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 'fwprintf.3p' command

$ man fwprintf.3p

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

fwprintf, swprintf, wprintf ? print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>
int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, ...);
int wprintf(const wchar_t *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 fwprintf() function shall place output on the named output stream. The wprintf() function shall place output on the standard output stream stdout. The swprintf() function shall place output followed by the

null wide character in consecutive wide characters starting at *ws; no more than n wide characters shall be written, including a terminating null wide character, which is always added (unless n is zero).

Each of these functions shall convert, format, and print its arguments under control of the format wide-character string. The format is com? posed of zero or more directives: ordinary wide-characters, which are simply copied to the output stream, and conversion specifications, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are evaluated but are 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 wide character % (see below) is replaced by the sequence "%n$", where n is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format wide-character strings that select arguments in an order appropriate to specific lan? guages (see the EXAMPLES section).

The format can contain either numbered argument specifications (that is, "%n$" and "*m$"), or unnumbered argument conversion specifications (that is, % and *), but not both. The only exception to this is that %% can be mixed with the "%n$" form. The results of mixing numbered and unnumbered argument specifications in a format wide-character string are undefined. When numbered argument specifications are used, specify? ing the Nth argument requires that all the leading arguments, from the first to the (N-1)th, are specified in the format wide-character string.

In format wide-character strings containing the "%n$" form of conver? sion specification, numbered arguments in the argument list can be ref? erenced from the format wide-character string as many times as re? quired.

In format wide-character strings containing the % form of conversion

specification, each argument in the argument list shall be used exactly once. It is unspecified whether an encoding error occurs if the format string contains wchar_t values that do not correspond to members of the character set of the current locale and the specified semantics do not require that value to be processed by wcrtomb().

All forms of the fwprintf() function allow for the insertion of a lo? cale-dependent radix character in the output string, output as a wide-character value. The radix character 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> ('.').

Each conversion specification is introduced by the '%' wide character or by the wide-character sequence "%n$", after which the following ap? pear in sequence:

* Zero or more flags (in any order), which modify the meaning of the conversion specification.
* An optional minimum field width. If the converted value has fewer wide characters than the field width, it shall be padded with <space> characters by default on the left; it shall be padded on the right, if the left-adjustment flag ('-'), described below, is given to the field width. The field width takes the form of an <as? terisk> ('*'), described below, or a decimal integer.
* An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the num? ber of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of signifi? cant digits for the g and G conversion specifiers; or the maximum number of wide characters to be printed from a string in the s con? version specifiers. The precision takes the form of a <period> ('.') followed either by an <asterisk> ('*'), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion wide character, the behavior is undefined.

* An optional length modifier that specifies the size of the argu? ment.

* A conversion specifier wide character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an argument of type int supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format wide-character strings contain? ing the "%n$" form of a conversion specification, a field width or pre? cision may be indicated by the sequence "*m$", where m is a decimal in? teger in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag wide characters and their meanings are:

' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For other conversions, the behavior is undefined. The numeric grouping wide character is used.

- The result of the conversion shall be left-justified within the field. The conversion shall be right-justified if this flag is not specified.

+ The result of a signed conversion shall always begin with a sign ('+' or '-'). The conversion shall begin with a sign only when a negative value is converted if this flag is not speci? fied.

<space> If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a <space> shall be prefixed to the result. This means that if

the <space> and '+' flags both appear, the <space> flag shall be ignored.

Specifies that the value is to be converted to an alternative form. For o conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be zero (if the value and precision are both 0, a single 0 is printed). For x or X conversion specifiers, a non-zero re? sult shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always con? tain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they normally are. For other conversion specifiers, the be? havior is undefined.

0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the '0' and '-' flags both appear, the '0' flag shall be ig? nored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and <apostrophe> flags both appear, the grouping wide char? acters are inserted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

hh Specifies that a following d, i, o, u, x, or X conversion spec? ifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer pro? motions, but its value shall be converted to signed char or un? signed char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.

h Specifies that a following d, i, o, u, x, or X conversion spec?

ifier applies to a short or unsigned short argument (the argu? ment will have been promoted according to the integer promo? tions, but its value shall be converted to short or unsigned short before printing); or that a following n conversion speci? fier applies to a pointer to a short argument.

l (ell) Specifies that a following d, i, o, u, x, or X conversion spec? ifier applies to a long or unsigned long argument; that a fol? lowing n conversion specifier applies to a pointer to a long argument; that a following c conversion specifier applies to a wint_t argument; that a following s conversion specifier ap? plies to a pointer to a wchar_t argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

ll (ell-ell)

Specifies that a following d, i, o, u, x, or X conversion spec? ifier applies to a long long or unsigned long long argument; or that a following n conversion specifier applies to a pointer to a long long argument.

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

z Specifies that a following d, i, o, u, x, or X conversion spec? ifier applies to a size_t or the corresponding signed integer type argument; or that a following n conversion specifier ap? plies to a pointer to a signed integer type corresponding to a size_t argument.

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

L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a long double argument.

