

**NAME**

**cap\_init, cap\_wrap, cap\_unwrap, cap\_sock, cap\_clone, cap\_close, cap\_limit\_get, cap\_limit\_set, cap\_send\_nvlist, cap\_recv\_nvlist, cap\_xfer\_nvlist, cap\_service\_open** - library for handling application capabilities

**LIBRARY**

Casper Library (libcasper, -lcasper)

**SYNOPSIS**

```
#define WITH_CASPER
```

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

```
#include <libcasper.h>
```

```
cap_channel_t *
```

```
cap_init(void);
```

```
cap_channel_t *
```

```
cap_wrap(int sock, int flags);
```

```
int
```

```
cap_unwrap(cap_channel_t *chan, int *flags);
```

```
int
```

```
cap_sock(const cap_channel_t *chan);
```

```
cap_channel_t *
```

```
cap_clone(const cap_channel_t *chan);
```

```
void
```

```
cap_close(cap_channel_t *chan);
```

```
int
```

```
cap_limit_get(const cap_channel_t *chan, nvlist_t **limitsp);
```

```
int
```

```
cap_limit_set(const cap_channel_t *chan, nvlist_t *limits);
```

```
int
```

```
cap_send_nvlist(const cap_channel_t *chan, const nvlist_t *nvl);
```

```

nvlst_t *
cap_recv_nvlst(const cap_channel_t *chan);

nvlst_t *
cap_xfer_nvlst(const cap_channel_t *chan, nvlst_t *nvl);

cap_channel_t *
cap_service_open(const cap_channel_t *chan, const char *name);

```

## DESCRIPTION

The **libcasper** library provides for the control of application capabilities through the casper process.

An application capability, represented by the *cap\_channel\_t* type, is a communication channel between the caller and the casper daemon or an instance of one of the daemon's services. A capability to the casper process, obtained with the **cap\_init()** function, allows a program to create capabilities to access the casper daemon's services via the **cap\_service\_open()** function.

The **cap\_init()** function instantiates a capability to allow a program to access the casper daemon. It must be called from a single-threaded context.

The **cap\_wrap()** function creates a *cap\_channel\_t* based on the socket supplied in the call. The function is used when a capability is inherited through the `execve(2)` system call, or sent over a `unix(4)` domain socket as a file descriptor, and has to be converted into a *cap\_channel\_t*. The *flags* argument defines the channel behavior. The supported flags are:

### CASPER\_NO\_UNIQ

The communication between the process and the casper daemon uses no unique version of nvlst.

The **cap\_unwrap()** function returns the `unix(4)` domain socket used by the daemon service, and frees the *cap\_channel\_t* structure.

The **cap\_clone()** function returns a clone of the capability passed as its only argument.

The **cap\_close()** function closes, and frees, the given capability.

The **cap\_sock()** function returns the `unix(4)` domain socket descriptor associated with the given capability for use with system calls such as: `kevent(2)`, `poll(2)`, and `select(2)`.

The **cap\_limit\_get()** function stores the current limits of the given capability in the *limitsp* argument. If the function returns 0 and NULL is stored in the *limitsp* argument, there are no limits set.

The **cap\_limit\_set()** function sets limits for the given capability. The limits are provided as an `nvlist(9)`. The exact format of the limits depends on the service that the capability represents. **cap\_limit\_set()** frees the limits passed to the call, whether or not the operation succeeds or fails.

The **cap\_send\_nvlist()** function sends the given `nvlist(9)` over the given capability. This is a low level interface to communicate with casper services. It is expected that most services will provide a higher level API.

The **cap\_recv\_nvlist()** function receives the given `nvlist(9)` over the given capability.

The **cap\_xfer\_nvlist()** function sends the given `nvlist(9)`, destroys it, and receives a new `nvlist(9)` in response over the given capability. It does not matter if the function succeeds or fails, the `nvlist(9)` given for sending will always be destroyed before the function returns.

The **cap\_service\_open()** function opens the casper service named in the call using the casper capability obtained via the **cap\_init()** function. The **cap\_service\_open()** function returns a capability that provides access to the opened service. Casper supports the following services in the base system:

system.dns	provides libc compatible DNS API
system.fileargs	provides an API for opening files specified on a command line
system.grp	provides a <code>getgrent(3)</code> compatible API
system.net	provides a libc compatible network API
system.netdb	provides libc compatible network proto API
system.pwd	provides a <code>getpwent(3)</code> compatible API
system.sysctl	provides a <code>sysctlbyname(3)</code> compatible API
system.syslog	provides a <code>syslog(3)</code> compatible API

## RETURN VALUES

The **cap\_clone()**, **cap\_init()**, **cap\_recv\_nvlist()**, **cap\_service\_open()**, **cap\_wrap()** and **cap\_xfer\_nvlist()** functions return `NULL` and set the *errno* variable on failure.

The **cap\_limit\_get()**, **cap\_limit\_set()** and **cap\_send\_nvlist()** functions return `-1` and set the *errno* variable on failure.

The **cap\_close()**, **cap\_sock()** and **cap\_unwrap()** functions always succeed.

## SEE ALSO

`errno(2)`, `execve(2)`, `kevent(2)`, `poll(2)`, `select(2)`, `cap_dns(3)`, `cap_fileargs(3)`, `cap_grp(3)`, `cap_net(3)`, `cap_netdb(3)`, `cap_pwd(3)`, `cap_sysctl(3)`, `cap_syslog(3)`, `libcasper_service(3)`, `capsicum(4)`, `unix(4)`, `nv(9)`

**HISTORY**

The **libcasper** library first appeared in FreeBSD 10.3.

**AUTHORS**

The **libcasper** library was implemented by Pawel Jakub Dawidek <*pawel@dawidek.net*> under sponsorship from the FreeBSD Foundation. The **libcasper** new architecture was implemented by Mariusz Zaborski <*oshogbo@FreeBSD.org*>