

---

# **pkgconf Documentation**

*Release 1.1.0*

**pkgconf authors**

**Mar 19, 2018**



---

# Contents

---

<b>1</b>	<b>libpkgconf - an API for managing <i>pkg-config</i> modules</b>	<b>1</b>
1.1	libpkgconf <i>argsplit</i> module . . . . .	1
1.2	libpkgconf <i>audit</i> module . . . . .	1
1.3	libpkgconf <i>cache</i> module . . . . .	2
1.4	libpkgconf <i>client</i> module . . . . .	3
1.5	libpkgconf <i>dependency</i> module . . . . .	7
1.6	libpkgconf <i>fragment</i> module . . . . .	8
1.7	libpkgconf <i>path</i> module . . . . .	10
1.8	libpkgconf <i>pkg</i> module . . . . .	12
1.9	libpkgconf <i>queue</i> module . . . . .	15
1.10	libpkgconf <i>tuple</i> module . . . . .	17
<b>2</b>	<b>Indices and tables</b>	<b>21</b>



---

## libpkgconf - an API for managing *pkg-config* modules

---

### 1.1 libpkgconf *argvsplit* module

This is a lowlevel module which provides parsing of strings into argument vectors, similar to what a shell would do.

void **pkgconf\_argv\_free** (char \*\**argv*)

Frees an argument vector.

**Parameters**

- **argv** (*char\*\**) – The argument vector to free.

**Returns** nothing

int **pkgconf\_argv\_split** (const char \**src*, int \**argc*, char \*\*\**argv*)

Splits a string into an argument vector.

**Parameters**

- **src** (*char\**) – The string to split.
- **argc** (*int\**) – A pointer to an integer to store the argument count.
- **argv** (*char\*\*\**) – A pointer to a pointer for an argument vector.

**Returns** 0 on success, -1 on error.

**Return type** int

### 1.2 libpkgconf *audit* module

The libpkgconf *audit* module contains the functions related to attaching an audit log file to a `pkgconf_client_t` object.

The audit log format is the same as the output generated by the `PKG_CONFIG_LOG` environment variable.

void **pkgconf\_audit\_set\_log** (pkgconf\_client\_t \*client, FILE \*auditf)

Sets the audit log file pointer on *client* to *auditf*. The callee is responsible for closing any previous log files.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to modify.
- **auditf** (*FILE\**) – The file pointer for the already open log file.

**Returns** nothing

void **pkgconf\_audit\_log** (pkgconf\_client\_t \*client, const char \*format, ...)

Logs a message to the opened audit log (if any).

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object the log message is for.
- **format** (*char\**) – The format string to use for the log messages.

**Returns** nothing

void **pkgconf\_audit\_log\_dependency** (pkgconf\_client\_t \*client, const pkgconf\_pkg\_t \*dep, const pkgconf\_dependency\_t \*depnode)

Convenience function which logs a dependency node to the opened audit log (if any).

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object the log message is for.
- **dep** (*pkgconf\_pkg\_t\**) – The dependency package object being logged.
- **depnode** (*pkgconf\_dependency\_t\**) – The dependency object itself being logged.

**Returns** nothing

## 1.3 libpkgconf *cache* module

The libpkgconf *cache* module manages a package/module object cache, allowing it to avoid loading duplicate copies of a package/module.

A cache is tied to a specific pkgconf client object, so package objects should not be shared across threads.

pkgconf\_pkg\_t \***pkgconf\_cache\_lookup** (const pkgconf\_client\_t \*client, const char \*id)

Looks up a package in the cache given an *id* atom, such as `gtk+-3.0` and returns the already loaded version if present.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to access.
- **id** (*char\**) – The package atom to look up in the client object's cache.

**Returns** A package object if present, else NULL.

**Return type** pkgconf\_pkg\_t \*

void **pkgconf\_cache\_add** (pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*pkg)

Adds an entry for the package to the package cache. The cache entry must be removed if the package is freed.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to modify.
- **pkg** (*pkgconf\_pkg\_t\**) – The package object to add to the client object's cache.

**Returns** nothing

void **pkgconf\_cache\_remove** (pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*pkg)  
 Deletes a package from the client object's package cache.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client object to modify.
- **pkg** (pkgconf\_pkg\_t \*) – The package object to remove from the client object's cache.

**Returns** nothing

void **pkgconf\_cache\_free** (pkgconf\_client\_t \*client)  
 Releases all resources related to a client object's package cache. This function should only be called to clear a client object's package cache, as it may release any package in the cache.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client object to modify.

## 1.4 libpkgconf *client* module

The libpkgconf *client* module implements the *pkgconf\_client\_t* “client” object. Client objects store all necessary state for libpkgconf allowing for multiple instances to run in parallel.

Client objects are not thread safe, in other words, a client object should not be shared across thread boundaries.

void **pkgconf\_client\_init** (pkgconf\_client\_t \*client, pkgconf\_error\_handler\_func\_t error\_handler)  
 Initialise a pkgconf client object.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client to initialise.
- **error\_handler** (pkgconf\_error\_handler\_func\_t) – An optional error handler to use for logging errors.
- **error\_handler\_data** (void\*) – user data passed to optional error handler

**Returns** nothing

pkgconf\_client\_t\* **pkgconf\_client\_new** (pkgconf\_error\_handler\_func\_t error\_handler)  
 Allocate and initialise a pkgconf client object.