If a length modifier appears with any conversion specifier other than

as specified above, the behavior is undefined.

The conversion specifiers and their meanings are:

d, i The int argument shall be converted to a signed decimal in the style "[-]dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be repre? sented in fewer digits, it shall be expanded with leading ze? ros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide char? acters.

o The unsigned argument shall be converted to unsigned octal for? mat in the style "dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of convert? ing zero with an explicit precision of zero shall be no wide characters.

u The unsigned argument shall be converted to unsigned decimal format in the style "dddd". The precision specifies the mini? mum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

x The unsigned argument shall be converted to unsigned hexadeci? mal format in the style "dddd"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer dig? its, it shall be expanded with leading zeros. The default pre? cision shall be 1. The result of converting zero with an ex? plicit precision of zero shall be no wide characters.

X Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".

f, F The double argument shall be converted to decimal notation in

the style "[-]ddd.ddd", where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.

A double argument representing an infinity shall be converted in one of the styles "[-]inf" or "[-]infinity"; which style is implementation-defined. A double argument representing a NaN shall be converted in one of the styles "[-]nan" or "[-]nan(n-char-sequence)"; which style, and the meaning of any n-char-se? quence, is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infin? ity", or "nan", respectively.

e, E The double argument shall be converted in the style "[-]d.ddde?dd", where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix charac? ter shall appear. The value shall be rounded in an implementa? tion-defined manner to the appropriate number of digits. The E conversion wide character shall produce a number with 'E' in? stead of 'e' introducing the exponent. The exponent shall al? ways contain at least two digits. If the value is zero, the ex? ponent shall be zero.

A double argument representing an infinity or NaN shall be con? verted in the style of an f or F conversion specifier.

g, G The double argument representing a floating-point number shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), depending on the value con?

verted and the precision. Let P equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E would have an exponent of X:

-- If P>X?-4, the conversion shall be with style f (or F) and precision P-(X+1).

-- Otherwise, the conversion shall be with style e (or E) and precision P-1.

Finally, unless the '#' flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.

A double argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

a, A A double argument representing a floating-point number shall be converted in the style "[-]0xh.hhhhp?d", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type double, except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.

A double argument representing an infinity or NaN shall be con? verted in the style of an f or F conversion specifier.

c If no l (ell) qualifier is present, the int argument shall be converted to a wide character as if by calling the btowc() function and the resulting wide character shall be written. Otherwise, the wint_t argument shall be converted to wchar_t, and written.

s If no l (ell) qualifier is present, the application shall en? sure that the argument is a pointer to a character array con? taining a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the mbrtowc() function, with the conversion state described by an mbstate_t object initialized to zero be? fore the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall en? sure that the array contains a null wide character.
If an l (ell) qualifier is present, the application shall en? sure that the argument is a pointer to an array of type wchar_t. Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the ar? ray, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.

p The application shall ensure that the argument is a pointer to void. The value of the pointer shall be converted to a se? quence of printable wide characters in an implementation-de? fined manner.

n The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters

written to the output so far by this call to one of the fw? printf() functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is unde? fined.

C Equivalent to lc.

S Equivalent to ls.

% Output a '%' wide character; no argument shall be converted. The entire conversion specification shall be %%.

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Charac? ters generated by fwprintf() and wprintf() shall be printed as if fputwc() had been called.

For a and A conversions, if FLT_RADIX is not a power of 2 and the re? sult is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For e, E, f, F, g, and G conversion specifiers, if the number of sig? nificant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representa? tion with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings L < U, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy L <= D <= U, with the extra stipulation that the error should have a correct sign for the current rounding direction.

The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful

execution of fwprintf() or wprintf() and the next successful completion of a call to fflush() or fclose() on the same stream, or a call to exit() or abort().

RETURN VALUE

Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide char? acter in the case of swprintf(), or a negative value if an output error was encountered, and set errno to indicate the error.

If n or more wide characters were requested to be written, swprintf() shall return a negative value, and set errno to indicate the error.

ERRORS

For the conditions under which fwprintf() and wprintf() fail and may fail, refer to fputwc().

In addition, all forms of fwprintf() shall fail if:

EILSEQ A wide-character code that does not correspond to a valid char? acter has been detected.

In addition, fwprintf() and wprintf() may fail if:

ENOMEM Insufficient storage space is available.

The swprintf() shall fail if:

EOVERFLOW

The value of n is greater than {INT_MAX} or the number of bytes needed to hold the output excluding the terminating null is greater than {INT_MAX}.

The following sections are informative.

EXAMPLES

To print the language-independent date and time format, the following statement could be used:

```
wprintf(format, weekday, month, day, hour, min);
```

For American usage, format could be a pointer to the wide-character string:

```
L"%s, %s %d, %d:%.2d\n"
```

producing the message:

Sunday, July 3, 10:02

whereas for German usage, format could be a pointer to the wide-charac?

ter string:

L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"

producing the message:

Sonntag, 3. Juli, 10:02

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Standard I/O Streams, btowc(), fputwc(), fwscanf(), mbr?towc(), setlocale()

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