NAME

sbuf, **sbuf_new**, **sbuf_new_auto**, **sbuf_new_for_sysctl**, **sbuf_clear**, **sbuf_get_flags**, **sbuf_set_flags**, **sbuf_clear_flags**, **sbuf_setpos**, **sbuf_bcat**, **sbuf_bcopyin**, **sbuf_bcpy**, **sbuf_cat**, **sbuf_copyin**, **sbuf_cpy**, **sbuf_nl_terminate**, **sbuf_printf**, **sbuf_vprintf**, **sbuf_putc**, **sbuf_set_drain**, **sbuf_trim**, **sbuf_error**, **sbuf_finish**, **sbuf_data**, **sbuf_len**, **sbuf_done**, **sbuf_delete**, **sbuf_start_section**, **sbuf_end_section**, **sbuf_hexdump**, **sbuf_printf_drain**, **sbuf_putbuf** - safe string composition

SYNOPSIS

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

typedef int
(sbuf_drain_func)(*void *arg*, *const char *data*, *int len*);

*struct sbuf **
sbuf_new(*struct sbuf *s*, *char *buf*, *int length*, *int flags*);

*struct sbuf **
sbuf_new_auto(*void*);

void
sbuf_clear(*struct sbuf *s*);

int
sbuf_get_flags(*struct sbuf *s*);

void
sbuf_set_flags(*struct sbuf *s*, *int flags*);

void
sbuf_clear_flags(*struct sbuf *s*, *int flags*);

int
sbuf_setpos(*struct sbuf *s*, *int pos*);

int
sbuf_bcat(*struct sbuf *s*, *const void *buf*, *size_t len*);

int
sbuf_bcpy(*struct sbuf *s*, *const void *buf*, *size_t len*);

int
sbuf_cat(*struct sbuf *s, const char *str*);

int
sbuf_cpy(*struct sbuf *s, const char *str*);

int
sbuf_nl_terminate(*struct sbuf **);

int
sbuf_printf(*struct sbuf *s, const char *fmt, ...*);

int
sbuf_vprintf(*struct sbuf *s, const char *fmt, va_list ap*);

int
sbuf_putc(*struct sbuf *s, int c*);

void
sbuf_set_drain(*struct sbuf *s, sbuf_drain_func *func, void *arg*);

int
sbuf_trim(*struct sbuf *s*);

int
sbuf_error(*struct sbuf *s*);

int
sbuf_finish(*struct sbuf *s*);

*char **
sbuf_data(*struct sbuf *s*);

ssize_t
sbuf_len(*struct sbuf *s*);

int
sbuf_done(*struct sbuf *s*);

void

sbuf_delete(*struct sbuf *s*);

void
sbuf_start_section(*struct sbuf *s, ssize_t *old_lenp*);

ssize_t
sbuf_end_section(*struct sbuf *s, ssize_t old_len, size_t pad, int c*);

void
sbuf_hexdump(*struct sbuf *sb, void *ptr, int length, const char *hdr, int flags*);

int
sbuf_printf_drain(*void *arg, const char *data, int len*);

void
sbuf_putbuf(*struct sbuf *s*);

#ifdef _KERNEL

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

int
sbuf_bcopyin(*struct sbuf *s, const void *uaddr, size_t len*);

int
sbuf_copyin(*struct sbuf *s, const void *uaddr, size_t len*);

#include <sys/sysctl.h>

*struct sbuf **
sbuf_new_for_sysctl(*struct sbuf *s, char *buf, int length, struct sysctl_req *req*);

#endif /* _KERNEL */

DESCRIPTION

The **sbuf** family of functions allows one to safely allocate, compose and release strings in kernel or user space.

Instead of arrays of characters, these functions operate on structures called *sbufs*, defined in

<*sys/sbuf.h*>.

Any errors encountered during the allocation or composition of the string will be latched in the data structure, making a single error test at the end of the composition sufficient to determine success or failure of the entire process.

The **sbuf_new**() function initializes the *sbuf* pointed to by its first argument. If that pointer is NULL, **sbuf_new**() allocates a *struct sbuf* using malloc(9). The *buf* argument is a pointer to a buffer in which to store the actual string; if it is NULL, **sbuf_new**() will allocate one using malloc(9). The *length* is the initial size of the storage buffer. The fourth argument, *flags*, may be comprised of the following flags:

SBUF_FIXEDLEN The storage buffer is fixed at its initial size. Attempting to extend the sbuf beyond this size results in an overflow condition.

SBUF_AUTOEXTEND This indicates that the storage buffer may be extended as necessary, so long as resources allow, to hold additional data.

SBUF_INCLUDENUL This causes the final nulterm byte to be counted in the length of the data.

SBUF_DRAINTOEOR Treat top-level sections started with **sbuf_start_section**() as a record boundary marker that will be used during drain operations to avoid records being split. If a record grows sufficiently large such that it fills the *sbuf* and therefore cannot be drained without being split, an error of EDEADLK is set.

