NAME

pkg_printf, **pkg_fprintf**, **pkg_dprintf**, **pkg_snprintf**, **pkg_asprintf**, **pkg_vprintf**, **pkg_vfprintf**, **pkg_vdprintf**, **pkg_vsnprintf**, **pkg_vasprintf** - formatted output of package data

LIBRARY

library "libpkg"

SYNOPSIS

#include <pkg.h>

int
pkg_printf(*const char * restrict format, ...*);

int
pkg_fprintf(*FILE * restrict stream, const char * restrict format, ...*);

int
pkg_dprintf(*int fd, const char * restrict format, ...*);

int
pkg_snprintf(*char * restrict str, size_t size, const char * restrict format, ...*);

int
pkg_asprintf(*char **ret, const char * restrict format, ...*);

#include <stdarg.h>

int
pkg_vprintf(*const char * restrict format, va_list ap*);

int
pkg_vfprintf(*FILE * restrict stream, const char * restrict format, va_list ap*);

int
pkg_vdprintf(*int fd, const char * restrict format, va_list ap*);

int
pkg_vsnprintf(*char * restrict str, size_t size, const char * restrict format, va_list ap*);

int

pkg_vasprintf(*char **ret, const char * restrict format, va_list ap*);

DESCRIPTION

The **pkg_printf**() family of functions produces output of package data according to a *format* as described below, analogously to the similarly named printf(3) family of functions. The **pkg_printf**() and **pkg_vprintf**() functions write output to stdout, the standard output stream; **pkg_fprintf**() and **pkg_vfprintf**() write output to the given output *stream*; **pkg_dprintf**() and **pkg_vdprintf**() write output to the given file descriptor; **pkg_snprintf**() and **pkg_vsnprintf**() write to the character string *str*; **pkg_asprintf**() and **pkg_vasprintf**() dynamically allocate a new string with malloc(3) to write to.

These functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable-length argument facilities of stdarg(3)) are converted for output.

These functions return the number of characters printed (not including the trailing '\0' used to end output to strings) or a negative value if an output error occurs, except for **pkg_snprintf**() or **pkg_vsnprintf**() which return the number of characters that would have been printed if the *size* were unlimited (again, not including the final '\0').

The **pkg_asprintf**() and **pkg_vasprintf**() functions set **ret* to be a pointer to a buffer sufficiently large to hold the formatted string. This pointer should be passed to free(3) to release the allocated storage when it is no longer needed. If sufficient space cannot be allocated, **pkg_asprintf**() and **pkg_vasprintf**() will return -1 and set *ret* to be a NULL pointer.

The **pkg_snprintf**() and **pkg_vsnprintf**() functions will write at most *size*-1 of the characters printed into the output string (the *size*'th character then gets the terminating '\0'); if the return value is greater than or equal to the *size* argument, the string was too short and some of the printed characters were discarded. The output is always null-terminated.

The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the **%** character. The arguments must correspond properly with the conversion specifier. After the **%**, the following appear in sequence:

- Zero or more of the following flags:

  ? The value should be converted to the "first alternate form".

  For integer valued conversions (**I**, **s**, **t** and **x**) this is a *humanized* form as a floating point

value scaled to the range 0 - 1000 followed by the SI powers-of-10 scale factor. See *SCALE FACTORS*.

For array valued conversions (**A, B**, **C**, **D**, **F**, **G**, **L**, **O**, **U**, **d**, and **r**) generate "0" if there are no items in the array, "1" otherwise.

For formats returning file modes (**Dp** or **Fp**) print the mode in the style of strmode(3).

For boolean valued formats (**dk**, **rk**, **a** and **k**) generate either "yes" or "no" for ‘true’ and ‘false’ respectively.

For the licence logic format **(l)** generate "" (empty), "&" or "|" for types ‘SINGLE’, ‘AND’ and ‘OR’ respectively.

The value should be converted to the "second alternate form".

For the integer valued conversions (**I**, **s**, **t**, **x**) this is a "humanized" form as a floating point value scaled to the range 0 - 1024 followed by the IEE/IEC and SI powers-of-2 scale factor. See *SCALE FACTORS*.

For array valued conversions (**A, B**, **C**, **D**, **F**, **G**, **L**, **O**, **U**, **d**, and **r**) generate the number of items in the array.

For formats returning file modes (**Dp** or **Fp**) print the mode as an octal integer with a leading 0.

For boolean valued formats (**dk**, **rk**, **a** and **k**) generate either "(*)" or "" (empty) for ‘true’ and ‘false’ respectively.

For the licence logic format **(l)** generate "==", "&&" or "||" for types ‘SINGLE’, ‘AND’ and ‘OR’ respectively.

