Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fscanf.3p' command

$ man fscanf.3p

FSCANF(3P) POSIX Programmer's Manual FSCANF(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fscanf, scanf, sscanf ? convert formatted input

SYNOPSIS

```
#include <stdio.h>
int fscanf(FILE *restrict stream, const char *restrict format, ...);
int scanf(const char *restrict format, ...);
int sscanf(const char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The fscanf() function shall read from the named input stream. The scanf() function shall read from the standard input stream stdin. The sscanf() function shall read from the string s. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string format de?

scribed below, and a set of pointer arguments indicating where the con? verted input should be stored. The result is undefined if there are in? sufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but otherwise ignored.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion specifier character % (see below) is replaced by the se? quence "%n$", where n is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific lan? guages. In format strings containing the "%n$" form of conversion spec? ifications, it is unspecified whether numbered arguments in the argu? ment list can be referenced from the format string more than once.

The format can contain either form of a conversion specification?that is, % or "%n$"?but the two forms cannot be mixed within a single format string. The only exception to this is that %% or %* can be mixed with the "%n$" form. When numbered argument specifications are used, speci? fying the Nth argument requires that all the leading arguments, from the first to the (N-1)th, are pointers.

The fscanf() function in all its forms shall allow detection of a lan? guage-dependent radix character in the input string. The radix charac? ter is defined in the current locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

The format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each direc? tive is composed of one of the following: one or more white-space char? acters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary character (neither '%' nor a white-space character); or a con? version specification. Each conversion specification is introduced by the character '%' or the character sequence "%n$", after which the fol? lowing appear in sequence:

* An optional assignment-suppressing character '*'.
* An optional non-zero decimal integer that specifies the maximum field width.
* An optional assignment-allocation character 'm'.
* An option length modifier that specifies the size of the receiving object.
* A conversion specifier character that specifies the type of conver? sion to be applied. The valid conversion specifiers are described below.

The fscanf() functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall re? turn. Failures are described as input failures (due to the unavailabil? ity of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space characters shall be ex? ecuted by reading input until no more valid input can be read, or up to the first byte which is not a white-space character, which remains un? read.

A directive that is an ordinary character shall be executed as follows: the next byte shall be read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent bytes shall remain unread. Similarly, if end-of-file, an encoding er? ror, or a read error prevents a character from being read, the direc? tive shall fail.

A directive that is a conversion specification defines a set of match? ing input sequences, as described below for each conversion character.

A conversion specification shall be executed in the following steps.

Input white-space characters (as specified by isspace()) shall be skipped, unless the conversion specification includes a [, c, C, or n conversion specifier.

An item shall be read from the input, unless the conversion specifica? tion includes an n conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to any specified maximum

field width, which may be measured in characters or bytes dependent on the conversion specifier) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item shall remain un? read. If the length of the input item is 0, the execution of the con? version specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a % conversion specifier, the input item (or, in the case of a %n conversion specification, the count of input bytes) shall be converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the con? version specification fails; this condition is a matching failure. Un? less assignment suppression was indicated by a '*', the result of the conversion shall be placed in the object pointed to by the first argu? ment following the format argument that has not already received a con? version result if the conversion specification is introduced by %, or in the nth argument if introduced by the character sequence "%n$". If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The %c, %s, and %[conversion specifiers shall accept an optional as? signment-allocation character 'm', which shall cause a memory buffer to be allocated to hold the string converted including a terminating null character. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer variable that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if malloc() had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set errno to [ENOMEM] and a con? version error shall result. If the function returns EOF, any memory successfully allocated for parameters using assignment-allocation char? acter 'm' by this call shall be freed before the function returns.

The length modifiers and their meanings are:

hh Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or unsigned char.

h Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short.

l (ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long or unsigned long; that a following a, A, e, E, f, F, g, or G con? version specifier applies to an argument with type pointer to double; or that a following c, s, or [conversion specifier ap? plies to an argument with type pointer to wchar_t. If the 'm' assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to wchar_t.

ll (ell-ell)

Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long or unsigned long long.

j Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to intmax_t or uintmax_t.

z Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to size_t or the corresponding signed integer type.

t Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.

L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to long dou? ble.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The following conversion specifiers are valid:

d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of strtol() with the value 10 for the base argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int.

i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of strtol() with 0 for the base argument. In the absence of a size modifier, the applica? tion shall ensure that the corresponding argument is a pointer to int.

o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of strtoul() with the value 8 for the base argument. In the absence of a size modi? fier, the application shall ensure that the corresponding argu? ment is a pointer to unsigned.

