NAME

libnftables − nftables frontend library

SYNOPSIS

#include <nftables/libnftables.h>

struct nft_ctx *nft_ctx_new(uint32_t *flags***);**
void nft_ctx_free(struct nft_ctx **ctx***);**

bool nft_ctx_get_dry_run(struct nft_ctx **ctx***);**
void nft_ctx_set_dry_run(struct nft_ctx **ctx***, bool** *dry***);**

unsigned int nft_ctx_output_get_flags(struct nft_ctx **ctx***);**
void nft_ctx_output_set_flags(struct nft_ctx **ctx***, unsigned int** *flags***);**

unsigned int nft_ctx_output_get_debug(struct nft_ctx **ctx***);**
void nft_ctx_output_set_debug(struct nft_ctx **ctx***, unsigned int** *mask***);**

FILE *nft_ctx_set_output(struct nft_ctx **ctx***, FILE** **fp***);**
int nft_ctx_buffer_output(struct nft_ctx **ctx***);**
int nft_ctx_unbuffer_output(struct nft_ctx **ctx***);**
const char *nft_ctx_get_output_buffer(struct nft_ctx **ctx***);**

FILE *nft_ctx_set_error(struct nft_ctx **ctx***, FILE** **fp***);**
int nft_ctx_buffer_error(struct nft_ctx **ctx***);**
int nft_ctx_unbuffer_error(struct nft_ctx **ctx***);**
const char *nft_ctx_get_error_buffer(struct nft_ctx **ctx***);**

int nft_ctx_add_include_path(struct nft_ctx **ctx***, const char** **path***);**
void nft_ctx_clear_include_paths(struct nft_ctx **ctx***);**

int nft_run_cmd_from_buffer(struct nft_ctx **nft***, const char** **buf***);**
int nft_run_cmd_from_filename(struct nft_ctx **nft***,**
const char **filename***);**

Link with *−lnftables*.

DESCRIPTION

This library was designed with nftables integration into applications in mind. Its API is therefore kept as simple as possible, which somewhat limits its flexibility. Due to support for JSON markup of input and output though, convenience in constructing and parsing of input and output data may be achieved by using a third−party library such as **libjansson**.

At the very basic level, one has to allocate a new object of type **struct nft_ctx** using **nft_ctx_new**() function, then pass commands via **nft_run_cmd_from_buffer**() or **nft_run_cmd_from_filename**() functions. By default, any output is written to **stdout** (or **stderr** for error messages). These file pointers may be changed using **nft_ctx_set_output**() and **nft_ctx_set_error**() functions. On top of that, it is possible to have any output buffered by the library for later retrieval as a static buffer. See **nft_ctx_buffer_output**() and **nft_ctx_buffer_error**() functions for details.

nft_ctx_new() and nft_ctx_free()

These functions aid in nft context management. In order to make use of the library, at least one context object has to be allocated. The context holds temporary data such as caches, library configuration and (if enabled) output and error buffers.

The **nft_ctx_new**() function allocates and returns a new context object. The parameter *flags* is unused at

this point and should be set to zero. For convenience, the macro **NFT_CTX_DEFAULT** is defined to that value.

The **nft_ctx_free**() function frees the context object pointed to by *ctx*, including any caches or buffers it may hold.

nft_ctx_get_dry_run() and nft_ctx_set_dry_run()
Dry−run setting controls whether ruleset changes are actually committed on kernel side or not. It allows to check whether a given operation would succeed without making actual changes to the ruleset. The default setting is **false**.

The **nft_ctx_get_dry_run**() function returns the dry−run setting's value contained in *ctx*.

The **nft_ctx_set_dry_run**() function sets the dry−run setting in *ctx* to the value of *dry*.

