NAME

SYSCTL_DECL, **SYSCTL_ADD_BOOL**, **SYSCTL_ADD_COUNTER_U64**, **SYSCTL_ADD_COUNTER_U64_ARRAY**, **SYSCTL_ADD_INT**, **SYSCTL_ADD_LONG**, **SYSCTL_ADD_NODE**, **SYSCTL_ADD_NODE_WITH_LABEL**, **SYSCTL_ADD_OPAQUE**, **SYSCTL_ADD_PROC**, **SYSCTL_ADD_QUAD**, **SYSCTL_ADD_ROOT_NODE**, **SYSCTL_ADD_S8**, **SYSCTL_ADD_S16**, **SYSCTL_ADD_S32**, **SYSCTL_ADD_S64**, **SYSCTL_ADD_SBINTIME_MSEC**, **SYSCTL_ADD_SBINTIME_USEC**, **SYSCTL_ADD_STRING**, **SYSCTL_ADD_CONST_STRING**, **SYSCTL_ADD_STRUCT**, **SYSCTL_ADD_TIMEVAL_SEC**, **SYSCTL_ADD_U8**, **SYSCTL_ADD_U16**, **SYSCTL_ADD_U32**, **SYSCTL_ADD_U64**, **SYSCTL_ADD_UAUTO**, **SYSCTL_ADD_UINT**, **SYSCTL_ADD_ULONG**, **SYSCTL_ADD_UMA_CUR**, **SYSCTL_ADD_UMA_MAX**, **SYSCTL_ADD_UQUAD**, **SYSCTL_CHILDREN**, **SYSCTL_STATIC_CHILDREN**, **SYSCTL_NODE_CHILDREN**, **SYSCTL_PARENT**, **SYSCTL_BOOL**, **SYSCTL_COUNTER_U64**, **SYSCTL_COUNTER_U64_ARRAY**, **SYSCTL_INT**, **SYSCTL_INT_WITH_LABEL**, **SYSCTL_LONG**, **sysctl_msec_to_ticks**, **SYSCTL_NODE**, **SYSCTL_NODE_WITH_LABEL**, **SYSCTL_OPAQUE**, **SYSCTL_PROC**, **SYSCTL_QUAD**, **SYSCTL_ROOT_NODE**, **SYSCTL_S8**, **SYSCTL_S16**, **SYSCTL_S32**, **SYSCTL_S64**, **SYSCTL_SBINTIME_MSEC**, **SYSCTL_SBINTIME_USEC**, **SYSCTL_STRING**, **SYSCTL_CONST_STRING**, **SYSCTL_STRUCT**, **SYSCTL_TIMEVAL_SEC**, **SYSCTL_U8**, **SYSCTL_U16**, **SYSCTL_U32**, **SYSCTL_U64**, **SYSCTL_UINT**, **SYSCTL_ULONG**, **SYSCTL_UMA_CUR**, **SYSCTL_UMA_MAX**, **SYSCTL_UQUAD** - Dynamic and static sysctl MIB creation functions

SYNOPSIS

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

SYSCTL_DECL(*name*);

*struct sysctl_oid **
SYSCTL_ADD_BOOL(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number, const char *name, int ctlflags, bool *ptr, uint8_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_COUNTER_U64(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number, const char *name, int ctlflags, counter_u64_t *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_COUNTER_U64_ARRAY(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number, const char *name, int ctlflags, counter_u64_t *ptr, intmax_t len, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_INT(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, int *ptr, int val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_LONG(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, long *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_NODE(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, int (*handler)(SYSCTL_HANDLER_ARGS), const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_NODE_WITH_LABEL(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent,*
*int number, const char *name, int ctlflags, int (*handler)(SYSCTL_HANDLER_ARGS),*
*const char *descr, const char *label*);

