

**NAME**

`argz_add`, `argz_add_sep`, `argz_append`, `argz_count`, `argz_create`, `argz_create_sep`, `argz_delete`, `argz_extract`, `argz_insert`, `argz_next`, `argz_replace`, `argz_stringify` – functions to handle an argz list

**SYNOPSIS**

```
#include <argz.h>

error_t argz_add(char **argz, size_t *argz_len, const char *str);

error_t argz_add_sep(char **argz, size_t *argz_len,
                    const char *str, int delim);

error_t argz_append(char **argz, size_t *argz_len,
                   const char *buf, size_t buf_len);

size_t argz_count(const char *argz, size_t argz_len);

error_t argz_create(char * const argv[], char **argz,
                  size_t *argz_len);

error_t argz_create_sep(const char *str, int sep, char **argz,
                      size_t *argz_len);

void argz_delete(char **argz, size_t *argz_len, char *entry);

void argz_extract(const char *argz, size_t argz_len, char **argv);

error_t argz_insert(char **argz, size_t *argz_len, char *before,
                  const char *entry);

char *argz_next(const char *argz, size_t argz_len, const char *entry);

error_t argz_replace(char **argz, size_t *argz_len, const char *str,
                   const char *with, unsigned int *replace_count);

void argz_stringify(char *argz, size_t len, int sep);
```

**DESCRIPTION**

These functions are glibc-specific.

An argz vector is a pointer to a character buffer together with a length. The intended interpretation of the character buffer is an array of strings, where the strings are separated by null bytes (`'\0'`). If the length is nonzero, the last byte of the buffer must be a null byte.

These functions are for handling argz vectors. The pair `(NULL,0)` is an argz vector, and, conversely, argz vectors of length 0 must have null pointer. Allocation of nonempty argz vectors is done using `malloc(3)`, so that `free(3)` can be used to dispose of them again.

**argz\_add()** adds the string *str* at the end of the array *\*argz*, and updates *\*argz* and *\*argz\_len*.

**argz\_add\_sep()** is similar, but splits the string *str* into substrings separated by the delimiter *delim*. For example, one might use this on a UNIX search path with delimiter `':'`.

**argz\_append()** appends the argz vector (*buf*, *buf\_len*) after (*\*argz*, *\*argz\_len*) and updates *\*argz* and *\*argz\_len*. (Thus, *\*argz\_len* will be increased by *buf\_len*.)

**argz\_count()** counts the number of strings, that is, the number of null bytes (`'\0'`), in (*argz*, *argz\_len*).

**argz\_create()** converts a UNIX-style argument vector *argv*, terminated by (*char \**) 0, into an argz vector (*\*argz*, *\*argz\_len*).

**argz\_create\_sep()** converts the null-terminated string *str* into an argz vector (*\*argz*, *\*argz\_len*) by breaking it up at every occurrence of the separator *sep*.

**argz\_delete()** removes the substring pointed to by *entry* from the argz vector (*\*argz*, *\*argz\_len*) and updates *\*argz* and *\*argz\_len*.

**argz\_extract()** is the opposite of **argz\_create()**. It takes the argz vector (*argz*, *argz\_len*) and fills the array starting at *argv* with pointers to the substrings, and a final NULL, making a UNIX-style argv vector. The

array *argv* must have room for *argz\_count(argz, argz\_len) + 1* pointers.

**argz\_insert()** is the opposite of **argz\_delete()**. It inserts the argument *entry* at position *before* into the argz vector (*\*argz*, *\*argz\_len*) and updates *\*argz* and *\*argz\_len*. If *before* is NULL, then *entry* will be inserted at the end.

**argz\_next()** is a function to step through the argz vector. If *entry* is NULL, the first entry is returned. Otherwise, the entry following is returned. It returns NULL if there is no following entry.

**argz\_replace()** replaces each occurrence of *str* with *with*, reallocating argz as necessary. If *replace\_count* is non-NULL, *\*replace\_count* will be incremented by the number of replacements.

**argz\_stringify()** is the opposite of **argz\_create\_sep()**. It transforms the argz vector into a normal string by replacing all null bytes ('\0') except the last by *sep*.

## RETURN VALUE

All argz functions that do memory allocation have a return type of *error\_t*, and return 0 for success, and **ENOMEM** if an allocation error occurs.

## ATTRIBUTES

For an explanation of the terms used in this section, see **attributes(7)**.

Interface	Attribute	Value
<b>argz_add()</b> , <b>argz_add_sep()</b> , <b>argz_append()</b> , <b>argz_count()</b> , <b>argz_create()</b> , <b>argz_create_sep()</b> , <b>argz_delete()</b> , <b>argz_extract()</b> , <b>argz_insert()</b> , <b>argz_next()</b> , <b>argz_replace()</b> , <b>argz_stringify()</b>	Thread safety	MT-Safe

## CONFORMING TO

These functions are a GNU extension. Handle with care.

## BUGS

Argz vectors without a terminating null byte may lead to Segmentation Faults.

## SEE ALSO

**envz\_add(3)**

## COLOPHON

This page is part of release 5.02 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/>.