nft_ctx_output_get_flags() and nft_ctx_output_set_flags()
The flags setting controls the output format.

```
enum {
        NFT_CTX_OUTPUT_REVERSEDNS     = (1 << 0),
        NFT_CTX_OUTPUT_SERVICE        = (1 << 1),
        NFT_CTX_OUTPUT_STATELESS      = (1 << 2),
        NFT_CTX_OUTPUT_HANDLE         = (1 << 3),
        NFT_CTX_OUTPUT_JSON           = (1 << 4),
        NFT_CTX_OUTPUT_ECHO           = (1 << 5),
        NFT_CTX_OUTPUT_GUID           = (1 << 6),
        NFT_CTX_OUTPUT_NUMERIC_PROTO  = (1 << 7),
        NFT_CTX_OUTPUT_NUMERIC_PRIO   = (1 << 8),
        NFT_CTX_OUTPUT_NUMERIC_SYMBOL = (1 << 9),
        NFT_CTX_OUTPUT_NUMERIC_TIME   = (1 << 10),
        NFT_CTX_OUTPUT_NUMERIC_ALL    = (NFT_CTX_OUTPUT_NUMERIC_PROTO |
                                NFT_CTX_OUTPUT_NUMERIC_PRIO |
                                NFT_CTX_OUTPUT_NUMERIC_SYMBOL |
                                NFT_CTX_OUTPUT_NUMERIC_TIME),
        NFT_CTX_OUTPUT_TERSE          = (1 << 11),
};
```

NFT_CTX_OUTPUT_REVERSEDNS
Reverse DNS lookups are performed for IP addresses when printing. Note that this may add significant delay to **list** commands depending on DNS resolver speed.

NFT_CTX_OUTPUT_SERVICE
Print port numbers as services as described in the /etc/services file.

NFT_CTX_OUTPUT_STATELESS
If stateless output has been requested, then stateful data is not printed. Stateful data refers to those objects that carry run−time data, e.g. the **counter** statement holds packet and byte counter values, making it stateful.

NFT_CTX_OUTPUT_HANDLE
Upon insertion into the ruleset, some elements are assigned a unique handle for identification purposes. For example, when deleting a table or chain, it may be identified either by name or handle. Rules on the other hand must be deleted by handle, because there is no other way to uniquely identify them. This flag makes ruleset listings include handle values.

NFT_CTX_OUTPUT_JSON
If enabled at compile−time, libnftables accepts input in JSON format and is able to print output in JSON format as well. See **libnftables−json**(5) for a description of the supported schema. This flag controls JSON output format, input is auto−detected.

NFT_CTX_OUTPUT_ECHO
The echo setting makes libnftables print the changes once they are committed to the kernel, just like a running instance of **nft monitor** would. Amongst other things, this allows to retrieve an added rule's handle atomically.

NFT_CTX_OUTPUT_GUID
Display UID and GID as described in the /etc/passwd and /etc/group files.

NFT_CTX_OUTPUT_NUMERIC_PROTO
Display layer 4 protocol numerically.

NFT_CTX_OUTPUT_NUMERIC_PRIO
Display base chain priority numerically.

NFT_CTX_OUTPUT_NUMERIC_SYMBOL
Display expression datatype as numeric value.

NFT_CTX_OUTPUT_NUMERIC_TIME
Display time, day and hour values in numeric format.

NFT_CTX_OUTPUT_NUMERIC_ALL
Display all numerically.

NFT_CTX_OUTPUT_TERSE
If terse output has been requested, then the contents of sets are not printed.

The **nft_ctx_output_get_flags**() function returns the output flags setting's value in *ctx*.

The **nft_ctx_output_set_flags**() function sets the output flags setting in *ctx* to the value of *val*.

nft_ctx_output_get_debug() and nft_ctx_output_set_debug()
Libnftables supports separate debugging of different parts of its internals. To facilitate this, debugging output is controlled via a bit mask. The bits are defined as such:

```
enum nft_debug_level {
    NFT_DEBUG_SCANNER       = 0x1,
    NFT_DEBUG_PARSER        = 0x2,
    NFT_DEBUG_EVALUATION    = 0x4,
    NFT_DEBUG_NETLINK       = 0x8,
    NFT_DEBUG_MNL           = 0x10,
    NFT_DEBUG_PROTO_CTX     = 0x20,
    NFT_DEBUG_SEGTREE       = 0x40,
};
```

NFT_DEBUG_SCANNER
Print LEX debug output.

NFT_DEBUG_PARSER
Print YACC debug output.

NFT_DEBUG_EVALUATION
Print debug information about evaluation phase.

NFT_DEBUG_NETLINK
Print netlink debug output.

NFT_DEBUG_MNL
Print libmnl debug output.

NFT_DEBUG_PROTO_CTX
Print protocol context debug output.