*struct sysctl_oid **
SYSCTL_ADD_OPAQUE(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, void *ptr, intptr_t len, const char *format, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_PROC(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, void *arg1, intptr_t arg2,*
*int (*handler) (SYSCTL_HANDLER_ARGS), const char *format, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_QUAD(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, int64_t *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_ROOT_NODE(*struct sysctl_ctx_list *ctx, int number, const char *name, int ctlflags,*
*int (*handler)(SYSCTL_HANDLER_ARGS), const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_S8(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*
*const char *name, int ctlflags, int8_t *ptr, int8_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_S16(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,*

*const char *name, int ctlflags, int16_t *ptr, int16_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_S32(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, int32_t *ptr, int32_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_S64(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, int64_t *ptr, int64_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_SBINTIME_MSEC(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent,
*int number, const char *name, int ctlflags, sbintime_t *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_SBINTIME_USEC(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent,
*int number, const char *name, int ctlflags, sbintime_t *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_STRING(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, char *ptr, intptr_t len, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_CONST_STRING(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, const char *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_STRUCT(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, void *ptr, struct_type, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_TIMEVAL_SEC(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, struct timeval *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_U8(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uint8_t *ptr, uint8_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_U16(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,

*const char *name, int ctlflags, uint16_t *ptr, uint16_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_U32(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uint32_t *ptr, uint32_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_U64(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uint64_t *ptr, uint64_t val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_UINT(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, unsigned int *ptr, unsigned int val, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_ULONG(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, unsigned long *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_UQUAD(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uint64_t *ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_UMA_CUR(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uma_zone_t ptr, const char *descr*);

*struct sysctl_oid **
SYSCTL_ADD_UMA_MAX(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, uma_zone_t ptr, const char *descr*);
*const char *descr*
*struct sysctl_oid **
SYSCTL_ADD_UAUTO(*struct sysctl_ctx_list *ctx, struct sysctl_oid_list *parent, int number,
*const char *name, int ctlflags, void *ptr, const char *descr*);

*struct sysctl_oid_list **
SYSCTL_CHILDREN(*struct sysctl_oid *oidp*);

*struct sysctl_oid_list **
SYSCTL_STATIC_CHILDREN(*struct sysctl_oid_list OID_NAME*);

*struct sysctl_oid_list **
SYSCTL_NODE_CHILDREN(*parent*, *name*);

*struct sysctl_oid **
SYSCTL_PARENT(*struct sysctl_oid *oid*);

SYSCTL_BOOL(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_COUNTER_U64(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

SYSCTL_COUNTER_U64_ARRAY(*parent*, *number*, *name*, *ctlflags*, *ptr*, *len*, *descr*);

SYSCTL_INT(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_INT_WITH_LABEL(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*, *label*);

SYSCTL_LONG(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

int
sysctl_msec_to_ticks(*SYSCTL_HANDLER_ARGS*);

SYSCTL_NODE(*parent*, *number*, *name*, *ctlflags*, *handler*, *descr*);

SYSCTL_NODE_WITH_LABEL(*parent*, *number*, *name*, *ctlflags*, *handler*, *descr*, *label*);

SYSCTL_OPAQUE(*parent*, *number*, *name*, *ctlflags*, *ptr*, *len*, *format*, *descr*);

SYSCTL_PROC(*parent*, *number*, *name*, *ctlflags*, *arg1*, *arg2*, *handler*, *format*, *descr*);

SYSCTL_QUAD(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_ROOT_NODE(*number*, *name*, *ctlflags*, *handler*, *descr*);

SYSCTL_S8(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_S16(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_S32(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_S64(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_SBINTIME_MSEC(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

SYSCTL_SBINTIME_USEC(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

SYSCTL_STRING(*parent*, *number*, *name*, *ctlflags*, *arg*, *len*, *descr*);

SYSCTL_CONST_STRING(*parent*, *number*, *name*, *ctlflags*, *arg*, *descr*);

SYSCTL_STRUCT(*parent*, *number*, *name*, *ctlflags*, *ptr*, *struct_type*, *descr*);