**Parameters**

- **error\_handler** (pkgconf\_error\_handler\_func\_t) – An optional error handler to use for logging errors.
- **error\_handler\_data** (void\*) – user data passed to optional error handler

**Returns** A pkgconf client object.

**Return type** pkgconf\_client\_t\*

void **pkgconf\_client\_deinit** (pkgconf\_client\_t \*client)  
 Release resources belonging to a pkgconf client object.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client to deinitialise.

**Returns** nothing

void **pkgconf\_client\_free** (pkgconf\_client\_t \**client*)

Release resources belonging to a pkgconf client object and then free the client object itself.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client to deinitialise and free.

**Returns** nothing

const char \***pkgconf\_client\_get\_sysroot\_dir** (const pkgconf\_client\_t \**client*)

Retrieves the client's sysroot directory (if any).

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object being accessed.

**Returns** A string containing the sysroot directory or NULL.

**Return type** const char \*

void **pkgconf\_client\_set\_sysroot\_dir** (pkgconf\_client\_t \**client*, const char \**sysroot\_dir*)

Sets or clears the sysroot directory on a client object. Any previous sysroot directory setting is automatically released if one was previously set.

Additionally, the global tuple \$(pc\_sysrootdir) is set as appropriate based on the new setting.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object being modified.
- **sysroot\_dir** (*char\**) – The sysroot directory to set or NULL to unset.

**Returns** nothing

const char \***pkgconf\_client\_get\_buildroot\_dir** (const pkgconf\_client\_t \**client*)

Retrieves the client's buildroot directory (if any).

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object being accessed.

**Returns** A string containing the buildroot directory or NULL.

**Return type** const char \*

void **pkgconf\_client\_set\_buildroot\_dir** (pkgconf\_client\_t \**client*, const char \**buildroot\_dir*)

Sets or clears the buildroot directory on a client object. Any previous buildroot directory setting is automatically released if one was previously set.

Additionally, the global tuple \$(pc\_top\_builddir) is set as appropriate based on the new setting.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object being modified.
- **buildroot\_dir** (*char\**) – The buildroot directory to set or NULL to unset.

**Returns** nothing

bool **pkgconf\_error** (const pkgconf\_client\_t \**client*, const char \**format*, ...)

Report an error to a client-registered error handler.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to report the error to.
- **format** (*char\**) – A printf-style format string to use for formatting the error message.

**Returns** true if the error handler processed the message, else false.



**Return type** bool

bool **pkgconf\_warn** (const pkgconf\_client\_t \**client*, const char \**format*, ...)

Report an error to a client-registered warn handler.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to report the error to.
- **format** (*char\**) – A printf-style format string to use for formatting the warning message.

**Returns** true if the warn handler processed the message, else false.

**Return type** bool

bool **pkgconf\_trace** (const pkgconf\_client\_t \**client*, const char \**format*, ...)

Report a message to a client-registered trace handler.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to report the trace message to.
- **format** (*char\**) – A printf-style format string to use for formatting the trace message.

**Returns** true if the trace handler processed the message, else false.

**Return type** bool

bool **pkgconf\_default\_error\_handler** (const char \**msg*, const pkgconf\_client\_t \**client*, const void \**data*)

The default pkgconf error handler.

**Parameters**

- **msg** (*char\**) – The error message to handle.
- **client** (*pkgconf\_client\_t\**) – The client object the error originated from.
- **data** (*void\**) – An opaque pointer to extra data associated with the client for error handling.

**Returns** true (the function does nothing to process the message)

**Return type** bool

unsigned int **pkgconf\_client\_get\_flags** (const pkgconf\_client\_t \**client*)

Retrieves resolver-specific flags associated with a client object.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to retrieve the resolver-specific flags from.

**Returns** a bitfield of resolver-specific flags

**Return type** uint

void **pkgconf\_client\_set\_flags** (pkgconf\_client\_t \**client*, unsigned int *flags*)

Sets resolver-specific flags associated with a client object.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to set the resolver-specific flags on.

**Returns** nothing

const char \***pkgconf\_client\_get\_prefix\_varname** (const pkgconf\_client\_t \**client*)

Retrieves the name of the variable that should contain a module's prefix. In some cases, it is necessary to override this variable to allow proper path relocation.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to retrieve the prefix variable name from.

**Returns** the prefix variable name as a string

**Return type** const char \*

void **pkgconf\_client\_set\_prefix\_varname** (pkgconf\_client\_t \**client*, const char \**prefix\_varname*)

Sets the name of the variable that should contain a module's prefix. If the variable name is NULL, then the default variable name (`prefix`) is used.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to set the prefix variable name on.
- **prefix\_varname** (*char\**) – The prefix variable name to set.

**Returns** nothing

**pkgconf\_client\_get\_warn\_handler** (const pkgconf\_client\_t \**client*)

Returns the warning handler if one is set, else NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to get the warn handler from.

