(3), **lc_cleanup**(3), **lc_process**(3), **lc_process_file**(3)