SYSCTL_TIMEVAL_SEC(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

SYSCTL_U8(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_U16(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_U32(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_U64(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_UINT(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_ULONG(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_UQUAD(*parent*, *number*, *name*, *ctlflags*, *ptr*, *val*, *descr*);

SYSCTL_UMA_MAX(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

SYSCTL_UMA_CUR(*parent*, *number*, *name*, *ctlflags*, *ptr*, *descr*);

DESCRIPTION

The **SYSCTL** kernel interface allows dynamic or static creation of sysctl(8) MIB entries. All static sysctls are automatically destroyed when the module which they are part of is unloaded. Most top level categories are created statically and are available to all kernel code and its modules.

DESCRIPTION OF ARGUMENTS

ctx Pointer to sysctl context or NULL, if no context. See sysctl_ctx_init(9) for how to create a new sysctl context. Programmers are strongly advised to use contexts to organize the dynamic OIDs which they create because when a context is destroyed all belonging sysctls are destroyed as well. This makes the sysctl cleanup code much simpler. Else deletion of all created OIDs is

required at module unload.

parent A pointer to a struct sysctl_oid_list, which is the head of the parent's list of children. This pointer is retrieved using the **SYSCTL_STATIC_CHILDREN**() macro for static sysctls and the **SYSCTL_CHILDREN**() macro for dynamic sysctls. The **SYSCTL_PARENT**() macro can be used to get the parent of an OID. The macro returns NULL if there is no parent.

number
The OID number that will be assigned to this OID. In almost all cases this should be set to OID_AUTO, which will result in the assignment of the next available OID number.

name The name of the OID. The newly created OID will contain a copy of the name.

ctlflags A bit mask of sysctl control flags. See the section below describing all the control flags.

arg1 First callback argument for procedure sysctls.

arg2 Second callback argument for procedure sysctls.

len The length of the data pointed to by the *ptr* argument. For string type OIDs a length of zero means that strlen(3) will be used to get the length of the string at each access to the OID. For array type OIDs the length must be greater than zero.

ptr Pointer to sysctl variable or string data. For sysctl values the pointer can be SYSCTL_NULL_XXX_PTR which means the OID is read-only and the returned value should be taken from the *val* argument.

val If the *ptr* argument is SYSCTL_NULL_XXX_PTR, gives the constant value returned by this OID. Else this argument is not used.

struct_type
Name of structure type.

handler
A pointer to the function that is responsible for handling read and write requests to this OID. There are several standard handlers that support operations on nodes, integers, strings and opaque objects. It is possible to define custom handlers using the **SYSCTL_PROC**() macro or the **SYSCTL_ADD_PROC**() function.

format A pointer to a string which specifies the format of the OID in a symbolic way. This format is

used as a hint by sysctl(8) to apply proper data formatting for display purposes.

Current formats:

N node
A char *
C int8_t
CU uint8_t
I int
IK[*n*] temperature in Kelvin, multiplied by an optional single digit power of ten scaling factor: 1 (default) gives deciKelvin, 0 gives Kelvin, 3 gives milliKelvin
IU unsigned int
L long
LU unsigned long
Q quad_t
QU u_quad_t
S int16_t
SU uint16_t
S,TYPE struct TYPE structures

descr A pointer to a textual description of the OID.

label A pointer to an aggregation label for this component of the OID. To make it easier to export sysctl data to monitoring systems that support aggregations through labels (e.g., Prometheus), this argument can be used to attach a label name to an OID. The label acts as a hint that this component's name should not be part of the metric's name, but attached to the metric as a label instead.

Labels should only be applied to siblings that are structurally similar and encode the same type of value, as aggregation is of no use otherwise.

NODE VALUE TYPES

Most of the macros and functions used to create sysctl nodes export a read-only constant or in-kernel variable whose type matches the type of the node's value. For example, **SYSCTL_INT**() reports the raw value of an associated variable of type *int*. However, nodes may also export a value that is a translatation of an internal representation.

