

**NAME**

**cnvlist\_get**, **cnvlist\_take**, **cnvlist\_free** - API for managing name/value pairs by cookie.

**LIBRARY**

Name/value pairs library (libnv, -lnv)

**SYNOPSIS**

```
#include <sys/cnv.h>
```

*const char \**

```
cnvlist_name(const void *cookie);
```

*int*

```
cnvlist_type(const void *cookie);
```

*bool*

```
cnvlist_get_bool(const void *cookie);
```

*uint64\_t*

```
cnvlist_get_number(const void *cookie);
```

*const char \**

```
cnvlist_get_string(const void *cookie);
```

*const nvlist\_t \**

```
cnvlist_get_nvlist(const void *cookie);
```

*const void \**

```
cnvlist_get_binary(const void *cookie, size_t *sizep);
```

*const bool \**

```
cnvlist_get_bool_array(const void *cookie, size_t *nitemsp);
```

*const uint64\_t \**

```
cnvlist_get_number_array(const void *cookie, size_t *nitemsp);
```

*const char \* const \**

```
cnvlist_get_string_array(const void *cookie, size_t *nitemsp);
```

*const nvlist\_t \* const \**

**cnvlist\_get\_nvlist\_array**(*const void \*cookie, size\_t \*nitemsp*);

*int*

**cnvlist\_get\_descriptor**(*const void \*cookie*);

*const int \**

**cnvlist\_get\_descriptor\_array**(*const void \*cookie, size\_t \*nitemsp*);

*bool*

**cnvlist\_take\_bool**(*void \*cookie*);

*uint64\_t*

**cnvlist\_take\_number**(*void \*cookie*);

*const char \**

**cnvlist\_take\_string**(*void \*cookie*);

*const nvlist\_t \**

**cnvlist\_take\_nvlist**(*void \*cookie*);

*const void \**

**cnvlist\_take\_binary**(*void \*cookie, size\_t \*sizep*);

*const bool \**

**cnvlist\_take\_bool\_array**(*void \*cookie, size\_t \*nitemsp*);

*const uint64\_t \**

**cnvlist\_take\_number\_array**(*void \*cookie, size\_t \*nitemsp*);

*const char \* const \**

**cnvlist\_take\_string\_array**(*void \*cookie, size\_t \*nitemsp*);

*const nvlist\_t \* const \**

**cnvlist\_take\_nvlist\_array**(*void \*cookie, size\_t \*nitemsp*);

*int*

**cnvlist\_take\_descriptor**(*void \*cookie*);

*const int \**

**cnvlist\_take\_descriptor\_array**(*void \*cookie, size\_t \*nitemsp*);

*void*

**cnvlist\_free\_null**(*void \*cookie*);

*void*

**cnvlist\_free\_bool**(*void \*cookie*);

*void*

**cnvlist\_free\_number**(*void \*cookie*);

*void*

**cnvlist\_free\_string**(*void \*cookie*);

*void*

**cnvlist\_free\_nvlist**(*void \*cookie*);

*void*

**cnvlist\_free\_descriptor**(*void \*cookie*);

*void*

**cnvlist\_free\_binary**(*void \*cookie*);

*void*

**cnvlist\_free\_bool\_array**(*void \*cookie*);

*void*

**cnvlist\_free\_number\_array**(*void \*cookie*);

*void*

**cnvlist\_free\_string\_array**(*void \*cookie*);

*void*

**cnvlist\_free\_nvlist\_array**(*void \*cookie*);

*void*

**cnvlist\_free\_descriptor\_array**(*void \*cookie*);

## DESCRIPTION

The **libnv** library permits easy management of name/value pairs and can send and receive them over sockets. For more information, also see [nv\(9\)](#).

The concept of cookies is explained in **nvlist\_next()**, **nvlist\_get\_parent()**, and **nvlist\_get\_pararr()** from **nv(9)**.

The **cnvlist\_name()** function returns the name of an element associated with the given cookie.

The **cnvlist\_type()** function returns the type of an element associated with the given cookie. Types which can be returned are described in **nv(9)**.

The **cnvlist\_get** family of functions obtains the value associated with the given cookie. Returned strings, nvlists, descriptors, binaries, or arrays must not be modified by the user, since they still belong to the nvlist. The nvlist must not be in an error state.

The **cnvlist\_take** family of functions returns the value associated with the given cookie and removes the element from the nvlist. When the value is a string, binary, or array value, the caller is responsible for freeing the returned memory with **free(3)**. When the value is an nvlist, the caller is responsible for destroying the returned nvlist with **nvlist\_destroy()**. When the value is a descriptor, the caller is responsible for closing the returned descriptor with the **close(2)**.

The **cnvlist\_free** family of functions removes an element of the supplied cookie and frees all resources. If an element of the given cookie has the wrong type or does not exist, the program is aborted.

## EXAMPLES

The following example demonstrates how to deal with **cnvlist** API.

```
int type;
void *cookie, *scookie, *bcookie;
nvlist_t *nvl;
char *name;

nvl = nvlist_create(0);
nvlist_add_bool(nvl, "test", 1 == 2);
nvlist_add_string(nvl, "test2", "cnvlist");
cookie = NULL;

while (nvlist_next(nvl, &type, &cookie) != NULL) {
    switch (type) {
        case NV_TYPE_BOOL:
            printf("test: %d\n", cnvlist_get_bool(cookie));
            bcookie = cookie;
            break;
    }
}
```

```
    case NV_TYPE_STRING:
        printf("test2: %s\n", cnvlist_get_string(cookie));
        scookie = cookie;
        break;
    }
}
```

```
name = cnvlist_take_string(scookie);
cnvlist_free_bool(bcookie);
```

```
printf("test2: %s\n", name);
free(name);
```

```
printf("nvlist_empty = %d\n", nvlist_empty(nvl));
nvlist_destroy(nvl);
```

```
return (0);
```

## SEE ALSO

close(2), free(3), nv(9)

## AUTHORS

The **cnv** API was created during the Google Summer Of Code 2016 by Adam Starak.