NAME

cap_sysctl - library for getting or setting system information in capability mode

LIBRARY

library "libcap_sysctl"

SYNOPSIS

#include <libcasper.h>
#include <casper/cap_sysctl.h>

int
cap_sysctl(*cap_channel_t *chan, const int *name, u_int namelen, void *oldp, size_t *oldlenp, const void *newp, size_t newlen*);

int
cap_sysctlbyname(*cap_channel_t *chan, const char *name, void *oldp, size_t *oldlenp, const void *newp, size_t newlen*);

int
cap_sysctlnametomib(*cap_channel_t *chan, const char *name, int *mibp, size_t *sizep*);

*cap_sysctl_limit_t **
cap_sysctl_limit_init(*cap_channel_t *chan*);

*cap_sysctl_limit_t **
cap_sysctl_limit_name(*cap_sysctl_limit_t *limit, const char *name, int flags*);

*cap_sysctl_limit_t **
cap_sysctl_limit_mib(*cap_sysctl_limit_t *limit, const int *mibp, u_int miblen, int flags*);

int
cap_sysctl_limit(*cap_sysctl_limit_t *limit*);

DESCRIPTION

The **cap_sysctl**(), **cap_sysctlbyname**() and **cap_sysctlnametomib**() functions are equivalent to sysctl(3), sysctlbyname(3) and sysctlnametomib(3), except that they are implemented by the ‘system.sysctl’ libcasper(3) service and require a corresponding libcasper(3) capability.

All of these functions, with the exceptions of **cap_sysctl_limit_init**() and **cap_sysctl_limit_mib**(), are reentrant but not thread-safe. That is, they may be called from separate threads only with different

cap_channel_t arguments or with synchronization.

LIMITS

By default, the **cap_sysctl** capability provides unrestricted access to the sysctl namespace. Applications typically only require access to a small number of sysctl variables; the **cap_sysctl_limit**() interface can be used to restrict the sysctls that can be accessed using the **cap_sysctl** capability.
cap_sysctl_limit_init() returns an opaque limit handle used to store a list of permitted sysctls and access rights. Rights are encoded using the following flags:

CAP_SYSCTL_READ	allow reads of the sysctl variable
CAP_SYSCTL_WRITE	allow writes of the sysctl variable
CAP_SYSCTL_RDWR	allow reads and writes of the sysctl variable
CAP_RECURSIVE	permit access to any child of the sysctl variable

The **cap_sysctl_limit_name**() function adds the sysctl identified by *name* to the limit list, and **cap_sysctl_limit_mib**() function adds the sysctl identified by *mibp* to the limit list. The access rights for the sysctl are specified in the *flags* parameter; at least one of CAP_SYSCTL_READ, CAP_SYSCTL_WRITE and CAP_SYSCTL_RDWR must be specified. **cap_sysctl_limit**() applies a set of sysctl limits to the capability, denying access to sysctl variables not belonging to the set. It consumes the limit handle. After either success or failure, the user must not access the handle again.

Once a set of limits is applied, subsequent calls to **cap_sysctl_limit**() will fail unless the new set is a subset of the current set.

cap_sysctlnametomib() will succeed so long as the named sysctl variable is present in the limit set, regardless of its access rights. When a sysctl variable name is added to a limit set, its MIB identifier is automatically added to the set.

EXAMPLES

The following example first opens a capability to casper, uses this capability to create the **system.sysctl** casper service, and then uses the **cap_sysctl** capability to get the value of kern.trap_enotcap.

```
cap_channel_t *capcas, *capsysctl;
const char *name = "kern.trap_enotcap";
void *limit;
size_t size;
bool value;

/* Open capability to Casper. */
capcas = cap_init();
```

```
if (capcas == NULL)
        err(1, "Unable to contact Casper");

/* Enter capability mode sandbox. */
if (cap_enter() < 0 && errno != ENOSYS)
        err(1, "Unable to enter capability mode");

/* Use Casper capability to create capability to the system.sysctl service. */
capsysctl = cap_service_open(capcas, "system.sysctl");
if (capsysctl == NULL)
        err(1, "Unable to open system.sysctl service");

/* Close Casper capability, we don't need it anymore. */
cap_close(capcas);

/* Create limit for one MIB with read access only. */
limit = cap_sysctl_limit_init(capsysctl);
(void)cap_sysctl_limit_name(limit, name, CAP_SYSCTL_READ);

/* Limit system.sysctl. */
if (cap_sysctl_limit(limit) < 0)
        err(1, "Unable to set limits");

/* Fetch value. */
size = sizeof(value);
if (cap_sysctlbyname(capsysctl, name, &value, &size, NULL, 0) < 0)
        err(1, "Unable to get value of sysctl");

printf("The value of %s is %d.\n", name, value);

cap_close(capsysctl);
```

RETURN VALUES

cap_sysctl_limit_init() will return a new limit handle on success or NULL on failure, and set *errno*. **cap_sysctl_limit_mib**() and **cap_sysctl_limit_name**() will return the modified limit handle on success or NULL on failure and set *errno*. After failure, the caller must not access the limit handle again. **cap_sysctl_limit**() will return -1 on failure and set *errno*. **cap_sysctl**(), **cap_sysctlbyname**(), and **cap_sysctlnametomib**() have the same return values as their non-capability-mode equivalents as documented in sysctl(3).

SEE ALSO

cap_enter(2), err(3), sysctl(3), sysctlbyname(3), sysctlnametomib(3), capsicum(4), nv(9)

HISTORY

The **cap_sysctl** service first appeared in FreeBSD 10.3.

AUTHORS

The **cap_sysctl** service was implemented by Pawel Jakub Dawidek <*pawel@dawidek.net*> under sponsorship from the FreeBSD Foundation.

This manual page was written by
Mariusz Zaborski <*oshogbo@FreeBSD.org*>.