The **sysctl_msec_to_ticks**() handler can be used with **SYSCTL_PROC**() or **SYSCTL_ADD_PROC**() to export a millisecond time interval. When using this handler, the *arg2* parameter points to an in-kernel variable of type *int* which stores a tick count suitable for use with functions like tsleep(9). The **sysctl_msec_to_ticks**() function converts this value to milliseconds when reporting the node's value.

Similarly, **sysctl_msec_to_ticks**() accepts new values in milliseconds and stores an equivalent value in ticks to **arg2*. Note that new code should use kernel variables of type *sbintime_t* instead of tick counts.

The **SYSCTL_ADD_SBINTIME_MSEC**() and **SYSCTL_ADD_SBINTIME_USEC**() functions and **SYSCTL_SBINTIME_MSEC**() and **SYSCTL_SBINTIME_USEC**() macros all create nodes which export an in-kernel variable of type *sbintime_t*. These nodes do not export the raw value of the associated variable. Instead, they export a 64-bit integer containing a count of either milliseconds (the MSEC variants) or microseconds (the USEC variants).

The **SYSCTL_ADD_TIMEVAL_SEC**() function and **SYSCTL_TIMEVAL_SEC**() macro create nodes which export an in-kernel variable of type *struct timeval*. These nodes do not export full value of the associated structure. Instead, they export a count in seconds as a simple integer which is stored in the *tv_sec* field of the associated variable. This function and macro are intended to be used with variables which store a non-negative interval rather than an absolute time. As a result, they reject attempts to store negative values.

CREATING ROOT NODES

Sysctl MIBs or OIDs are created in a hierarchical tree. The nodes at the bottom of the tree are called root nodes, and have no parent OID. To create bottom tree nodes the **SYSCTL_ROOT_NODE**() macro or the **SYSCTL_ADD_ROOT_NODE**() function needs to be used. By default all static sysctl node OIDs are global and need a **SYSCTL_DECL**() statement prior to their **SYSCTL_NODE**() definition statement, typically in a so-called header file.

CREATING SYSCTL STRINGS

Zero terminated character strings sysctls are created either using the **SYSCTL_STRING**() macro or the **SYSCTL_ADD_STRING**() function. If the *len* argument in zero, the string length is computed at every access to the OID using strlen(3). Use the **SYSCTL_CONST_STRING**() macro or the **SYSCTL_ADD_CONST_STRING**() function to add a sysctl for a constant string.

CREATING OPAQUE SYSCTLS

The **SYSCTL_OPAQUE**() or **SYSCTL_STRUCT**() macros or the **SYSCTL_ADD_OPAQUE**() or **SYSCTL_ADD_STRUCT**() functions create an OID that handle any chunk of data of the size specified by the *len* argument and data pointed to by the *ptr* argument. When using the structure version the type is encoded as part of the created sysctl.

CREATING CUSTOM SYSCTLS

The **SYSCTL_PROC**() macro and the **SYSCTL_ADD_PROC**() function create OIDs with the specified *handler* function. The handler is responsible for handling all read and write requests to the OID. This OID type is especially useful if the kernel data is not easily accessible, or needs to be processed before exporting.

CREATING A STATIC SYSCTL

Static sysctls are declared using one of the **SYSCTL_BOOL**(), **SYSCTL_COUNTER_U64**(), **SYSCTL_COUNTER_U64_ARRAY**(), **SYSCTL_INT**(), **SYSCTL_INT_WITH_LABEL**(), **SYSCTL_LONG**(), **SYSCTL_NODE**(), **SYSCTL_NODE_WITH_LABEL**(), **SYSCTL_OPAQUE**(), **SYSCTL_PROC**(), **SYSCTL_QUAD**(), **SYSCTL_ROOT_NODE**(), **SYSCTL_S8**(), **SYSCTL_S16**(), **SYSCTL_S32**(), **SYSCTL_S64**(), **SYSCTL_SBINTIME_MSEC**(), **SYSCTL_SBINTIME_USEC**(), **SYSCTL_STRING**(), **SYSCTL_CONST_STRING**(), **SYSCTL_STRUCT**(), **SYSCTL_TIMEVAL_SEC**(), **SYSCTL_U8**(), **SYSCTL_U16**(), **SYSCTL_U32**(), **SYSCTL_U64**(), **SYSCTL_UINT**(), **SYSCTL_ULONG**(), **SYSCTL_UQUAD**(), **SYSCTL_UMA_CUR**() or **SYSCTL_UMA_MAX**() macros.