**Returns** a function pointer to the warn handler or NULL

**pkgconf\_client\_set\_warn\_handler** (pkgconf\_client\_t \**client*, pkgconf\_error\_handler\_func\_t *conf\_error\_handler\_func*, pkg\_warn\_handler\_t *warn\_handler*, void \**warn\_handler\_data*)

Sets a warn handler on a client object or uninstalls one if set to NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to set the warn handler on.
- **warn\_handler** (*pkgconf\_error\_handler\_func\_t*) – The warn handler to set.
- **warn\_handler\_data** (*void\**) – Optional data to associate with the warn handler.

**Returns** nothing

**pkgconf\_client\_get\_error\_handler** (const pkgconf\_client\_t \**client*)

Returns the error handler if one is set, else NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to get the error handler from.

**Returns** a function pointer to the error handler or NULL

**pkgconf\_client\_set\_error\_handler** (pkgconf\_client\_t \**client*, pkgconf\_error\_handler\_func\_t *error\_handler*, void \**error\_handler\_data*)

Sets a warn handler on a client object or uninstalls one if set to NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to set the error handler on.
- **error\_handler** (*pkgconf\_error\_handler\_func\_t*) – The error handler to set.

- **error\_handler\_data** (*void\**) – Optional data to associate with the error handler.

**Returns** nothing

**pkgconf\_client\_get\_trace\_handler** (const pkgconf\_client\_t \**client*)

Returns the error handler if one is set, else NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to get the error handler from.

**Returns** a function pointer to the error handler or NULL

**pkgconf\_client\_set\_trace\_handler** (pkgconf\_client\_t \**client*, pkgconf\_error\_handler\_func\_t *trace\_handler*, void \**trace\_handler\_data*)

Sets a warn handler on a client object or uninstalls one if set to NULL.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object to set the error handler on.
- **trace\_handler** (*pkgconf\_error\_handler\_func\_t*) – The error handler to set.
- **trace\_handler\_data** (*void\**) – Optional data to associate with the error handler.

**Returns** nothing

## 1.5 libpkgconf *dependency* module

The *dependency* module provides support for building *dependency lists* (the basic component of the overall *dependency graph*) and *dependency nodes* which store dependency information.

pkgconf\_dependency\_t \***pkgconf\_dependency\_add** (pkgconf\_list\_t \**list*, const char \**package*, const char \**version*, pkgconf\_pkg\_comparator\_t *compare*)

Adds a parsed dependency to a dependency list as a dependency node.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client object that owns the package this dependency list belongs to.
- **list** (*pkgconf\_list\_t\**) – The dependency list to add a dependency node to.
- **package** (*char\**) – The package *atom* to set on the dependency node.
- **version** (*char\**) – The package *version* to set on the dependency node.
- **compare** (*pkgconf\_pkg\_comparator\_t*) – The comparison operator to set on the dependency node.

**Returns** A dependency node.

**Return type** pkgconf\_dependency\_t \*

void **pkgconf\_dependency\_append** (pkgconf\_list\_t \**list*, pkgconf\_dependency\_t \**tail*)

Adds a dependency node to a pre-existing dependency list.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The dependency list to add a dependency node to.
- **tail** (*pkgconf\_dependency\_t\**) – The dependency node to add to the tail of the dependency list.

**Returns** nothing

void **pkgconf\_dependency\_free** (pkgconf\_list\_t \*list)  
Release a dependency list and its child dependency nodes.

**Parameters**

- **list** (pkgconf\_list\_t \*) – The dependency list to release.

**Returns** nothing

void **pkgconf\_dependency\_parse\_str** (pkgconf\_list\_t \*deplist\_head, const char \*depends)  
Parse a dependency declaration into a dependency list. Commas are counted as whitespace to allow for constructs such as @SUBSTVAR@, zlib being processed into , zlib.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client object that owns the package this dependency list belongs to.
- **deplist\_head** (pkgconf\_list\_t \*) – The dependency list to populate with dependency nodes.
- **depends** (char \*) – The dependency data to parse.

**Returns** nothing

void **pkgconf\_dependency\_parse** (const pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*pkg, pkgconf\_list\_t \*deplist, const char \*depends)  
Preprocess dependency data and then process that dependency declaration into a dependency list. Commas are counted as whitespace to allow for constructs such as @SUBSTVAR@, zlib being processed into , zlib.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The client object that owns the package this dependency list belongs to.
- **pkg** (pkgconf\_pkg\_t \*) – The package object that owns this dependency list.
- **deplist** (pkgconf\_list\_t \*) – The dependency list to populate with dependency nodes.
- **depends** (char \*) – The dependency data to parse.

**Returns** nothing

## 1.6 libpkgconf *fragment* module

The *fragment* module provides low-level management and rendering of fragment lists. A *fragment list* contains various *fragments* of text (such as -I /usr/include) in a manner which is composable, mergeable and reorderable.

void **pkgconf\_fragment\_add** (const pkgconf\_client\_t \*client, pkgconf\_list\_t \*list, const char \*string)  
Adds a *fragment* of text to a *fragment list*, possibly modifying the fragment if a sysroot is set.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The pkgconf client being accessed.
- **list** (pkgconf\_list\_t \*) – The fragment list.
- **string** (char \*) – The string of text to add as a fragment to the fragment list.