SBUF_NOWAIT Indicates that attempts to extend the storage buffer should fail in low memory conditions, like malloc(9) M_NOWAIT.

Note that if *buf* is not NULL, it must point to an array of at least *length* characters. The result of accessing that array directly while it is in use by the sbuf is undefined.

The **sbuf_new_auto**() function is a shortcut for creating a completely dynamic **sbuf**. It is the equivalent of calling **sbuf_new**() with values NULL, NULL, 0, and SBUF_AUTOEXTEND.

The **sbuf_new_for_sysctl**() function will set up an sbuf with a drain function to use **SYSCTL_OUT**() when the internal buffer fills. Note that if the various functions which append to an sbuf are used while a non-sleepable lock is held, the user buffer should be wired using **sysctl_wire_old_buffer**().

The **sbuf_delete**() function clears the *sbuf* and frees any memory allocated for it. There must be a call to **sbuf_delete**() for every call to **sbuf_new**(). Any attempt to access the sbuf after it has been deleted will fail.

The **sbuf_clear**() function invalidates the contents of the *sbuf* and resets its position to zero.

The **sbuf_get_flags**() function returns the current user flags. The **sbuf_set_flags**() and **sbuf_clear_flags**() functions set or clear one or more user flags, respectively. The user flags are described under the **sbuf_new**() function.

The **sbuf_setpos**() function sets the *sbuf*'s end position to *pos*, which is a value between zero and the current position in the buffer. It can only truncate the sbuf to the new position.

The **sbuf_bcat**() function appends the first *len* bytes from the buffer *buf* to the *sbuf*.

The **sbuf_bcopyin**() function copies *len* bytes from the specified userland address into the *sbuf*.

The **sbuf_bcpy**() function replaces the contents of the *sbuf* with the first *len* bytes from the buffer *buf*.

The **sbuf_cat**() function appends the NUL-terminated string *str* to the *sbuf* at the current position.

The **sbuf_set_drain**() function sets a drain function *func* for the *sbuf*, and records a pointer *arg* to be passed to the drain on callback. The drain function cannot be changed while *sbuf_len* is non-zero.

The registered drain function *sbuf_drain_func* will be called with the argument *arg* provided to **sbuf_set_drain**(), a pointer *data* to a byte string that is the contents of the sbuf, and the length *len* of the data. If the drain function exists, it will be called when the sbuf internal buffer is full, or on behalf of **sbuf_finish**(). The drain function may drain some or all of the data, but must drain at least 1 byte. The return value from the drain function, if positive, indicates how many bytes were drained. If negative, the return value indicates the negative error code which will be returned from this or a later call to **sbuf_finish**(). If the returned drained length is 0, an error of EDEADLK is set. To do unbuffered draining, initialize the sbuf with a two-byte buffer. The drain will be called for every byte added to the sbuf. The **sbuf_bcopyin**(), **sbuf_bcpy**(), **sbuf_clear**(), **sbuf_copyin**(), **sbuf_cpy**(), **sbuf_trim**(), **sbuf_data**(), and **sbuf_len**() functions cannot be used on an sbuf with a drain.

The **sbuf_copyin**() function copies a NUL-terminated string from the specified userland address into the *sbuf*. If the *len* argument is non-zero, no more than *len* characters (not counting the terminating NUL) are copied; otherwise the entire string, or as much of it as can fit in the *sbuf*, is copied.

The **sbuf_cpy**() function replaces the contents of the *sbuf* with those of the NUL-terminated string *str*. This is equivalent to calling **sbuf_cat**() with a fresh *sbuf* or one which position has been reset to zero with **sbuf_clear**() or **sbuf_setpos**().

The **sbuf_nl_terminate**() function appends a trailing newline character, if the current line is non-empty

and not already terminated by a newline character.

The **sbuf_printf**() function formats its arguments according to the format string pointed to by *fmt* and appends the resulting string to the *sbuf* at the current position.

The **sbuf_vprintf**() function behaves the same as **sbuf_printf**() except that the arguments are obtained from the variable-length argument list *ap*.

The **sbuf_putc**() function appends the character *c* to the *sbuf* at the current position.

The **sbuf_trim**() function removes trailing whitespace from the *sbuf*.

The **sbuf_error**() function returns any error value that the *sbuf* may have accumulated, either from the drain function, or ENOMEM if the *sbuf* overflowed. This function is generally not needed and instead the error code from **sbuf_finish**() is the preferred way to discover whether an sbuf had an error.

The **sbuf_finish**() function will call the attached drain function if one exists until all the data in the *sbuf* is flushed. If there is no attached drain, **sbuf_finish**() NUL-terminates the *sbuf*. In either case it marks the *sbuf* as finished, which means that it may no longer be modified using **sbuf_setpos**(), **sbuf_cat**(), **sbuf_cpy**(), **sbuf_printf**() or **sbuf_putc**(), until **sbuf_clear**() is used to reset the sbuf.

