NAME

vis, **nvis**, **strvis**, **stravis**, **strnvis**, **strvisx**, **strnvisx**, **strenvisx**, **svis**, **snvis**, **strsvis**, **strsnvis**, **strsvisx**, **strsnvisx**, **strsenvisx** - visually encode characters

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <vis.h>

*char **
vis(*char *dst, int c, int flag, int nextc*);

*char **
nvis(*char *dst, size_t dlen, int c, int flag, int nextc*);

int
strvis(*char *dst, const char *src, int flag*);

int
stravis(*char **dst, const char *src, int flag*);

int
strnvis(*char *dst, size_t dlen, const char *src, int flag*);

int
strvisx(*char *dst, const char *src, size_t len, int flag*);

int
strnvisx(*char *dst, size_t dlen, const char *src, size_t len, int flag*);

int
strenvisx(*char *dst, size_t dlen, const char *src, size_t len, int flag, int *cerr_ptr*);

*char **
svis(*char *dst, int c, int flag, int nextc, const char *extra*);

*char **
snvis(*char *dst, size_t dlen, int c, int flag, int nextc, const char *extra*);

int
strsvis(*char *dst, const char *src, int flag, const char *extra*);

int
strsnvis(*char *dst, size_t dlen, const char *src, int flag, const char *extra*);

int
strsvisx(*char *dst, const char *src, size_t len, int flag, const char *extra*);

int
strsnvisx(*char *dst, size_t dlen, const char *src, size_t len, int flag, const char *extra*);

int
strsenvisx(*char *dst, size_t dlen, const char *src, size_t len, int flag, const char *extra, int *cerr_ptr*);

DESCRIPTION

The **vis**() function copies into *dst* a string which represents the character *c*. If *c* needs no encoding, it is copied in unaltered. The string is null terminated, and a pointer to the end of the string is returned. The maximum length of any encoding is four bytes (not including the trailing NUL); thus, when encoding a set of characters into a buffer, the size of the buffer should be four times the number of bytes encoded, plus one for the trailing NUL. The flag parameter is used for altering the default range of characters considered for encoding and for altering the visual representation. The additional character, *nextc*, is only used when selecting the VIS_CSTYLE encoding format (explained below).

The **strvis**(), **stravis**(), **strnvis**(), **strvisx**(), and **strnvisx**() functions copy into *dst* a visual representation of the string *src*. The **strvis**() and **strnvis**() functions encode characters from *src* up to the first NUL. The **strvisx**() and **strnvisx**() functions encode exactly *len* characters from *src* (this is useful for encoding a block of data that may contain NUL's). Both forms NUL terminate *dst*. The size of *dst* must be four times the number of bytes encoded from *src* (plus one for the NUL). Both forms return the number of characters in *dst* (not including the trailing NUL). The **stravis**() function allocates space dynamically to hold the string. The "**n**" versions of the functions also take an additional argument *dlen* that indicates the length of the *dst* buffer. If *dlen* is not large enough to fit the converted string then the **strnvis**() and **strnvisx**() functions return -1 and set *errno* to ENOSPC. The **strenvisx**() function takes an additional argument, *cerr_ptr*, that is used to pass in and out a multibyte conversion error flag. This is useful when processing single characters at a time when it is possible that the locale may be set to something other than the locale of the characters in the input data.

The functions **svis**(), **snvis**(), **strsvis**(), **strsnvis**(), **strsvisx**(), **strsnvisx**(), and **strsenvisx**() correspond to **vis**(), **nvis**(), **strvis**(), **strnvis**(), **strvisx**(), **strnvisx**(), and **strenvisx**() but have an additional argument *extra*, pointing to a NUL terminated list of characters. These characters will be copied encoded or

backslash-escaped into *dst*. These functions are useful e.g. to remove the special meaning of certain characters to shells.

The encoding is a unique, invertible representation composed entirely of graphic characters; it can be decoded back into the original form using the unvis(3), strunvis(3) or strnunvis(3) functions.

There are two parameters that can be controlled: the range of characters that are encoded (applies only to **vis**(), **nvis**(), **strvis**(), **strnvis**(), **strvisx**(), and **strnvisx**()), and the type of representation used. By default, all non-graphic characters, except space, tab, and newline are encoded (see isgraph(3)). The following flags alter this:

VIS_DQ Also encode double quotes

VIS_GLOB Also encode the magic characters (‘*’, ‘?’, ‘[’, and ‘#’) recognized by glob(3).

VIS_SHELL Also encode the meta characters used by shells (in addition to the glob characters): (‘’’, ‘‘’, ‘"’, ‘;’, ‘&’, ‘<’, ‘>’, ‘(’, ‘)’, ‘|’, ‘]’, ‘\’, ‘$’, ‘!’, ‘^’, and ‘~’).

VIS_SP Also encode space.

VIS_TAB Also encode tab.

VIS_NL Also encode newline.

VIS_WHITE Synonym for VIS_SP | VIS_TAB | VIS_NL.