CREATING A DYNAMIC SYSCTL

Dynamic nodes are created using one of the **SYSCTL_ADD_BOOL**(), **SYSCTL_ADD_COUNTER_U64**(), **SYSCTL_ADD_COUNTER_U64_ARRAY**(), **SYSCTL_ADD_INT**(), **SYSCTL_ADD_LONG**(), **SYSCTL_ADD_NODE**(), **SYSCTL_ADD_NODE_WITH_LABEL**(), **SYSCTL_ADD_OPAQUE**(), **SYSCTL_ADD_PROC**(), **SYSCTL_ADD_QUAD**(), **SYSCTL_ADD_ROOT_NODE**(), **SYSCTL_ADD_S8**(), **SYSCTL_ADD_S16**(), **SYSCTL_ADD_S32**(), **SYSCTL_ADD_S64**(), **SYSCTL_ADD_SBINTIME_MSEC**(), **SYSCTL_ADD_SBINTIME_USEC**(), **SYSCTL_ADD_STRING**(), **SYSCTL_ADD_CONST_STRING**(), **SYSCTL_ADD_STRUCT**(), **SYSCTL_ADD_TIMEVAL_SEC**(), **SYSCTL_ADD_U8**(), **SYSCTL_ADD_U16**(), **SYSCTL_ADD_U32**(), **SYSCTL_ADD_U64**(), **SYSCTL_ADD_UAUTO**(), **SYSCTL_ADD_UINT**(), **SYSCTL_ADD_ULONG**(), **SYSCTL_ADD_UQUAD**(), **SYSCTL_ADD_UMA_CUR**() or **SYSCTL_ADD_UMA_MAX**() functions. See sysctl_remove_oid(9) or sysctl_ctx_free(9) for more information on how to destroy a dynamically created OID.

CONTROL FLAGS

For most of the above functions and macros, declaring a type as part of the access flags is not necessary -- however, when declaring a sysctl implemented by a function, including a type in the access mask is required:

CTLTYPE_NODE This is a node intended to be a parent for other nodes.

CTLTYPE_INT This is a signed integer.

CTLTYPE_STRING This is a nul-terminated string stored in a character array.

CTLTYPE_S8 This is an 8-bit signed integer.

CTLTYPE_S16 This is a 16-bit signed integer.

CTLTYPE_S32 This is a 32-bit signed integer.

CTLTYPE_S64 This is a 64-bit signed integer.

CTLTYPE_OPAQUE This is an opaque data structure.

CTLTYPE_STRUCT Alias for CTLTYPE_OPAQUE.

CTLTYPE_U8 This is an 8-bit unsigned integer.

CTLTYPE_U16 This is a 16-bit unsigned integer.

CTLTYPE_U32 This is a 32-bit unsigned integer.

CTLTYPE_U64 This is a 64-bit unsigned integer.

CTLTYPE_UINT This is an unsigned integer.

CTLTYPE_LONG This is a signed long.

CTLTYPE_ULONG This is an unsigned long.

All sysctl types except for new node declarations require one of the following flags to be set indicating the read and write disposition of the sysctl:

CTLFLAG_RD This is a read-only sysctl.

CTLFLAG_RDTUN This is a read-only sysctl and tunable which is tried fetched once from the system environment early during module load or system boot.

CTLFLAG_WR This is a writable sysctl.

CTLFLAG_RW This sysctl is readable and writable.