0 (zero) Zero padding. For all integer valued conversions and humanized numbers the converted value is padded on the left with zeros rather than blanks. For string valued conversions, this has no effect and the converted value is padded on the left with blanks.

- A negative field width flag; the converted value is to be left adjusted on the field boundary. The converted value is padded on the right with blanks, rather than on the left with blanks or zeros. Applies to all scalar-valued conversions. "**-**" overrides a "**0**" if both are given.

‘ ’ (space) A blank should be left before a positive number produced by a signed conversion (**I**, **s**, **t**, or **x**).

+ A sign must always be placed before an integer or humanized number produced by a numerical conversion. A "+" overrides a space if both are used.

‘’’ Numerical (integer) conversions should be grouped and separated by thousands using the non-monetary separator returned by localeconv(3). Has no visible effect in the default "C" locale.

- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces (or zeroes, if the zero-padding flag has been given and the conversion supports it) on the left (or spaces on the right, if the left-adjustment flag has been given) to fill out the field width.

- One or two characters that specify the type of conversion to be applied.

- An optional "row format" for array valued conversions (**A, B**, **C**, **D**, **F**, **G**, **L**, **O**, **U**, **d**, and **r**) or the timestamp value conversion (**t**). Which conversion characters are permissible in the row format is context dependent. See the *FORMAT CODES* section for details.

SCALE FACTORS

Humanized number conversions scale the number to lie within the range 1 - 1000 (power of ten conversions using the **?** format modifier) or 1 - 1024 (power of two conversions using the **#** format modifier) and append a scale factor as follows:

The SI power of ten suffixes are

Suffix	Description	Multiplier
	(none)	1
k	kilo	1,000
M	mega	1,000,000
G	giga	1,000,000,000
T	tera	1,000,000,000,000
P	peta	1,000,000,000,000,000
E	exa	1,000,000,000,000,000,000

The IEE/IEC (and now also SI) power of two suffixes are:

Suffix	Description	Multiplier

	(none)	1
Ki	kibi	1,024
Mi	mebi	1,048,576
Gi	gibi	1,073,741,824
Ti	tebi	1,099,511,627,776
Pi	pebi	1,125,899,906,842,624
Ei	exbi	1,152,921,504,606,846,976

FORMAT CODES

Format codes will format the output classified as the type shown in square brackets. **%I** is unique in that it can only be used inside a "row format." All other format codes may be used stand-alone. When used in this fashion they will consume one argument of the indicated type from the function's argument list.

The array valued format codes (**A**, **B**, **C**, **D**, **F**, **G**, **L**, **O**, **U**, **d**, and **r**) and the timestamp format code (**t**) can be followed by a "row format". They will use a default row format (detailed below) if one is not given explicitly.

The row format is bracketed by the character sequences **%{** and **%}** and, for array values only, may be optionally divided into two by the character sequence **%|**. For array values, it contains one or two strings containing any number of a context sensitive subset of format conversions from those described here. For timestamp values it contains any number of format conversion specifiers with meanings as described in strftime(3).

The first or only format string is repeatedly processed for each of the array items in turn. The optional second format string is processed as a separator between each of the array items. If no row format is given, output will be generated according to a default format, detailed below.

Within a "row format" string, you may use any of the single-character non-array valued format codes except for **%S**, but only the two-character format codes which correspond to the parent item and have the same first character. Array valued format codes may not be used within row formats, nor may you embed one "row format" within another. Only one argument, a *struct pkg* * pointer is consumed from the argument list. Thus this is a legal *format* string:

```
"%B%{%n-%v:%Bn%|\n%}"
```

which serves to print out a list of the shared libraries required by the programs within the package, each prefixed by the package name and version.

The conversion specifiers and their meanings are:

%A Annotations [array] *struct pkg* *

Default row format **%A%{%An: %Av\n%|%}**

%An
Annotation tag name [string] *struct pkg_note* *

%Av
Annotation value [string] *struct pkg_note* *

%B Required shared libraries [array] *struct pkg* *

Default row format: **%B%{%Bn\n%|%}**

%Bn Required shared library name [string] *struct pkg_shlib* *

%C Categories [array] *struct pkg* *

Default row format: **%C%{%Cn%|, %}**

%Cn Category name [string] *struct pkg_category* *

%D Directories [array] *struct pkg* *

Default row format: **%D%{%Dn\n%|%}**

%Dg
Directory ownership: group name [string] *struct pkg_dir* *

%Dn
Directory path name [string] *struct pkg_dir* *

%Dp
Directory permissions [mode] *struct pkg_dir* *

%Du
Directory ownership: user name [string] *struct pkg_dir* *

%F Files [array] *struct pkg* *

Default row format: **%F%{%Fn\n%|%}**

%Fg File ownership: group name [string] *struct pkg_file* *

%Fn File path name [string] *struct pkg_file* *

%Fp File permissions [mode] *struct pkg_file* *

%Fs File SHA256 checksum [string] *struct pkg_file* *

%Fu File ownership: user name [string] *struct pkg_file* *

%G Groups [array] *struct pkg* *

Default row format: **%G%{%Gn\n%|%}**

%Gn
Group name [string] *struct pkg_group* *

%I Row counter [integer].