**Returns** nothing

bool **pkgconf\_fragment\_has\_system\_dir** (const pkgconf\_client\_t \*client, const pkgconf\_fragment\_t \*frag)

Checks if a *fragment* contains a *system path*. System paths are detected at compile time and optionally overridden by the `PKG_CONFIG_SYSTEM_INCLUDE_PATH` and `PKG_CONFIG_SYSTEM_LIBRARY_PATH` environment variables.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object the fragment belongs to.
- **frag** (*pkgconf\_fragment\_t\**) – The fragment being checked.

**Returns** true if the fragment contains a system path, else false

**Return type** bool

void **pkgconf\_fragment\_copy** (const pkgconf\_client\_t \*client, pkgconf\_list\_t \*list, const pkgconf\_fragment\_t \*base, bool is\_private)

Copies a *fragment* to another *fragment list*, possibly removing a previous copy of the *fragment* in a process known as *mergeback*.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client being accessed.
- **list** (*pkgconf\_list\_t\**) – The list the fragment is being added to.
- **base** (*pkgconf\_fragment\_t\**) – The fragment being copied.
- **is\_private** (*bool*) – Whether the fragment list is a *private* fragment list (static linking).

**Returns** nothing

void **pkgconf\_fragment\_filter** (const pkgconf\_client\_t \*client, pkgconf\_list\_t \*dest, pkgconf\_list\_t \*src, pkgconf\_fragment\_filter\_func\_t filter\_func)

Copies a *fragment list* to another *fragment list* which match a user-specified filtering function.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client being accessed.
- **dest** (*pkgconf\_list\_t\**) – The destination list.
- **src** (*pkgconf\_list\_t\**) – The source list.
- **filter\_func** (*pkgconf\_fragment\_filter\_func\_t*) – The filter function to use.
- **data** (*void\**) – Optional data to pass to the filter function.

**Returns** nothing

size\_t **pkgconf\_fragment\_render\_len** (const pkgconf\_list\_t \*list)

Calculates the required memory to store a *fragment list* when rendered as a string.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The *fragment list* being rendered.
- **escape** (*bool*) – Whether or not to escape special shell characters.

**Returns** the amount of bytes required to represent the *fragment list* when rendered

**Return type** size\_t

void **pkgconf\_fragment\_render\_buf** (const pkgconf\_list\_t \*list, char \*buf, size\_t buflen)

Renders a *fragment list* into a buffer.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The *fragment list* being rendered.
- **buf** (*char\**) – The buffer to render the fragment list into.
- **buflen** (*size\_t*) – The length of the buffer.
- **escape** (*bool*) – Whether or not to escape special shell characters.

**Returns** nothing

char \***pkgconf\_fragment\_render** (const pkgconf\_list\_t \**list*)  
Allocate memory and render a *fragment list* into it.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The *fragment list* being rendered.
- **escape** (*bool*) – Whether or not to escape special shell characters.

**Returns** An allocated string containing the rendered *fragment list*.

**Return type** char \*

void **pkgconf\_fragment\_delete** (pkgconf\_list\_t \**list*, pkgconf\_fragment\_t \**node*)  
Delete a *fragment node* from a *fragment list*.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The *fragment list* to delete from.
- **node** (*pkgconf\_fragment\_t\**) – The *fragment node* to delete.

**Returns** nothing

void **pkgconf\_fragment\_free** (pkgconf\_list\_t \**list*)  
Delete an entire *fragment list*.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The *fragment list* to delete.

**Returns** nothing

bool **pkgconf\_fragment\_parse** (const pkgconf\_client\_t \**client*, pkgconf\_list\_t \**list*, pkgconf\_list\_t \**vars*, const char \**value*)  
Parse a string into a *fragment list*.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client being accessed.
- **list** (*pkgconf\_list\_t\**) – The *fragment list* to add the fragment entries to.
- **vars** (*pkgconf\_list\_t\**) – A list of variables to use for variable substitution.
- **value** (*char\**) – The string to parse into fragments.

**Returns** true on success, false on parse error

## 1.7 libpkgconf *path* module

The *path* module provides functions for manipulating lists of paths in a cross-platform manner. Notably, it is used by the *pkgconf client* to parse the `PKG_CONFIG_PATH`, `PKG_CONFIG_LIBDIR` and related environment variables.

void **pkgconf\_path\_add** (const char \**text*, pkgconf\_list\_t \**dirlist*)  
Adds a path node to a path list. If the path is already in the list, do nothing.

**Parameters**

- **text** (*char\**) – The path text to add as a path node.
- **dirlist** (*pkgconf\_list\_t\**) – The path list to add the path node to.
- **filter** (*bool*) – Whether to perform duplicate filtering.

**Returns** nothing

size\_t **pkgconf\_path\_split** (const char \**text*, pkgconf\_list\_t \**dirlist*)  
Splits a given text input and inserts paths into a path list.

**Parameters**

- **text** (*char\**) – The path text to split and add as path nodes.
- **dirlist** (*pkgconf\_list\_t\**) – The path list to have the path nodes added to.
- **filter** (*bool*) – Whether to perform duplicate filtering.

**Returns** number of path nodes added to the path list**Return type** size\_t

size\_t **pkgconf\_path\_build\_from\_environ** (const char \**environ*, const char \**fallback*, pkgconf\_list\_t \**dirlist*)

Adds the paths specified in an environment variable to a path list. If the environment variable is not set, an optional default set of paths is added.

