NAME

sysctl_add_oid, **sysctl_move_oid**, **sysctl_remove_oid**, **sysctl_remove_name** - runtime sysctl tree manipulation

SYNOPSIS

#include <sys/types.h>
#include <sys/sysctl.h>

*struct sysctl_oid **
sysctl_add_oid(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number, const char *name, int kind, void *arg1, intmax_t arg2, int (*handler) (SYSCTL_HANDLER_ARGS), const char *format, const char *descr, const char *label*);

int
sysctl_move_oid(*struct sysctl_oid *oidp, struct sysctl_oid_list *parent*);

int
sysctl_remove_oid(*struct sysctl_oid *oidp, int del, int recurse*);

int
sysctl_remove_name(*struct sysctl_oid *oidp, const char *name, int del, int recurse*);

DESCRIPTION

These functions provide the interface for creating and deleting sysctl OIDs at runtime for example during the lifetime of a module. The wrapper macros defined by sysctl(9) are recommended when creating new OIDs. **sysctl_add_oid**() should not be called directly from the code.

Dynamic OIDs of type CTLTYPE_NODE are reusable so that several code sections can create and delete them, but in reality they are allocated and freed based on their reference count. As a consequence, it is possible for two or more code sections to create partially overlapping trees that they both can use. It is not possible to create overlapping leaves, nor to create different child types with the same name and parent.

The **sysctl_add_oid**() function creates a raw OID of any type and connects it to its parent node, if any. If the OID is successfully created, the function returns a pointer to it else it returns NULL. Many of the arguments for **sysctl_add_oid**() are common to the wrapper macros defined by sysctl(9).

The **sysctl_move_oid**() function reparents an existing OID. The OID is assigned a new number as if it had been created with *number* set to OID_AUTO.

The **sysctl_remove_oid**() function removes a dynamically created OID from the tree and optionally freeing its resources. It takes the following arguments:

oidp A pointer to the dynamic OID to be removed. If the OID is not dynamic, or the pointer is NULL, the function returns EINVAL.

del If non-zero, **sysctl_remove_oid**() will try to free the OID's resources when the reference count of the OID becomes zero. However, if *del* is set to 0, the routine will only deregister the OID from the tree, without freeing its resources. This behaviour is useful when the caller expects to rollback (possibly partially failed) deletion of many OIDs later.

recurse If non-zero, attempt to remove the node and all its children. If *recurse* is set to 0, any attempt to remove a node that contains any children will result in a ENOTEMPTY error. *WARNING*: *use recursive deletion with extreme caution*! Normally it should not be needed if contexts are used. Contexts take care of tracking inter-dependencies between users of the tree. However, in some extreme cases it might be necessary to remove part of the subtree no matter how it was created, in order to free some other resources. Be aware, though, that this may result in a system panic(9) if other code sections continue to use removed subtrees.

The **sysctl_remove_name**() function looks up the child node matching the *name* argument and then invokes the **sysctl_remove_oid**() function on that node, passing along the *del* and *recurse* arguments. If a node having the specified name does not exist an error code of ENOENT is returned. Else the error code from **sysctl_remove_oid**() is returned.

In most cases the programmer should use contexts, as described in sysctl_ctx_init(9), to keep track of created OIDs, and to delete them later in orderly fashion.

SEE ALSO

sysctl(8), sysctl(9), sysctl_ctx_free(9), sysctl_ctx_init(9)

HISTORY

These functions first appeared in FreeBSD 4.2.

AUTHORS

Andrzej Bialecki <*abial@FreeBSD.org*>

BUGS

Sharing nodes between many code sections causes interdependencies that sometimes may lock the resources. For example, if module A hooks up a subtree to an OID created by module B, module B will be unable to delete that OID. These issues are handled properly by sysctl contexts.

Many operations on the tree involve traversing linked lists. For this reason, OID creation and removal is relatively costly.