Rocky Enterprise Linux 9.2 Manual Pages on command 'TAILQ_FIRST.3'

$ man TAILQ_FIRST.3

TAILQ(3) Linux Programmer's Manual TAILQ(3)

NAME

TAILQ_CONCAT, TAILQ_EMPTY, TAILQ_ENTRY, TAILQ_FIRST, TAILQ_FOREACH, TAILQ_FOREACH_REVERSE, TAILQ_HEAD, TAILQ_HEAD_INITIALIZER, TAILQ_INIT, TAILQ_INSERT_AFTER, TAILQ_INSERT_BEFORE, TAILQ_INSERT_HEAD, TAILQ_IN? SERT_TAIL, TAILQ_LAST, TAILQ_NEXT, TAILQ_PREV, TAILQ_REMOVE - implemen? tation of a doubly linked tail queue

SYNOPSIS

```
#include <sys/queue.h>
void TAILQ_CONCAT(TAILQ_HEAD *head1, TAILQ_HEAD *head2,
        TAILQ_ENTRY NAME);
int TAILQ_EMPTY(TAILQ_HEAD *head);
TAILQ_ENTRY(TYPE);
struct TYPE *TAILQ_FIRST(TAILQ_HEAD *head);
TAILQ_FOREACH(struct TYPE *var, TAILQ_HEAD *head, TAILQ_ENTRY NAME);
TAILQ_FOREACH_REVERSE(struct TYPE *var, TAILQ_HEAD *head, HEADNAME,
        TAILQ_ENTRY NAME);
TAILQ_HEAD(HEADNAME, TYPE);
```

TAILQ_HEAD TAILQ_HEAD_INITIALIZER(TAILQ_HEAD head);

void TAILQ_INIT(TAILQ_HEAD *head);

void TAILQ_INSERT_AFTER(TAILQ_HEAD *head, struct TYPE *listelm,
struct TYPE *elm, TAILQ_ENTRY NAME);

void TAILQ_INSERT_BEFORE(struct TYPE *listelm, struct TYPE *elm,
TAILQ_ENTRY NAME);

void TAILQ_INSERT_HEAD(TAILQ_HEAD *head, struct TYPE *elm,
TAILQ_ENTRY NAME);

void TAILQ_INSERT_TAIL(TAILQ_HEAD *head, struct TYPE *elm,
TAILQ_ENTRY NAME);

struct TYPE *TAILQ_LAST(TAILQ_HEAD *head, HEADNAME);

struct TYPE *TAILQ_NEXT(struct TYPE *elm, TAILQ_ENTRY NAME);

struct TYPE *TAILQ_PREV(struct TYPE *elm, HEADNAME, TAILQ_ENTRY NAME);

void TAILQ_REMOVE(TAILQ_HEAD *head, struct TYPE *elm, TAILQ_ENTRY NAME);

DESCRIPTION

These macros define and operate on doubly linked tail queues.

In the macro definitions, TYPE is the name of a user defined structure,
that must contain a field of type TAILQ_ENTRY, named NAME. The argu?
ment HEADNAME is the name of a user defined structure that must be de?
clared using the macro TAILQ_HEAD().

A tail queue is headed by a structure defined by the TAILQ_HEAD()
macro. This structure contains a pair of pointers, one to the first
element in the tail queue and the other to the last element in the tail
queue. The elements are doubly linked so that an arbitrary element can
be removed without traversing the tail queue. New elements can be
added to the tail queue after an existing element, before an existing
element, at the head of the tail queue, or at the end of the tail
queue. A TAILQ_HEAD structure is declared as follows:

TAILQ_HEAD(HEADNAME, TYPE) head;

where struct HEADNAME is the structure to be defined, and struct TYPE
is the type of the elements to be linked into the tail queue. A
pointer to the head of the tail queue can later be declared as:

struct HEADNAME *headp;

(The names head and headp are user selectable.)

The macro TAILQ_HEAD_INITIALIZER() evaluates to an initializer for the tail queue head.

The macro TAILQ_CONCAT() concatenates the tail queue headed by head2 onto the end of the one headed by head1 removing all entries from the former.