**Parameters**

- **environ** (*char\**) – The environment variable to look up.
- **fallback** (*char\**) – The fallback paths to use if the environment variable is not set.
- **dirlist** (*pkgconf\_list\_t\**) – The path list to add the path nodes to.
- **filter** (*bool*) – Whether to perform duplicate filtering.

**Returns** number of path nodes added to the path list**Return type** size\_t

bool **pkgconf\_path\_match\_list** (const char \**path*, const pkgconf\_list\_t \**dirlist*)

Checks whether a path has a matching prefix in a path list.

**Parameters**

- **path** (*char\**) – The path to check against a path list.
- **dirlist** (*pkgconf\_list\_t\**) – The path list to check the path against.

**Returns** true if the path list has a matching prefix, otherwise false**Return type** bool

void **pkgconf\_path\_free** (pkgconf\_list\_t \**dirlist*)

Releases any path nodes attached to the given path list.

**Parameters**

- **dirlist** (*pkgconf\_list\_t\**) – The path list to clean up.

**Returns** nothing

bool **pkgconf\_path\_relocate** (char \**buf*, size\_t *buflen*)

Relocates a path, possibly calling normpath() or cygwin\_conv\_path() on it.

**Parameters**

- **buf** (*char\**) – The path to relocate.
- **buflen** (*size\_t*) – The buffer length the path is contained in.

**Returns** true on success, false on error

**Return type** bool

## 1.8 libpkgconf *pkg* module

The *pkg* module provides dependency resolution services and the overall *.pc* file parsing routines.

void **pkgconf\_pkg\_dir\_list\_build**(pkgconf\_client\_t \**client*)

Bootstraps the package search paths. If the `PKGCONF_PKG_PKG_ENV_ONLY` flag is set on the client, then only the `PKG_CONFIG_PATH` environment variable will be used, otherwise both the `PKG_CONFIG_PATH` and `PKG_CONFIG_LIBDIR` environment variables will be used.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to bootstrap.

**Returns** nothing

pkgconf\_pkg\_t \***pkgconf\_pkg\_new\_from\_file**(const pkgconf\_client\_t \**client*, const char \**filename*, FILE \**f*)

Parse a *.pc* file into a `pkgconf_pkg_t` object structure.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **filename** (*char\**) – The filename of the package file (including full path).
- **f** (*FILE\**) – The file object to read from.

**Returns** A `pkgconf_pkg_t` object which contains the package data.

**Return type** `pkgconf_pkg_t*`

void **pkgconf\_pkg\_free**(pkgconf\_client\_t \**client*, pkgconf\_pkg\_t \**pkg*)

Releases all releases for a given `pkgconf_pkg_t` object.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The client which owns the `pkgconf_pkg_t` object, *pkg*.
- **pkg** (*pkgconf\_pkg\_t\**) – The package to free.

**Returns** nothing

pkgconf\_pkg\_t \***pkgconf\_pkg\_ref**(const pkgconf\_client\_t \**client*, pkgconf\_pkg\_t \**pkg*)

Adds an additional reference to the package object.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object which owns the package being referenced.
- **pkg** (*pkgconf\_pkg\_t\**) – The package object being referenced.

**Returns** The package itself with an incremented reference count.

**Return type** `pkgconf_pkg_t*`



void **pkgconf\_pkg\_unref** (pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*pkg)

Releases a reference on the package object. If the reference count is 0, then also free the package.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The pkgconf client object which owns the package being dereferenced.
- **pkg** (pkgconf\_pkg\_t \*) – The package object being dereferenced.

**Returns** nothing

pkgconf\_pkg\_t \***pkgconf\_scan\_all** (pkgconf\_client\_t \*client, void \*data, pkgconf\_pkg\_iteration\_func\_t func)

Iterates over all packages found in the *package directory list*, running `func` on them. If `func` returns true, then stop iteration and return the last iterated package.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The pkgconf client object to use for dependency resolution.
- **data** (void \*) – An opaque pointer to data to provide the iteration function with.
- **func** (pkgconf\_pkg\_iteration\_func\_t) – A function which is called for each package to determine if the package matches, always return `false` to iterate over all packages.

**Returns** A package object reference if one is found by the scan function, else NULL.

**Return type** pkgconf\_pkg\_t \*

pkgconf\_pkg\_t \***pkgconf\_pkg\_find** (pkgconf\_client\_t \*client, const char \*name)

Search for a package.

**Parameters**

- **client** (pkgconf\_client\_t \*) – The pkgconf client object to use for dependency resolution.
- **name** (char \*) – The name of the package *atom* to use for searching.

**Returns** A package object reference if the package was found, else NULL.

**Return type** pkgconf\_pkg\_t \*

int **pkgconf\_compare\_version** (const char \*a, const char \*b)

Compare versions using RPM version comparison rules as described in the LSB.

**Parameters**

- **a** (char \*) – The first version to compare in the pair.
- **b** (char \*) – The second version to compare in the pair.

**Returns** -1 if the first version is greater, 0 if both versions are equal, 1 if the second version is greater.

**Return type** int

pkgconf\_pkg\_t \***pkgconf\_builtin\_pkg\_get** (const char \*name)

Looks up a built-in package. The package should not be freed or dereferenced.