NFT_DEBUG_SEGTREE

Print segtree (i.e. interval sets) debug output.

The **nft_ctx_output_get_debug**() function returns the debug output setting's value in *ctx*.

The **nft_ctx_output_set_debug**() function sets the debug output setting in *ctx* to the value of *mask*.

Controlling library standard and error output

By default, any output from the library (e.g., after a **list** command) is written to *stdout* and any error messages are written to *stderr*. To give applications control over them, there are functions to assign custom file pointers as well as having the library buffer what would be written for later retrieval in a static buffer. This buffer is guaranteed to be null−terminated and must not be freed. Note that the retrieval functions rewind the buffer position indicator. Further library output will probably overwrite the buffer content and potentially render it invalid (due to reallocation).

The **nft_ctx_set_output**() and **nft_ctx_set_error**() functions set the output or error file pointer in *ctx* to the value of *fp*. They return the previous value to aid in temporary file pointer overrides. On error, these functions return NULL. This happens only if *fp* is NULL or invalid (tested using **ferror**() function).

The **nft_ctx_buffer_output**() and **nft_ctx_buffer_error**() functions enable library standard or error output buffering. The functions return zero on success, non−zero otherwise. This may happen if the internal call to **fopencookie**() failed.

The **nft_ctx_unbuffer_output**() and **nft_ctx_unbuffer_error**() functions disable library standard or error output buffering. On failure, the functions return non−zero which may only happen if buffering was not enabled at the time the function was called.

The **nft_ctx_get_output_buffer**() and **nft_ctx_get_error_buffer**() functions return a pointer to the buffered output (which may be empty).

nft_ctx_add_include_path() and nft_ctx_clear_include_path()

The **include** command in nftables rulesets allows to outsource parts of the ruleset into a different file. The include path defines where these files are searched for. Libnftables allows to have a list of those paths which are searched in order. The default include path list contains a single compile−time defined entry (typically */etc/*).

The **nft_ctx_add_include_path**() function extends the list of include paths in *ctx* by the one given in *path*. The function returns zero on success or non−zero if memory allocation failed.

The **nft_ctx_clear_include_paths**() function removes all include paths, even the built−in default one.

nft_run_cmd_from_buffer() and nft_run_cmd_from_filename()

These functions perform the actual work of parsing user input into nftables commands and executing them.

The **nft_run_cmd_from_buffer**() function passes the command(s) contained in *buf* (which must be null−terminated) to the library, respecting settings and state in *nft*.

The **nft_run_cmd_from_filename**() function passes the content of *filename* to the library, respecting settings and state in *nft*.

Both functions return zero on success. A non−zero return code indicates an error while parsing or executing the command. This event should be accompanied by an error message written to library error output.

EXAMPLE

```
#include <stdio.h>
#include <string.h>
#include <nftables/libnftables.h>
```

```
int main(void)
{
	char *list_cmd = "list ruleset";
	struct nft_ctx *nft;
	const char *output, *p;
	char buf[256];
	int rc = 0;

	nft = nft_ctx_new(NFT_CTX_DEFAULT);
	if (!nft)
		return 1;

	while (1) {
		if (nft_ctx_buffer_output(nft) ||
		    nft_run_cmd_from_buffer(nft, list_cmd)) {
			rc = 1;
			break;
		}
		output = nft_ctx_get_output_buffer(nft);
		if (strlen(output)) {
			printf("\nThis is the current ruleset:\n| ");
			for (p = output; *(p + 1); p++) {
				if (*p == '\n')
					printf("\n| ");
				else
					putchar(*p);
			}
			putchar('\n');
		} else {
			printf("\nCurrent ruleset is empty.\n");
		}
		nft_ctx_unbuffer_output(nft);

		printf("\nEnter command ('q' to quit): ");
		fflush(stdout);
		fgets(buf, 256, stdin);
		if (strlen(buf))
			buf[strlen(buf) - 1] = '\0';

		if (buf[0] == 'q' && buf[1] == '\0')
			break;

		if (nft_run_cmd_from_buffer(nft, buf)) {
			rc = 1;
			break;
		}
	}

	nft_ctx_free(nft);
	return rc;
}
```

SEE ALSO

libnftables-json(5), **nft**(8)

AUTHOR

Phil Sutter <phil@nwl.cc>
Author.