This format code may only be used as part of a "row format."

%L Licenses [array] *struct pkg* *

Default row format: **%L%{%Ln%| %l %}**

%Ln Licence name [string] *struct pkg_license* *

%M Package message [string] *struct pkg* *

%N Repository identity [string] *struct pkg* *

%O Options [array] *struct pkg* *

Default row format: **%O%{%On %Ov\n%|%}**

%On
Option name [string] *struct pkg_option* *

%Ov
Option value [string] *struct pkg_option* *

%Od
Option default value [string] (if known: will produce an empty string if not.) *struct pkg_option* *

%OD
Option description [string] (if known: will produce an empty string if not.) *struct pkg_option* *

%R Repository path - the path relative to the repository root that package may be downloaded from [string]. *struct pkg* *

%S Arbitrary character string [string] *const char* *

%U Users [array] *struct pkg* *

Default row format: **%U%{%Un\n%|%}**

%Un
User name [string] *struct pkg_user* *

%V Old version [string]. Valid only during operations when one version of a package is being replaced by another. *struct pkg* *

%a Autoremove flag [boolean] *struct pkg* *

%b Provided shared libraries [array] *struct pkg* *

Default row format: **%b%{%bn\n%|%}**

%bn Provided shared library name [string] *struct pkg_shlib* *

%c Comment [string] *struct pkg* *

%d Dependencies [array] *struct pkg* *

Default row format: **%d%{%dn-%dv\n%|%}**

%dk Dependency lock status [boolean] *struct pkg_dep* *

%dn Dependency name [string] *struct pkg_dep* *

%do Dependency origin [string] *struct pkg_dep* *

%dv Dependency version [string] *struct pkg_dep* *

%e Description [string] *struct pkg* *

%i Additional information [string] *struct pkg* *

%k Locking status [boolean] *struct pkg* *

%l License logic [licence-logic] *struct pkg* *

%m Maintainer [string] *struct pkg* *

%n Package name [string] *struct pkg* *

%o Origin [string] *struct pkg* *

%p Prefix [string] *struct pkg* *

%r Requirements [array] *struct pkg* *

Default row format: **%r%{%rn-%rv\n%|%}**

%rk Requirement lock status [boolean] *struct pkg_dep* *

%rn Requirement name [string] *struct pkg_dep* *

%ro Requirement origin [string] *struct pkg_dep* *

%rv Requirement version [string] *struct pkg_dep* *

%s Package flat size [integer] *struct pkg* *

%t Installation timestamp [date-time] *struct pkg* *

%u Package checksum [string] *struct pkg* *

%v Package version [string] *struct pkg* *

%w Home page URL [string] *struct pkg* *

%x Package tarball size [integer] *struct pkg* *

%z Package short checksum [string] *struct pkg* *

%% A ‘%’ is written. No argument is converted. The complete conversion specification is ‘%%’.

The decimal point character is defined in the program’s locale (category LC_NUMERIC).

In no case does a non-existent or small field width cause truncation of a numeric field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

ARRAY VALUES

Effective format modifiers:

? First Alternate Form: 0 if the array is empty, 1 if it has any number of elements within it

Second Alternate Form: The number of elements in the array

STRING VALUES

Effective format modifiers:

- Left align

INTEGER VALUES

Effective format modifiers:

- Left align

? First Alternate Form: humanized number (decimal)

Second Alternate Form: humanized number (binary)

0 Zero pad

‘ ’ Blank for plus

+ Explicit + or - sign

‘’ Thousands separator

BOOLEAN VALUES

The two possible values ‘true’ or ‘false’ may be output in one of three different styles: plain; or alternate forms 1 and 2 specified using format modifiers.

Value	Plain (%a)	Alt 1 (%?a)	Alt 2 (%#a)
FALSE	false	no	
TRUE	true	yes	(*)

The second alternate form produces no output for **false**.

Effective format modifiers:

? First Alternate Form

Second Alternate Form

- Left align

FILE MODE VALUES

The file mode is a bitmap representing setid, user, group and other permissions. The plain format prints it as an octal value, for example:

4755

The first alternate form is similar but adds a leading zero:

04755

Whilst the second alternate form produces a string in the style of strmode(3):

-rwsr-xr-x

Note: there is always a space at the end of the strmode(3) output.