**Parameters**

- **name** (char \*) – An atom corresponding to a built-in package to search for.

**Returns** the built-in package if present, else NULL.

**Return type** pkgconf\_pkg\_t \*

const char \***pkgconf\_pkg\_get\_comparator** (const pkgconf\_dependency\_t \**pkgdep*)  
Returns the comparator used in a depgraph dependency node as a string.

**Parameters**

- **pkgdep** (*pkgconf\_dependency\_t \**) – The depgraph dependency node to return the comparator for.

**Returns** A string matching the comparator or "???".

**Return type** char \*

pkgconf\_pkg\_comparator\_t **pkgconf\_pkg\_comparator\_lookup\_by\_name** (const char \**name*)  
Look up the appropriate comparator bytecode in the comparator set (defined in `pkg.c`, see `pkgconf_pkg_comparator_names` and `pkgconf_pkg_comparator_impls`).

**Parameters**

- **name** (*char \**) – The comparator to look up by *name*.

**Returns** The comparator bytecode if found, else `PKGCONF_CMP_ANY`.

**Return type** pkgconf\_pkg\_comparator\_t

pkgconf\_pkg\_t \***pkgconf\_pkg\_verify\_dependency** (pkgconf\_client\_t \**client*, pkgconf\_dependency\_t \**pkgdep*, unsigned int \**eflags*)

Verify a `pkgconf_dependency_t` node in the depgraph. If the dependency is solvable, return the appropriate `pkgconf_pkg_t` object, else `NULL`.

**Parameters**

- **client** (*pkgconf\_client\_t \**) – The `pkgconf` client object to use for dependency resolution.
- **pkgdep** (*pkgconf\_dependency\_t \**) – The dependency graph node to solve.
- **eflags** (*uint \**) – An optional pointer that, if set, will be populated with an error code from the resolver.

**Returns** On success, the appropriate `pkgconf_pkg_t` object to solve the dependency, else `NULL`.

**Return type** pkgconf\_pkg\_t \*

unsigned int **pkgconf\_pkg\_verify\_graph** (pkgconf\_client\_t \**client*, pkgconf\_pkg\_t \**root*, int *depth*)  
Verify the graph dependency nodes are satisfiable by walking the tree using `pkgconf_pkg_traverse()`.

**Parameters**

- **client** (*pkgconf\_client\_t \**) – The `pkgconf` client object to use for dependency resolution.
- **root** (*pkgconf\_pkg\_t \**) – The root entry in the package dependency graph which should contain the top-level dependencies to resolve.
- **depth** (*int*) – The maximum allowed depth for dependency resolution.

**Returns** On success, `PKGCONF_PKG_ERRF_OK` (0), else an error code.

**Return type** unsigned int

unsigned int **pkgconf\_pkg\_traverse** (pkgconf\_client\_t \**client*, pkgconf\_pkg\_t \**root*, pkgconf\_pkg\_traverse\_func\_t *func*, void \**data*, int *maxdepth*)  
Walk and resolve the dependency graph up to *maxdepth* levels.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **root** (*pkgconf\_pkg\_t\**) – The root of the dependency graph.
- **func** (*pkgconf\_pkg\_traverse\_func\_t*) – A traversal function to call for each resolved node in the dependency graph.
- **data** (*void\**) – An opaque pointer to data to be passed to the traversal function.
- **maxdepth** (*int*) – The maximum depth to walk the dependency graph for. -1 means infinite recursion.

**Returns** PKGCONF\_PKG\_ERRF\_OK on success, else an error code.

**Return type** unsigned int

int **pkgconf\_pkg\_cflags** (*pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*root, pkgconf\_list\_t \*list, int maxdepth*)

Walks a dependency graph and extracts relevant CFLAGS fragments.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **root** (*pkgconf\_pkg\_t\**) – The root of the dependency graph.
- **list** (*pkgconf\_list\_t\**) – The fragment list to add the extracted CFLAGS fragments to.
- **maxdepth** (*int*) – The maximum allowed depth for dependency resolution. -1 means infinite recursion.

**Returns** PKGCONF\_PKG\_ERRF\_OK if successful, otherwise an error code.

**Return type** unsigned int

int **pkgconf\_pkg\_libs** (*pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*root, pkgconf\_list\_t \*list, int maxdepth*)

Walks a dependency graph and extracts relevant LIBS fragments.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **root** (*pkgconf\_pkg\_t\**) – The root of the dependency graph.
- **list** (*pkgconf\_list\_t\**) – The fragment list to add the extracted LIBS fragments to.
- **maxdepth** (*int*) – The maximum allowed depth for dependency resolution. -1 means infinite recursion.

**Returns** PKGCONF\_PKG\_ERRF\_OK if successful, otherwise an error code.

**Return type** unsigned int

## 1.9 libpkgconf *queue* module

The *queue* module provides an interface that allows easily building a dependency graph from an arbitrary set of dependencies. It also provides support for doing “preflight” checks on the entire dependency graph prior to working with it.