The **sbuf_data**() function returns the actual string; **sbuf_data**() only works on a finished *sbuf*. The **sbuf_len**() function returns the length of the string. For an *sbuf* with an attached drain, **sbuf_len**() returns the length of the un-drained data. **sbuf_done**() returns non-zero if the *sbuf* is finished.

The **sbuf_start_section**() and **sbuf_end_section**() functions may be used for automatic section alignment. The arguments *pad* and *c* specify the padding size and a character used for padding. The arguments *old_lenp* and *old_len* are to save and restore the current section length when nested sections are used. For the top level section NULL and -1 can be specified for *old_lenp* and *old_len* respectively.

The **sbuf_hexdump**() function prints an array of bytes to the supplied sbuf, along with an ASCII representation of the bytes if possible. See the hexdump(3) man page for more details on the interface.

The **sbuf_printf_drain**() function is a drain function that will call printf, or log to the console. The argument *arg* must be either NULL, or a valid pointer to a *size_t*. If *arg* is not NULL, the total bytes drained will be added to the value pointed to by *arg*.

The **sbuf_putbuf**() function printfs the sbuf to stdout if in userland, and to the console and log if in the kernel. The *sbuf* must be finished before calling **sbuf_putbuf**(). It does not drain the buffer or update any pointers.

NOTES

If an operation caused an *sbuf* to overflow, most subsequent operations on it will fail until the *sbuf* is finished using **sbuf_finish**() or reset using **sbuf_clear**(), or its position is reset to a value between 0 and one less than the size of its storage buffer using **sbuf_setpos**(), or it is reinitialized to a sufficiently short string using **sbuf_cpy**().

Drains in user-space will not always function as indicated. While the drain function will be called immediately on overflow from the *sbuf_putc*, *sbuf_bcat*, *sbuf_cat* functions, *sbuf_printf* and *sbuf_vprintf* currently have no way to determine whether there will be an overflow until after it occurs, and cannot do a partial expansion of the format string. Thus when using libsbuf the buffer may be extended to allow completion of a single printf call, even though a drain is attached.

RETURN VALUES

The **sbuf_new**() function returns NULL if it failed to allocate a storage buffer, and a pointer to the new *sbuf* otherwise.

The **sbuf_setpos**() function returns -1 if *pos* was invalid, and zero otherwise.

The **sbuf_bcat**(), **sbuf_cat**(), **sbuf_cpy**(), **sbuf_printf**(), **sbuf_putc**(), and **sbuf_trim**() functions all return -1 if the buffer overflowed, and zero otherwise.

The **sbuf_error**() function returns a non-zero value if the buffer has an overflow or drain error, and zero otherwise.

The **sbuf_len**() function returns -1 if the buffer overflowed.

The **sbuf_copyin**() function returns -1 if copying string from userland failed, and number of bytes copied otherwise.

The **sbuf_end_section**() function returns the section length or -1 if the buffer has an error.

The **sbuf_finish**(*9*) function (the kernel version) returns ENOMEM if the sbuf overflowed before being finished, or returns the error code from the drain if one is attached.

The **sbuf_finish**(*3*) function (the userland version) will return zero for success and -1 and set errno on error.

EXAMPLES

```
#include <sys/types.h>
#include <sys/sbuf.h>
```

```
struct sbuf *sb;

sb = sbuf_new_auto();
sbuf_cat(sb, "Customers found:\n");
TAILQ_FOREACH(foo, &foolist, list) {
        sbuf_printf(sb, "   %4d %s\n", foo->index, foo->name);
        sbuf_printf(sb, "      Address: %s\n", foo->address);
        sbuf_printf(sb, "      Zip: %s\n", foo->zipcode);
}
if (sbuf_finish(sb) != 0) /* Check for any and all errors */
        err(1, "Could not generate message");
transmit_msg(sbuf_data(sb), sbuf_len(sb));
sbuf_delete(sb);
```

SEE ALSO

hexdump(3), printf(3), strcat(3), strcpy(3), copyin(9), copyinstr(9), printf(9)

HISTORY

The **sbuf** family of functions first appeared in FreeBSD 4.4.

AUTHORS

The **sbuf** family of functions was designed by Poul-Henning Kamp <*phk@FreeBSD.org*> and implemented by Dag-Erling Smørgrav <*des@FreeBSD.org*>. Additional improvements were suggested by Justin T. Gibbs <*gibbs@FreeBSD.org*>. Auto-extend support added by Kelly Yancey <*kbyanc@FreeBSD.org*>. Drain functionality added by Matthew Fleming <*mdf@FreeBSD.org*>.

This manual page was written by Dag-Erling Smørgrav <*des@FreeBSD.org*>.