CTLFLAG_RWTUN This is a readable and writeable sysctl and tunable which is tried fetched once from the system environment early during module load or system boot.

CTLFLAG_NOFETCH In case the node is marked as a tunable using the CTLFLAG_[XX]TUN, this flag will prevent fetching the initial value from the system environment. Typically this flag should only be used for very early low level system setup

code, and not by common drivers and modules.

CTLFLAG_MPSAFE This sysctl(9) handler is MP safe. Do not grab Giant around calls to this handler. This should only be used for **SYSCTL_PROC**() entries.

Additionally, any of the following optional flags may also be specified:

CTLFLAG_ANYBODY Any user or process can write to this sysctl.

CTLFLAG_CAPRD A process in capability mode can read from this sysctl.

CTLFLAG_CAPWR A process in capability mode can write to this sysctl.

CTLFLAG_SECURE This sysctl can be written to only if the effective securelevel of the process is <= 0.

CTLFLAG_PRISON This sysctl can be written to by processes in jail(2).

CTLFLAG_SKIP When iterating the sysctl name space, do not list this sysctl.

CTLFLAG_TUN Advisory flag that a system tunable also exists for this variable. The initial sysctl value is tried fetched once from the system environment early during module load or system boot.

CTLFLAG_DYN Dynamically created OIDs automatically get this flag set.

CTLFLAG_VNET OID references a VIMAGE-enabled variable.

EXAMPLES

Sample use of **SYSCTL_DECL**() to declare the *security* sysctl tree for use by new nodes:

```
SYSCTL_DECL(_security);
```

Examples of integer, opaque, string, and procedure sysctls follow:

```
/*
 * Example of a constant integer value.  Notice that the control
 * flags are CTLFLAG_RD, the variable pointer is SYSCTL_NULL_INT_PTR,
 * and the value is declared.
 */
```

```
SYSCTL_INT(_debug_sizeof, OID_AUTO, bio, CTLFLAG_RD, SYSCTL_NULL_INT_PTR,
    sizeof(struct bio), "sizeof(struct bio)");

/*
 * Example of a variable integer value.  Notice that the control
 * flags are CTLFLAG_RW, the variable pointer is set, and the
 * value is 0.
 */
static int  doingcache = 1;                     /* 1 => enable the cache */
SYSCTL_INT(_debug, OID_AUTO, vfscache, CTLFLAG_RW, &doingcache, 0,
    "Enable name cache");

/*
 * Example of a variable string value.  Notice that the control
 * flags are CTLFLAG_RW, that the variable pointer and string
 * size are set.  Unlike newer sysctls, this older sysctl uses a
 * static oid number.
 */
char kernelname[MAXPATHLEN] = "/kernel";    /* XXX bloat */
SYSCTL_STRING(_kern, KERN_BOOTFILE, bootfile, CTLFLAG_RW,
    kernelname, sizeof(kernelname), "Name of kernel file booted");

/*
 * Example of an opaque data type exported by sysctl.  Notice that
 * the variable pointer and size are provided, as well as a format
 * string for sysctl(8).
 */
static l_fp pps_freq;/* scaled frequency offset (ns/s) */
SYSCTL_OPAQUE(_kern_ntp_pll, OID_AUTO, pps_freq, CTLFLAG_RD,
    &pps_freq, sizeof(pps_freq), "I", "");

/*
 * Example of a procedure based sysctl exporting string
 * information.  Notice that the data type is declared, the NULL
 * variable pointer and 0 size, the function pointer, and the
 * format string for sysctl(8).
 */
SYSCTL_PROC(_kern_timecounter, OID_AUTO, hardware, CTLTYPE_STRING |
    CTLFLAG_RW, NULL, 0, sysctl_kern_timecounter_hardware, "A",
    "");
```