VIS_META Synonym for VIS_WHITE | VIS_GLOB | VIS_SHELL.

VIS_SAFE Only encode "unsafe" characters. Unsafe means control characters which may cause common terminals to perform unexpected functions. Currently this form allows space, tab, newline, backspace, bell, and return -- in addition to all graphic characters -- unencoded.

(The above flags have no effect for **svis**(), **snvis**(), **strsvis**(), **strsnvis**(), **strsvisx**(), and **strsnvisx**(). When using these functions, place all graphic characters to be encoded in an array pointed to by *extra*. In general, the backslash character should be included in this array, see the warning on the use of the VIS_NOSLASH flag below).

There are six forms of encoding. All forms use the backslash character ‘\’ to introduce a special sequence; two backslashes are used to represent a real backslash, except VIS_HTTPSTYLE that uses

‘%’, or VIS_MIMESTYLE that uses ‘=’. These are the visual formats:

(default) Use an ‘M’ to represent meta characters (characters with the 8th bit set), and use caret ‘^’ to represent control characters (see iscntrl(3)). The following formats are used:

\^C Represents the control character ‘C’. Spans characters ‘\000’ through ‘\037’, and ‘\177’ (as ‘\^?’).

\M-C Represents character ‘C’ with the 8th bit set. Spans characters ‘\241’ through ‘\376’.

\M^C Represents control character ‘C’ with the 8th bit set. Spans characters ‘\200’ through ‘\237’, and ‘\377’ (as ‘\M^?’).

\040 Represents ASCII space.

\240 Represents Meta-space.

VIS_CSTYLE Use C-style backslash sequences to represent standard non-printable characters. The following sequences are used to represent the indicated characters:

\a -- BEL (007)
\b -- BS (010)
\f -- NP (014)
\n -- NL (012)
\r -- CR (015)
\s -- SP (040)
\t -- HT (011)
\v -- VT (013)
\0 -- NUL (000)

When using this format, the *nextc* parameter is looked at to determine if a NUL character can be encoded as ‘\0’ instead of ‘\000’. If *nextc* is an octal digit, the latter representation is used to avoid ambiguity.

Non-printable characters without C-style backslash sequences use the default representation.

VIS_OCTAL Use a three digit octal sequence. The form is ‘\ddd’ where *d* represents an octal digit.

VIS_CSTYLE | VIS_OCTAL
: Same as VIS_CSTYLE except that non-printable characters without C-style backslash sequences use a three digit octal sequence.

VIS_HTTPSTYLE
: Use URI encoding as described in RFC 1738. The form is '%xx' where *x* represents a lower case hexadecimal digit.

VIS_MIMESTYLE
: Use MIME Quoted-Printable encoding as described in RFC 2045, only don't break lines and don't handle CRLF. The form is '=XX' where *X* represents an upper case hexadecimal digit.

There is one additional flag, VIS_NOSLASH, which inhibits the doubling of backslashes and the backslash before the default format (that is, control characters are represented by '^C' and meta characters as 'M-C'). With this flag set, the encoding is ambiguous and non-invertible.

MULTIBYTE CHARACTER SUPPORT

These functions support multibyte character input. The encoding conversion is influenced by the setting of the LC_CTYPE environment variable which defines the set of characters that can be copied without encoding.

If VIS_NOLOCALE is set, processing is done assuming the C locale and overriding any other environment settings.

When 8-bit data is present in the input, LC_CTYPE must be set to the correct locale or to the C locale. If the locales of the data and the conversion are mismatched, multibyte character recognition may fail and encoding will be performed byte-by-byte instead.

As noted above, *dst* must be four times the number of bytes processed from *src*. But note that each multibyte character can be up to MB_LEN_MAX bytes so in terms of multibyte characters, *dst* must be four times MB_LEN_MAX times the number of characters processed from *src*.

ENVIRONMENT

LC_CTYPE Specify the locale of the input data. Set to C if the input data locale is unknown.

ERRORS

The functions **nvis**() and **snvis**() will return NULL and the functions **strnvis**(), **strnvisx**(), **strsnvis**(), and **strsnvisx**(), will return -1 when the *dlen* destination buffer size is not enough to perform the conversion while setting *errno* to:

[ENOSPC] The destination buffer size is not large enough to perform the conversion.

SEE ALSO

unvis(1), vis(1), glob(3), unvis(3)

T. Berners-Lee, *Uniform Resource Locators (URL)*, RFC 1738.

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC 2045.

HISTORY

The **vis**(), **strvis**(), and **strvisx**() functions first appeared in 4.4BSD. The **svis**(), **strsvis**(), and **strsvisx**() functions appeared in NetBSD 1.5 and FreeBSD 9.2. The buffer size limited versions of the functions (**nvis**(), **strnvis**(), **strnvisx**(), **snvis**(), **strsnvis**(), and **strsnvisx**()) appeared in NetBSD 6.0 and FreeBSD 9.2. Multibyte character support was added in NetBSD 7.0 and FreeBSD 9.2.