Effective format modifiers (all forms):

- Left align

Additionally, when the value is printed as an integer (i.e., plain or alternate form 1), these additional modifiers take effect:

? First Alternate Form: add leading zero to octal integer

0 Zero pad

LICENSE LOGIC VALUES

License-logic is a three-valued type: one of ‘SINGLE’, ‘OR’ or ‘AND’, which shows whether the package is distributed under the terms of a single license, or when there are several applicable licenses, whether these should be treated as alternatives or applied in aggregate. There are three different output styles: plain; or alternate forms 1 and 2 specified using format modifiers.

Logic	**Plain (%l)**	**Alt 1 (%?l)**	**Alt 2 (%#l)**
SINGLE	single		==
OR	or	|	||
AND	and	&	&&

Effective format modifiers:

? First Alternate Form

Second Alternate Form

- Left align

DATE-TIME VALUES

When used outside of a "row format" string may be followed by an optional strftime(3) format, enclosed in **%{** and **%}**, which will be used to format the timestamp. Otherwise the timestamp is printed as an integer value of the number of seconds since the Epoch (00:00:00 UTC, 1 January 1970; see time(3)).

Effective format modifiers:

- Left align

Additionally, when the value is printed as an integer (i.e., without strftime(3) format codes enclosed in **%{** and **%}**, the following format modifiers are also effective:

? First Alternate Form: humanized number (decimal)

Second Alternate Form: humanized number (binary)

0 Zero pad

‘ ’ Blank for plus

+ Explicit + or - sign

‘’’ Thousands separator

EXAMPLES

To print the package installation timestamp in the form "Sunday, July 3, 10:02",

```
#include <pkg.h>
pkg_fprintf(stdout, "%t%{%A, %B %e, %R%}\n", pkg);
```

To print the package name and version, followed by the name and version of all of the packages it depends upon, one per line, each indented by one tab stop:

```
#include <pkg.h>
pkg_printf("%n-%v\n%d%{\t%dn-%dv%|%\n%}\n", pkg, pkg, pkg);
```

Note that the item separator part of the row format is only printed between individual row items. Thus to fill the character array *buf* with a one-line string listing all of the licenses for the package separated by "and" or "or" as appropriate:

```
#include <pkg.h>
char buf[256];
pkg_snprintf(buf, sizeof(buf), "%L%{%Ln%| %l %}", pkg);
```

ERRORS

In addition to the errors documented for the write(2) system call, the **pkg_printf**() family of functions may fail if:

[EILSEQ] An invalid wide character code was encountered.

[ENOMEM] Insufficient storage space is available.

SEE ALSO

pkg_create(3), pkg_repos(3), pkg-keywords(5), pkg-lua-script(5), pkg-repository(5), pkg-script(5),

pkg-triggers(5), pkg.conf(5), pkg(8), pkg-add(8), pkg-alias(8), pkg-annotate(8), pkg-audit(8), pkg-autoremove(8), pkg-check(8), pkg-clean(8), pkg-config(8), pkg-create(8), pkg-delete(8), pkg-fetch(8), pkg-info(8), pkg-install(8), pkg-lock(8), pkg-query(8), pkg-register(8), pkg-repo(8), pkg-rquery(8), pkg-search(8), pkg-set(8), pkg-shell(8), pkg-shlib(8), pkg-ssh(8), pkg-stats(8), pkg-triggers(8), pkg-update(8), pkg-updating(8), pkg-upgrade(8), pkg-version(8), pkg-which(8)

BUGS

The **pkg_printf** family of functions do not correctly handle multibyte characters in the *format* argument.

There is no way to sort the output of array valued items.

SECURITY CONSIDERATIONS

Equivalents to the **sprintf**() and **vsprintf**() functions are not supplied. Instead, use **pkg_snprintf**() to write into a fixed length buffer without danger of overflow.

The **pkg_printf**() family, like the **printf**() family of functions it is modelled on, is also easily misused in a manner allowing malicious users to arbitrarily change a running program’s functionality by either causing the program to print potentially sensitive data "left on the stack", or causing it to generate a memory fault or bus error by dereferencing an invalid pointer.

Programmers are therefore strongly advised to never pass untrusted strings as the *format* argument, as an attacker can put format specifiers in the string to mangle your stack, leading to a possible security hole. This holds true even if the string was built using a function like **snprintf**(), as the resulting string may still contain user-supplied conversion specifiers for later interpolation by **pkg_printf**().

Always use the proper secure idiom:

```
pkg_snprintf(buffer, sizeof(buffer), "%s", string);
```