The following is an example of how to create a new top-level category and how to hook up another subtree to an existing static node. This example does not use contexts, which results in tedious management of all intermediate oids, as they need to be freed later on:

```
#include <sys/sysctl.h>
 ...
/*
 * Need to preserve pointers to newly created subtrees,
 * to be able to free them later:
 */
static struct sysctl_oid *root1;
static struct sysctl_oid *root2;
static struct sysctl_oid *oidp;
static int a_int;
static char *string = "dynamic sysctl";
 ...

root1 = SYSCTL_ADD_ROOT_NODE(NULL,
            OID_AUTO, "newtree", CTLFLAG_RW, 0, "new top level tree");
oidp = SYSCTL_ADD_INT(NULL, SYSCTL_CHILDREN(root1),
            OID_AUTO, "newint", CTLFLAG_RW, &a_int, 0, "new int leaf");
 ...
root2 = SYSCTL_ADD_NODE(NULL, SYSCTL_STATIC_CHILDREN(_debug),
            OID_AUTO, "newtree", CTLFLAG_RW, 0, "new tree under debug");
oidp = SYSCTL_ADD_STRING(NULL, SYSCTL_CHILDREN(root2),
            OID_AUTO, "newstring", CTLFLAG_RD, string, 0, "new string leaf");
```

This example creates the following subtrees:

```
debug.newtree.newstring
newtree.newint
```

Care should be taken to free all OIDs once they are no longer needed!

SYSCTL NAMING

When adding, modifying, or removing sysctl names, it is important to be aware that these interfaces may be used by users, libraries, applications, or documentation (such as published books), and are implicitly published application interfaces. As with other application interfaces, caution must be taken not to break existing applications, and to think about future use of new name spaces so as to avoid the need to rename or remove interfaces that might be depended on in the future.

The semantics chosen for a new sysctl should be as clear as possible, and the name of the sysctl must closely reflect its semantics. Therefore the sysctl name deserves a fair amount of consideration. It should be short but yet representative of the sysctl meaning. If the name consists of several words, they should be separated by underscore characters, as in *compute_summary_at_mount*. Underscore characters may be omitted only if the name consists of not more than two words, each being not longer than four characters, as in *bootfile*.

For boolean sysctls, negative logic should be totally avoided. That is, do not use names like *no_foobar* or *foobar_disable*. They are confusing and lead to configuration errors. Use positive logic instead: *foobar, foobar_enable*.

A temporary sysctl node OID that should not be relied upon must be designated as such by a leading underscore character in its name. For example: *_dirty_hack*.

SEE ALSO

sysctl(3), sysctl(8), device_get_sysctl(9), sysctl_add_oid(9), sysctl_ctx_free(9), sysctl_ctx_init(9), sysctl_remove_oid(9)

HISTORY

The sysctl(8) utility first appeared in 4.4BSD.

AUTHORS

The **sysctl** implementation originally found in BSD has been extensively rewritten by Poul-Henning Kamp in order to add support for name lookups, name space iteration, and dynamic addition of MIB nodes.

This man page was written by Robert N. M. Watson.

SECURITY CONSIDERATIONS

When creating new sysctls, careful attention should be paid to the security implications of the monitoring or management interface being created. Most sysctls present in the kernel are read-only or writable only by the superuser. Sysctls exporting extensive information on system data structures and operation, especially those implemented using procedures, will wish to implement access control to limit the undesired exposure of information about other processes, network connections, etc.

The following top level sysctl name spaces are commonly used:

compat Compatibility layer information.

debug Debugging information. Various name spaces exist under *debug*.

hw Hardware and device driver information.

kern Kernel behavior tuning; generally deprecated in favor of more specific name spaces.

machdep Machine-dependent configuration parameters.

net Network subsystem. Various protocols have name spaces under *net*.

regression Regression test configuration and information.

security Security and security-policy configuration and information.

sysctl Reserved name space for the implementation of sysctl.

user Configuration settings relating to user application behavior. Generally, configuring applications using kernel sysctls is discouraged.

vfs Virtual file system configuration and information.

vm Virtual memory subsystem configuration and information.