u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of strtoul() with the value 10 for the base argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned.

x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of strtoul() with the value 16 for the base argument. In the absence of a size modifier, the application shall ensure that the corre? sponding argument is a pointer to unsigned.

a, e, f, g

Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of strtod(). In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to float.

If the fprintf() family of functions generates character string

representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754?1985, the fs?canf() family of functions shall recognize them as input.

s Matches a sequence of bytes that are not white-space charac?ters. If the 'm' assignment-allocation character is not speci?fied, the application shall ensure that the corresponding argu?ment is a pointer to the initial byte of an array of char, signed char, or unsigned char large enough to accept the se?quence and a terminating null character code, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a char.

If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each charac?ter shall be converted to a wide character as if by a call to the mbrtowc() function, with the conversion state described by an mbstate_t object initialized to zero before the first char?acter is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corre?sponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t.

[Matches a non-empty sequence of bytes from a set of expected bytes (the scanset). The normal skip over white-space charac?ters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the ini?tial byte of an array of char, signed char, or unsigned char large enough to accept the sequence and a terminating null byte, which shall be added automatically. Otherwise, the ap?plication shall ensure that the corresponding argument is a

pointer to a pointer to a char.

If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each charac? ter in the sequence shall be converted to a wide character as if by a call to the mbrtowc() function, with the conversion state described by an mbstate_t object initialized to zero be? fore the first character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the termi? nating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t.

The conversion specification includes all subsequent bytes in the format string up to and including the matching <right-square-bracket> (']'). The bytes between the square brackets (the scanlist) comprise the scanset, unless the byte after the <left-square-bracket> is a <circumflex> ('^'), in which case the scanset contains all bytes that do not appear in the scan? list between the <circumflex> and the <right-square-bracket>. If the conversion specification begins with "[]" or "[^]", the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined.

c Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space characters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the ap?

plication shall ensure that the corresponding argument is a pointer to the initial byte of an array of char, signed char, or unsigned char large enough to accept the sequence. Other? wise, the application shall ensure that the corresponding argu? ment is a pointer to a pointer to a char.

If an l (ell) qualifier is present, the input shall be a se? quence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the mbrtowc() function, with the conversion state described by an mbstate_t object initialized to zero be? fore the first character is converted. No null wide character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t.

p Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding fprintf() func? tions. The application shall ensure that the corresponding ar? gument is a pointer to a pointer to void. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.

n No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the fscanf() functions. Execution of a %n con? version specification shall not increment the assignment count returned at the completion of execution of the function. No ar?

gument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.

C Equivalent to lc.

S Equivalent to ls.

% Matches a single '%' character; no conversion or assignment occurs. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and x, respectively.

If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in sscanf() shall be equivalent to encountering end-of-file for fscanf().

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including <newline> characters) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.

The fscanf() and scanf() functions may mark the last data access timestamp of the file associated with stream for update. The last data access timestamp shall be marked for update by the first successful execution of fgetc(), fgets(), fread(), getc(), getchar(), getdelim(), getline(), gets(), fscanf(), or scanf() using stream that returns data not supplied by a prior call to ungetc().

RETURN VALUE

Upon successful completion, these functions shall return the number of

successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned and errno shall be set to indi? cate the error. If a read error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the fscanf() functions fail and may fail, refer to fgetc() or fgetwc().

In addition, the fscanf() function shall fail if:

EILSEQ Input byte sequence does not form a valid character.

ENOMEM Insufficient storage space is available.

In addition, the fscanf() function may fail if:

EINVAL There are insufficient arguments.

The following sections are informative.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to n the value 3, to i the value 25, to x the value 5.432, and name contains the string "Hamster".

The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to i, 789.0 to x, skips 0123, and places the string "56\0" in name. The next call to getchar() shall return the character 'a'.

The following call uses fscanf() to read three floating-point numbers from standard input into the input array.

```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

APPLICATION USAGE

If the application calling fscanf() has any objects of type wint_t or wchar_t, it must also include the <wchar.h> header to have these objects defined.

For functions that allocate memory as if by malloc(), the application should release such memory when it is no longer required by a call to free(). For fscanf(), this is memory allocated via use of the 'm' assignment-allocation character.

RATIONALE

This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few ``obvious'' things were not included. Specifically, the set of characters allowed in a scanset is limited to single-byte characters. In other similar places, multi-byte characters have been permitted, but for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications needing this could use the corresponding wide-character functions to achieve the desired results.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Standard I/O Streams, fprintf(), getc(), setlocale(), strtod(), strtol(), strtoul(), wcrtomb()

The Base Definitions volume of POSIX.1?2017, Chapter 7, Locale, <inttypes.h>, <langinfo.h>, <stdio.h>, <wchar.h>

COPYRIGHT

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .