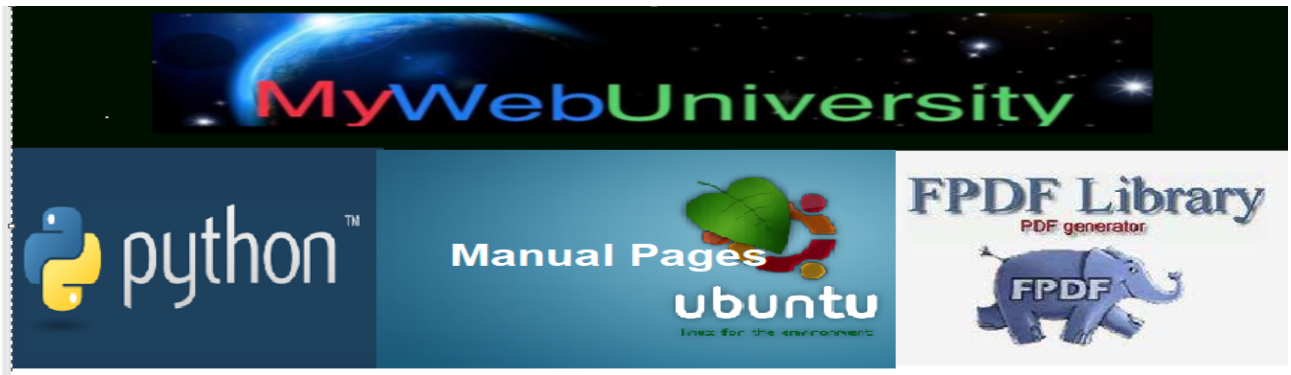

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'register_printf_modifier.3'

$ man register_printf_modifier.3

printf.h(3head) printf.h(3head)

NAME

printf.h, register_printf_specifier, register_printf_modifier, register_printf_type, printf_function, printf_arginfo_size_function, printf_va_arg_function, printf_info, PA_INT, PA_CHAR, PA_WCHAR, PA_STRING, PA_WSTRING, PA_POINTER, PA_FLOAT, PA_DOUBLE, PA_LAST, PA_FLAG_LONG_LONG, PA_FLAG_LONG_DOUBLE, PA_FLAG_LONG, PA_FLAG_SHORT, PA_FLAG_PTR - define

custom behavior for printf-like functions

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <printf.h>
int register_printf_specifier(int spec, printf_function func,
                              printf_arginfo_size_function arginfo);
int register_printf_modifier(const wchar_t *str);
int register_printf_type(printf_va_arg_function fct);
```

Callbacks

```
typedef int printf_function(FILE *stream, const struct printf_info *info,
                            const void *const args[]);
typedef int printf_arginfo_size_function(const struct printf_info *info,
                            size_t n, int argtypes[n], int size[n]);
typedef void printf_va_arg_function(void *mem, va_list *ap);
```

Types

```
struct printf_info {
    int            prec;           // Precision
    int            width;          // Width
    wchar_t        spec;           // Format letter
    unsigned int   is_long_double:1;// L or ll flag
    unsigned int   is_short:1;     // h flag
    unsigned int   is_long:1;      // l flag
    unsigned int   alt:1;          // # flag
    unsigned int   space:1;        // Space flag
    unsigned int   left:1;         // - flag
    unsigned int   showsign:1;     // + flag
    unsigned int   group:1;        // ' flag
    unsigned int   extra:1;        // For special use
    unsigned int   is_char:1;      // hh flag
    unsigned int   wide:1;         // True for wide character streams
    unsigned int   i18n:1;         // I flag
    unsigned int   is_binary128:1;  /* Floating-point argument is
                                       ABI-compatible with
                                       IEC 60559 binary128 */
    unsigned short user;           // Bits for user-installed modifiers
    wchar_t        pad;            // Padding character
};
```

Constants

```
#define PA_FLAG_LONG_LONG    /* ... */
#define PA_FLAG_LONG_DOUBLE  /* ... */
#define PA_FLAG_LONG         /* ... */
#define PA_FLAG_SHORT        /* ... */
#define PA_FLAG_PTR          /* ... */
```

DESCRIPTION

These functions serve to extend and/or modify the behavior of the printf(3) family of func?

tions.

register_printf_specifier()

This function registers a custom conversion specifier for the printf(3) family of functions.

spec The character which will be used as a conversion specifier in the format string.

func Callback function that will be executed by the printf(3) family of functions to for?

mat the input arguments into the output stream.

stream Output stream where the formatted output should be printed. This stream transparently represents the output, even in the case of functions that write to a string.

info Structure that holds context information, including the modifiers specified in the format string. This holds the same contents as in arginfo.

args Array of pointers to the arguments to the printf(3)-like function.

arginfo

Callback function that will be executed by the printf(3) family of functions to know how many arguments should be parsed for the custom specifier and also their types.

info Structure that holds context information, including the modifiers specified in the format string. This holds the same contents as in func.

n Number of arguments remaining to be parsed.

argtypes

This array should be set to define the type of each of the arguments that will be parsed. Each element in the array represents one of the arguments to be parsed, in the same order that they are passed to the printf(3)-like function. Each element should be set to a base type (PA_*) from the enum above, or a custom one, and optionally ORed with an appropriate length modifier (PA_FLAG_*).

The type is determined by using one of the following constants:

PA_INT int.

PA_CHAR

int, cast to char.

PA_WCHAR

wchar_t.

PA_STRING

const char *, a '\0'-terminated string.

PA_WSTRING

const wchar_t *, a wide character L'\0'-terminated string.

PA_POINTER

void *.

PA_FLOAT

float.

PA_DOUBLE

double.

PA_LAST

TODO.

size For user-defined types, the size of the type (in bytes) should also be speci?

fied through this array. Otherwise, leave it unused.

arginfo is called before func, and prepares some information needed to call func.

register_printf_modifier()

TODO

register_printf_type()

TODO

RETURN VALUE

register_printf_specifier(), register_printf_modifier(), and register_printf_type() return

zero on success, or -1 on error.

Callbacks

The callback of type printf_function should return the number of characters written, or -1

on error.

The callback of type printf_arginfo_size_function should return the number of arguments to

be parsed by this specifier. It also passes information about the type of those arguments

to the caller through argtypes. On error, it should return -1.

ERRORS

EINVAL The specifier was not a valid character.

STANDARDS

GNU.

HISTORY

register_printf_function(3) is an older function similar to register_printf_specifier(), and

is now deprecated. That function can't handle user-defined types.

register_printf_specifier() supersedes register_printf_function(3).

EXAMPLES

The following example program registers the 'b' and 'B' specifiers to print integers in bi?

nary format, mirroring rules for other unsigned conversion specifiers like 'x' and 'u'.

This can be used to print in binary prior to C23.

```
/* This code is in the public domain */
#include <err.h>
#include <limits.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/param.h>
#include <printf.h>
#define GROUP_SEP  '\''
struct Printf_Pad {
    char    ch;
    size_t  len;
};
static int b_printf(FILE *stream, const struct printf_info *info,
                    const void *const args[]);
static int b_arginf_sz(const struct printf_info *info,
                    size_t n, int argtypes[n], int size[n]);
static uintmax_t b_value(const struct printf_info *info,
                    const void *arg);
static size_t b_bin_repr(char bin[UINTMAX_WIDTH],
                    const struct printf_info *info, const void *arg);
static size_t b_bin_len(const struct printf_info *info,
                    ptrdiff_t min_len);
static size_t b_pad_len(const struct printf_info *info,
                    ptrdiff_t bin_len);
static ssize_t b_print_prefix(FILE *stream,
                    const struct printf_info *info);
static ssize_t b_pad_zeros(FILE *stream, const struct printf_info *info,
                    ptrdiff_t min_len);
```

```
static ssize_t b_print_number(FILE *stream,
                const struct printf_info *info,
                const char bin[UINTMAX_WIDTH],
                size_t min_len, size_t bin_len);
static char pad_ch(const struct printf_info *info);
static ssize_t pad_spaces(FILE *stream, size_t pad_len);
int
main(void)
{
    if (register_printf_specifier('b', b_printf, b_arginf_sz) == -1)
        err(EXIT_FAILURE, "register_printf_specifier('b', ...)");
    if (register_printf_specifier('B', b_printf, b_arginf_sz) == -1)
        err(EXIT_FAILURE, "register_printf_specifier('B', ...)");
    printf("....----....----....----....----\n");
    printf("%llb;\n", 0x5Ellu);
    printf("%lB;\n", 0x5Elu);
    printf("%b;\n", 0x5Eu);
    printf("%hB;\n", 0x5Eu);
    printf("%hhb;\n", 0x5Eu);
    printf("%jb;\n", (uintmax_t)0x5E);
    printf("%zb;\n", (size_t)0x5E);
    printf("....----....----....----....----\n");
    printf("%#b;\n", 0x5Eu);
    printf("%#B;\n", 0x5Eu);
    printf("....----....----....----....----\n");
    printf("%10b;\n", 0x5Eu);
    printf("%010b;\n", 0x5Eu);
    printf("%.10b;\n", 0x5Eu);
    printf("....----....----....----....----\n");
    printf("%-10B;\n", 0x5Eu);
    printf("....----....----....----....----\n");
    printf("%'B;\n", 0x5Eu);
    printf("....----....----....----....----\n");
```

```
    printf("....----....----....----....----\n");
    printf("%#16.12b;\n", 0xAB);
    printf("%-#'20.12b;\n", 0xAB);
    printf("%#'020B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%#020B;\n", 0xAB);
    printf("%'020B;\n", 0xAB);
    printf("%020B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%#021B;\n", 0xAB);
    printf("%'021B;\n", 0xAB);
    printf("%021B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%#022B;\n", 0xAB);
    printf("%'022B;\n", 0xAB);
    printf("%022B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%#023B;\n", 0xAB);
    printf("%'023B;\n", 0xAB);
    printf("%023B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%-#'19.11b;\n", 0xAB);
    printf("%#'019B;\n", 0xAB);
    printf("%#019B;\n", 0xAB);
    printf("....----....----....----....----\n");
    printf("%'019B;\n", 0xAB);
    printf("%019B;\n", 0xAB);
    printf("%#016b;\n", 0xAB);
    printf("....----....----....----....----\n");
    return 0;
}
static int
b_printf(FILE *stream, const struct printf_info *info,
```

```
        const void *const args[])
{
    char                bin[UINTMAX_WIDTH];
    size_t              min_len, bin_len;
    ssize_t             len, tmp;
    struct Printf_Pad   pad = {0};
    len = 0;
    min_len = b_bin_repr(bin, info, args[0]);
    bin_len = b_bin_len(info, min_len);
    pad.ch = pad_ch(info);
    if (pad.ch == ' ')
        pad.len = b_pad_len(info, bin_len);
    /* Padding with ' ' (right aligned) */
    if ((pad.ch == ' ') && !info->left) {
        tmp = pad_spaces(stream, pad.len);
        if (tmp == EOF)
            return EOF;
        len += tmp;
    }
    /* "0b"/"0B" prefix */
    if (info->alt) {
        tmp = b_print_prefix(stream, info);
        if (tmp == EOF)
            return EOF;
        len += tmp;
    }
    /* Padding with '0' */
    if (pad.ch == '0') {
        tmp = b_pad_zeros(stream, info, min_len);
        if (tmp == EOF)
            return EOF;
        len += tmp;
    }
```

```
    /* Print number (including leading 0s to fill precision) */
    tmp = b_print_number(stream, info, bin, min_len, bin_len);
    if (tmp == EOF)
        return EOF;
    len += tmp;
    /* Padding with ' ' (left aligned) */
    if (info->left) {
        tmp = pad_spaces(stream, pad.len);
        if (tmp == EOF)
            return EOF;
        len += tmp;
    }
    return len;
}
static int
b_arginf_sz(const struct printf_info *info, size_t n, int argtypes[n],
            [[maybe_unused]] int size[n])
{
    if (n < 1)
        return -1;
    if (info->is_long_double)
        argtypes[0] = PA_INT | PA_FLAG_LONG_LONG;
    else if (info->is_long)
        argtypes[0] = PA_INT | PA_FLAG_LONG;
    else
        argtypes[0] = PA_INT;
    return 1;
}
static uintmax_t
b_value(const struct printf_info *info, const void *arg)
{
    if (info->is_long_double)
        return *(const unsigned long long *)arg;
```

```
    if (info->is_long)
        return *(const unsigned long *)arg;
    /* short and char are both promoted to int */
    return *(const unsigned int *)arg;
}
static size_t
b_bin_repr(char bin[UINTMAX_WIDTH],
          const struct printf_info *info, const void *arg)
{
    size_t     min_len;
    uintmax_t  val;
    val = b_value(info, arg);
    bin[0] = '0';
    for (min_len = 0; val; min_len++) {
        bin[min_len] = '0' + (val % 2);
        val >>= 1;
    }
    return MAX(min_len, 1);
}
static size_t
b_bin_len(const struct printf_info *info, ptrdiff_t min_len)
{
    return MAX(info->prec, min_len);
}
static size_t
b_pad_len(const struct printf_info *info, ptrdiff_t bin_len)
{
    ptrdiff_t  pad_len;
    pad_len = info->width - bin_len;
    if (info->alt)
        pad_len -= 2;
    if (info->group)
        pad_len -= (bin_len - 1) / 4;
```

```
    return MAX(pad_len, 0);
}
static ssize_t
b_print_prefix(FILE *stream, const struct printf_info *info)
{
    ssize_t len;
    len = 0;
    if (fputc('0', stream) == EOF)
        return EOF;
    len++;
    if (fputc(info->spec, stream) == EOF)
        return EOF;
    len++;
    return len;
}
static ssize_t
b_pad_zeros(FILE *stream, const struct printf_info *info,
            ptrdiff_t min_len)
{
    ssize_t    len;
    ptrdiff_t  tmp;
    len = 0;
    tmp = info->width - (info->alt * 2);
    if (info->group)
        tmp -= tmp / 5 - !(tmp % 5);
    for (ptrdiff_t i = tmp - 1; i > min_len - 1; i--) {
        if (fputc('0', stream) == EOF)
            return EOF;
        len++;
        if (!info->group || (i % 4))
            continue;
        if (fputc(GROUP_SEP, stream) == EOF)
            return EOF;
```

```
        len++;
    }
    return len;
}
static ssize_t
b_print_number(FILE *stream, const struct printf_info *info,
               const char bin[UINTMAX_WIDTH],
               size_t min_len, size_t bin_len)
{
    ssize_t  len;
    len = 0;
    /* Print leading zeros to fill precision */
    for (size_t i = bin_len - 1; i > min_len - 1; i--) {
        if (fputc('0', stream) == EOF)
            return EOF;
        len++;
        if (!info->group || (i % 4))
            continue;
        if (fputc(GROUP_SEP, stream) == EOF)
            return EOF;
        len++;
    }
    /* Print number */
    for (size_t i = min_len - 1; i < min_len; i--) {
        if (fputc(bin[i], stream) == EOF)
            return EOF;
        len++;
        if (!info->group || (i % 4) || !i)
            continue;
        if (fputc(GROUP_SEP, stream) == EOF)
            return EOF;
        len++;
    }
```

```
    return len;
}
static char
pad_ch(const struct printf_info *info)
{
    if ((info->prec != -1) || (info->pad == ' ') || info->left)
        return ' ';
    return '0';
}
static ssize_t
pad_spaces(FILE *stream, size_t pad_len)
{
    ssize_t  len;
    len = 0;
    for (size_t i = pad_len - 1; i < pad_len; i--) {
        if (fputc(' ', stream) == EOF)
            return EOF;
        len++;
    }
    return len;
}
```

SEE ALSO

asprintf(3), printf(3), wprintf(3)