Using the *queue* module functions is the recommended way of working with dependency graphs.

void **pkgconf\_queue\_push** (pkgconf\_list\_t \*list, const char \*package)  
Pushes a requested dependency onto the dependency resolver's queue.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – the dependency resolution queue to add the package request to.
- **package** (*char\**) – the dependency atom requested

**Returns** nothing

bool **pkgconf\_queue\_compile** (pkgconf\_client\_t \*client, pkgconf\_pkg\_t \*world, pkgconf\_list\_t \*list)  
Compile a dependency resolution queue into a dependency resolution problem if possible, otherwise report an error.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **world** (*pkgconf\_pkg\_t\**) – The designated root of the dependency graph.
- **list** (*pkgconf\_list\_t\**) – The list of dependency requests to consider.

**Returns** true if the built dependency resolution problem is consistent, else false

**Return type** bool

void **pkgconf\_queue\_free** (pkgconf\_list\_t \*list)  
Release any memory related to a dependency resolution queue.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The dependency resolution queue to release.

**Returns** nothing

void **pkgconf\_queue\_apply** (pkgconf\_client\_t \*client, pkgconf\_list\_t \*list, pkgconf\_queue\_apply\_func\_t func, int maxdepth, void \*data)  
Attempt to compile a dependency resolution queue into a dependency resolution problem, then attempt to solve the problem and feed the solution to a callback function if a complete dependency graph is found.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **list** (*pkgconf\_list\_t\**) – The list of dependency requests to consider.
- **func** (*pkgconf\_queue\_apply\_func\_t*) – The callback function to call if a solution is found by the dependency resolver.
- **maxdepth** (*int*) – The maximum allowed depth for the dependency resolver. A depth of -1 means unlimited.
- **data** (*void\**) – An opaque pointer which is passed to the callback function.

**Returns** true if the dependency resolver found a solution, otherwise false.

**Return type** bool

void **pkgconf\_queue\_validate** (pkgconf\_client\_t \**client*, pkgconf\_list\_t \**list*, pkgconf\_queue\_apply\_func\_t *func*, int *maxdepth*, void \**data*)

Attempt to compile a dependency resolution queue into a dependency resolution problem, then attempt to solve the problem.

#### Parameters

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to use for dependency resolution.
- **list** (*pkgconf\_list\_t\**) – The list of dependency requests to consider.
- **maxdepth** (*int*) – The maximum allowed depth for the dependency resolver. A depth of -1 means unlimited.

**Returns** true if the dependency resolver found a solution, otherwise false.

**Return type** bool

## 1.10 libpkgconf *tuple* module

The *tuple* module provides key-value mappings backed by a linked list. The key-value mapping is mainly used for variable substitution when parsing .pc files.

There are two sets of mappings: a *pkgconf\_pkg\_t* specific mapping, and a *global* mapping. The *tuple* module provides convenience wrappers for managing the *global* mapping, which is attached to a given client object.

void **pkgconf\_tuple\_add\_global** (pkgconf\_client\_t \**client*, const char \**key*, const char \**value*)

Defines a global variable, replacing the previous declaration if one was set.

#### Parameters

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to modify.
- **key** (*char\**) – The key for the mapping (variable name).
- **value** (*char\**) – The value for the mapped entry.

**Returns** nothing

void **pkgconf\_tuple\_find\_global** (const pkgconf\_client\_t \**client*, const char \**key*)

Looks up a global variable.

#### Parameters

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to access.
- **key** (*char\**) – The key or variable name to look up.

**Returns** the contents of the variable or NULL

**Return type** char \*

void **pkgconf\_tuple\_free\_global** (pkgconf\_client\_t \**client*)

Delete all global variables associated with a pkgconf client object.

#### Parameters

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to modify.

**Returns** nothing

void **pkgconf\_tuple\_define\_global** (pkgconf\_client\_t \**client*, const char \**kv*)

Parse and define a global variable.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to modify.
- **kv** (*char\**) – The variable in the form of `key=value`.

**Returns** nothing

`pkgconf_tuple_t *pkgconf_tuple_add`(const `pkgconf_client_t *client`, `pkgconf_list_t *list`, const `char *key`, const `char *value`, bool *parse*)

Optionally parse and then define a variable.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to access.
- **list** (*pkgconf\_list\_t\**) – The variable list to add the new variable to.
- **key** (*char\**) – The name of the variable being added.
- **value** (*char\**) – The value of the variable being added.
- **parse** (*bool*) – Whether or not to parse the value for variable substitution.

**Returns** a variable object

**Return type** `pkgconf_tuple_t*`

`char *pkgconf_tuple_find`(const `pkgconf_client_t *client`, `pkgconf_list_t *list`, const `char *key`)

Look up a variable in a variable list.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to access.
- **list** (*pkgconf\_list\_t\**) – The variable list to search.
- **key** (*char\**) – The variable name to search for.

**Returns** the value of the variable or NULL

**Return type** `char*`

`char *pkgconf_tuple_parse`(const `pkgconf_client_t *client`, `pkgconf_list_t *vars`, const `char *value`)

Parse an expression for variable substitution.

**Parameters**

- **client** (*pkgconf\_client\_t\**) – The pkgconf client object to access.
- **list** (*pkgconf\_list\_t\**) – The variable list to search for variables (along side the global variable list).
- **value** (*char\**) – The `key=value` string to parse.

**Returns** the variable data with any variables substituted

**Return type** `char*`

`void pkgconf_tuple_free_entry`(`pkgconf_tuple_t *tuple`, `pkgconf_list_t *list`)

Deletes a variable object, removing it from any variable lists and releasing any memory associated with it.