The macro TAILQ_EMPTY() evaluates to true if there are no items on the tail queue.

The macro TAILQ_ENTRY() declares a structure that connects the elements in the tail queue.

The macro TAILQ_FIRST() returns the first item on the tail queue or NULL if the tail queue is empty.

The macro TAILQ_FOREACH() traverses the tail queue referenced by head in the forward direction, assigning each element in turn to var. var is set to NULL if the loop completes normally, or if there were no ele? ments.

The macro TAILQ_FOREACH_REVERSE() traverses the tail queue referenced by head in the reverse direction, assigning each element in turn to var.

The macro TAILQ_INIT() initializes the tail queue referenced by head.

The macro TAILQ_INSERT_HEAD() inserts the new element elm at the head of the tail queue.

The macro TAILQ_INSERT_TAIL() inserts the new element elm at the end of the tail queue.

The macro TAILQ_INSERT_AFTER() inserts the new element elm after the element listelm.

The macro TAILQ_INSERT_BEFORE() inserts the new element elm before the element listelm.

The macro TAILQ_LAST() returns the last item on the tail queue. If the tail queue is empty the return value is NULL.

The macro TAILQ_NEXT() returns the next item on the tail queue, or NULL if this item is the last.

The macro TAILQ_PREV() returns the previous item on the tail queue, or

NULL if this item is the first.

The macro TAILQ_REMOVE() removes the element elm from the tail queue.

RETURN VALUE

TAILQ_EMPTY() returns nonzero if the queue is empty, and zero if the queue contains at least one entry.

TAILQ_FIRST(), TAILQ_LAST(), TAILQ_NEXT(), and TAILQ_PREV() return a pointer to the first, last, next or previous TYPE structure, respec? tively.

TAILQ_HEAD_INITIALIZER() returns an initializer that can be assigned to the queue head.

CONFORMING TO

Not in POSIX.1, POSIX.1-2001 or POSIX.1-2008. Present on the BSDs. (TAILQ functions first appeared in 4.4BSD).

BUGS

The macros TAILQ_FOREACH() and TAILQ_FOREACH_REVERSE() don't allow var to be removed or freed within the loop, as it would interfere with the traversal. The macros TAILQ_FOREACH_SAFE() and TAILQ_FOREACH_RE? VERSE_SAFE(), which are present on the BSDs but are not present in glibc, fix this limitation by allowing var to safely be removed from the list and freed from within the loop without interfering with the traversal.

EXAMPLES

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/queue.h>
struct entry {
    int data;
    TAILQ_ENTRY(entry) entries;             /* Tail queue. */
};
TAILQ_HEAD(tailhead, entry);
int
main(void)
```

```
{
    struct entry *n1, *n2, *n3, *np;
    struct tailhead head;                   /* Tail queue head. */
    int i;
    TAILQ_INIT(&head);                      /* Initialize the queue. */
    n1 = malloc(sizeof(struct entry));      /* Insert at the head. */
    TAILQ_INSERT_HEAD(&head, n1, entries);
    n1 = malloc(sizeof(struct entry));      /* Insert at the tail. */
    TAILQ_INSERT_TAIL(&head, n1, entries);
    n2 = malloc(sizeof(struct entry));      /* Insert after. */
    TAILQ_INSERT_AFTER(&head, n1, n2, entries);
    n3 = malloc(sizeof(struct entry));      /* Insert before. */
    TAILQ_INSERT_BEFORE(n2, n3, entries);
    TAILQ_REMOVE(&head, n2, entries);       /* Deletion. */
    free(n2);
                                    /* Forward traversal. */
    i = 0;
    TAILQ_FOREACH(np, &head, entries)
        np->data = i++;
                                    /* Reverse traversal. */
    TAILQ_FOREACH_REVERSE(np, &head, tailhead, entries)
        printf("%i\n", np->data);
                                    /* TailQ Deletion. */
    n1 = TAILQ_FIRST(&head);
    while (n1 != NULL) {
        n2 = TAILQ_NEXT(n1, entries);
        free(n1);
        n1 = n2;
    }
    TAILQ_INIT(&head);
    exit(EXIT_SUCCESS);
}
```

insque(3), queue(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.