**Parameters**

- **tuple** (*pkgconf\_tuple\_t\**) – The variable object to release.
- **list** (*pkgconf\_list\_t\**) – The variable list the variable object is attached to.

**Returns** nothing

void **pkgconf\_tuple\_free** (pkgconf\_list\_t \**list*)

Deletes a variable list and any variables attached to it.

**Parameters**

- **list** (*pkgconf\_list\_t\**) – The variable list to delete.

**Returns** nothing





## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `search`



## P

- pkgconf\_argv\_free (C function), 1
- pkgconf\_argv\_split (C function), 1
- pkgconf\_audit\_log (C function), 2
- pkgconf\_audit\_log\_dependency (C function), 2
- pkgconf\_audit\_set\_log (C function), 1
- pkgconf\_builtin\_pkg\_get (C function), 13
- pkgconf\_cache\_add (C function), 2
- pkgconf\_cache\_free (C function), 3
- pkgconf\_cache\_lookup (C function), 2
- pkgconf\_cache\_remove (C function), 3
- pkgconf\_client\_deinit (C function), 3
- pkgconf\_client\_free (C function), 3
- pkgconf\_client\_get\_buildroot\_dir (C function), 4
- pkgconf\_client\_get\_error\_handler (C function), 6
- pkgconf\_client\_get\_flags (C function), 5
- pkgconf\_client\_get\_prefix\_varname (C function), 5
- pkgconf\_client\_get\_sysroot\_dir (C function), 4
- pkgconf\_client\_get\_trace\_handler (C function), 7
- pkgconf\_client\_get\_warn\_handler (C function), 6
- pkgconf\_client\_init (C function), 3
- pkgconf\_client\_new (C function), 3
- pkgconf\_client\_set\_buildroot\_dir (C function), 4
- pkgconf\_client\_set\_error\_handler (C function), 6
- pkgconf\_client\_set\_flags (C function), 5
- pkgconf\_client\_set\_prefix\_varname (C function), 6
- pkgconf\_client\_set\_sysroot\_dir (C function), 4
- pkgconf\_client\_set\_trace\_handler (C function), 7
- pkgconf\_client\_set\_warn\_handler (C function), 6
- pkgconf\_compare\_version (C function), 13
- pkgconf\_default\_error\_handler (C function), 5
- pkgconf\_dependency\_add (C function), 7
- pkgconf\_dependency\_append (C function), 7
- pkgconf\_dependency\_free (C function), 8
- pkgconf\_dependency\_parse (C function), 8
- pkgconf\_dependency\_parse\_str (C function), 8
- pkgconf\_error (C function), 4
- pkgconf\_fragment\_add (C function), 8
- pkgconf\_fragment\_copy (C function), 9
- pkgconf\_fragment\_delete (C function), 10
- pkgconf\_fragment\_filter (C function), 9
- pkgconf\_fragment\_free (C function), 10
- pkgconf\_fragment\_has\_system\_dir (C function), 8
- pkgconf\_fragment\_parse (C function), 10
- pkgconf\_fragment\_render (C function), 10
- pkgconf\_fragment\_render\_buf (C function), 9
- pkgconf\_fragment\_render\_len (C function), 9
- pkgconf\_path\_add (C function), 10
- pkgconf\_path\_build\_from\_envron (C function), 11
- pkgconf\_path\_free (C function), 11
- pkgconf\_path\_match\_list (C function), 11
- pkgconf\_path\_relocate (C function), 11
- pkgconf\_path\_split (C function), 11
- pkgconf\_pkg\_cflags (C function), 15
- pkgconf\_pkg\_comparator\_lookup\_by\_name (C function), 14
- pkgconf\_pkg\_dir\_list\_build (C function), 12
- pkgconf\_pkg\_find (C function), 13
- pkgconf\_pkg\_free (C function), 12
- pkgconf\_pkg\_get\_comparator (C function), 14
- pkgconf\_pkg\_libs (C function), 15
- pkgconf\_pkg\_new\_from\_file (C function), 12
- pkgconf\_pkg\_ref (C function), 12
- pkgconf\_pkg\_traverse (C function), 14
- pkgconf\_pkg\_unref (C function), 12
- pkgconf\_pkg\_verify\_dependency (C function), 14
- pkgconf\_pkg\_verify\_graph (C function), 14
- pkgconf\_queue\_apply (C function), 16
- pkgconf\_queue\_compile (C function), 16
- pkgconf\_queue\_free (C function), 16
- pkgconf\_queue\_push (C function), 16
- pkgconf\_queue\_validate (C function), 16
- pkgconf\_scan\_all (C function), 13
- pkgconf\_trace (C function), 5
- pkgconf\_tuple\_add (C function), 18
- pkgconf\_tuple\_add\_global (C function), 17
- pkgconf\_tuple\_define\_global (C function), 17
- pkgconf\_tuple\_find (C function), 18
- pkgconf\_tuple\_find\_global (C function), 17

pkgconf\_tuple\_free (C function), 18  
pkgconf\_tuple\_free\_entry (C function), 18  
pkgconf\_tuple\_free\_global (C function), 17  
pkgconf\_tuple\_parse (C function), 18  
pkgconf\_warn (C function), 5