# RFIDentify Client

1.0

Generated by Doxygen 1.6.3

Thu May 13 13:47:41 2010

# Contents

# Chapter 1

# Todo List

**File avahi_dns_handler.c** Multiple RFID servers are not handled correctly.

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

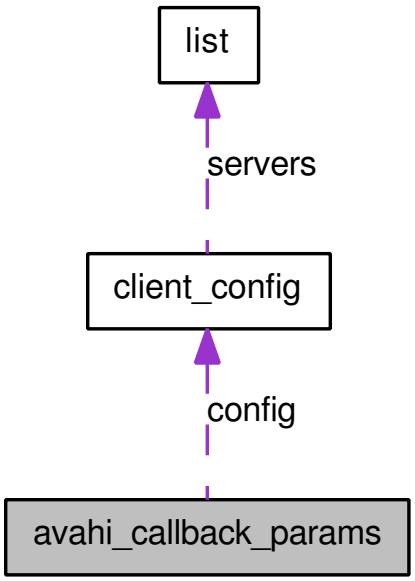
# Data Structure Documentation

## 4.1 avahi_callback_params Struct Reference

Aggregation of various parameters to Avahi callbacks.

`#include <client.h>`

Collaboration diagram for avahi_callback_params:




### Data Fields

- AvahiSimplePoll * poll

  *Current poll object.*

- AvahiClient * client

  *Currently working client.*

- struct client_config * config
  *Client configuration status.*

### 4.1.1 Detailed Description

Aggregation of various parameters to Avahi callbacks.

Definition at line 44 of file client.h.

The documentation for this struct was generated from the following file:

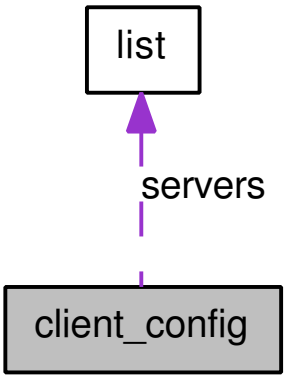
- client/client.h

## 4.2 client_config Struct Reference

Defines a client configuration context.

`#include <client.h>`

Collaboration diagram for client_config:




### Data Fields

- char * config_file
  *Location of configuration file.*
- char * mode
  *Current mode of client.*
- char * action
  *Server REST action.*
- char * last_tag
  *Last tag encountered by reader.*
- list * servers
  *List of rfid_server_info structures.*
- pthread_mutex_t lock
  *Protects modifications, especially last_tag.*

### 4.2.1 Detailed Description

Defines a client configuration context.

Definition at line 33 of file client.h.

The documentation for this struct was generated from the following file:

- client/client.h

## 4.3 list Struct Reference

`#include <list.h>`

### Data Fields

- void * **car**
- void * **cdr**

### 4.3.1 Detailed Description

standardization:

a list ends with a cell that looks like:

```
     +---+---+
---> | v | 0 |
     +---+---+
```

and empty list then looks like:

```
+---+---+
| 0 | 0 |
+---+---+
```

it should look like this:

```
0
```

but this doesnt work, because we would need to be able to modify the pointers but of course they are passed by value...

so this implementation needs to be careful on edge cases, checking when there are fewer than 2 elements and whatnot

Definition at line 53 of file list.h.

The documentation for this struct was generated from the following file:

- client/list.h

## 4.4 reader Struct Reference

Completely identifies a currently connected RFID reader.

`#include <reader.h>`

### Data Fields

- int **connected**
- struct ftdi_context * **ftdic**
- struct usb_device * **dev**

### 4.4.1 Detailed Description

Completely identifies a currently connected RFID reader.

Definition at line 67 of file reader.h.

The documentation for this struct was generated from the following file:

- client/reader.h

## 4.5 rfid_server_info Struct Reference

Defines a relevant configuration of an RFID server.

`#include <client.h>`

### Data Fields

- pthread_mutex_t lock

  *Protects modifications across threads.*

- char * url

  *RFID server url.*

- uint16_t port

  *RFID server port.*

- int stable

  *True if the URL will not change. Eg. false if it may be modified by mDNS.*

- char * name

  *Unique name of the given server. Provided by mDNS.*

### 4.5.1 Detailed Description

Defines a relevant configuration of an RFID server.

Definition at line 20 of file client.h.

### 4.5.2 Field Documentation

#### 4.5.2.1 pthread_mutex_t lock

Protects modifications across threads.

As the url may be updated at any time, we must lock it before modification or reading.

Definition at line 25 of file client.h.

The documentation for this struct was generated from the following file:

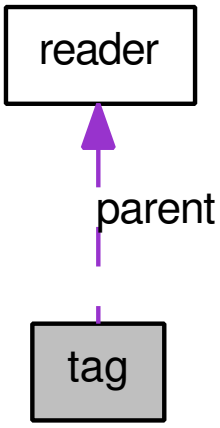
- client/client.h

## 4.6 tag Struct Reference

An RFID identification.

`#include <reader.h>`

Collaboration diagram for tag:




### Data Fields

- char * **id**
- struct reader * parent

  *Unique ID.*

### 4.6.1 Detailed Description

An RFID identification.

Definition at line 82 of file reader.h.

The documentation for this struct was generated from the following file:

- client/reader.h

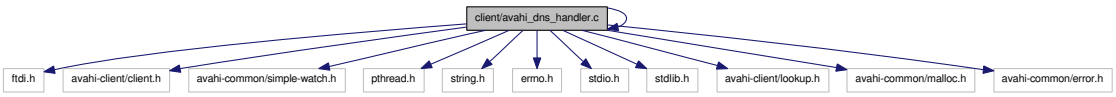
# Chapter 5

# File Documentation

## 5.1 client/avahi_dns_handler.c File Reference

```
#include "client.h"
#include <ftdi.h>
#include <avahi-client/client.h>
#include <avahi-common/simple-watch.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <avahi-client/lookup.h>
#include <avahi-common/malloc.h>
#include <avahi-common/error.h>
```

Include dependency graph for avahi_dns_handler.c:




This graph shows which files directly or indirectly include this file:


### Functions

- void resolve_callback (AvahiServiceResolver *r, AvahiIfIndex interface, AvahiProtocol protocol, AvahiResolverEvent event, const char *name, const char *type, const char *domain, const char

∗host_name, const AvahiAddress ∗address, uint16_t port, AvahiStringList ∗txt, AvahiLookupResultFlags flags, void ∗userdata)

- void browse_callback (AvahiServiceBrowser ∗b, AvahiIfIndex interface, AvahiProtocol protocol, AvahiBrowserEvent event, const char ∗name, const char ∗type, const char ∗domain, AvahiLookupResultFlags flags, void ∗userdata)
- void client_callback (AvahiClient ∗c, AvahiClientState state, void ∗userdata)
- void ∗ avahi_function (void ∗args)

  *mDNS handling loop which looks for RFIDentify servers. Look for avahi_dns_handler.c*

### 5.1.1 Detailed Description

The Avahi mDNS service discovery system is initialized and listens for RFID servers. Upon recieving an updated, the server configuration information is similarly updated.

**Author**

Willi Ballenthin

**Date**

Spring, 2010

**Todo**

Multiple RFID servers are not handled correctly.

Definition in file avahi_dns_handler.c.

### 5.1.2 Function Documentation

#### 5.1.2.1 void∗ avahi_function (void ∗ *args*)

mDNS handling loop which looks for RFIDentify servers. Look for avahi_dns_handler.c

PThread. Initializes an mDNS Avahi service, and listens for relevant services. Updates the server info configuration as necessary.

**Parameters**

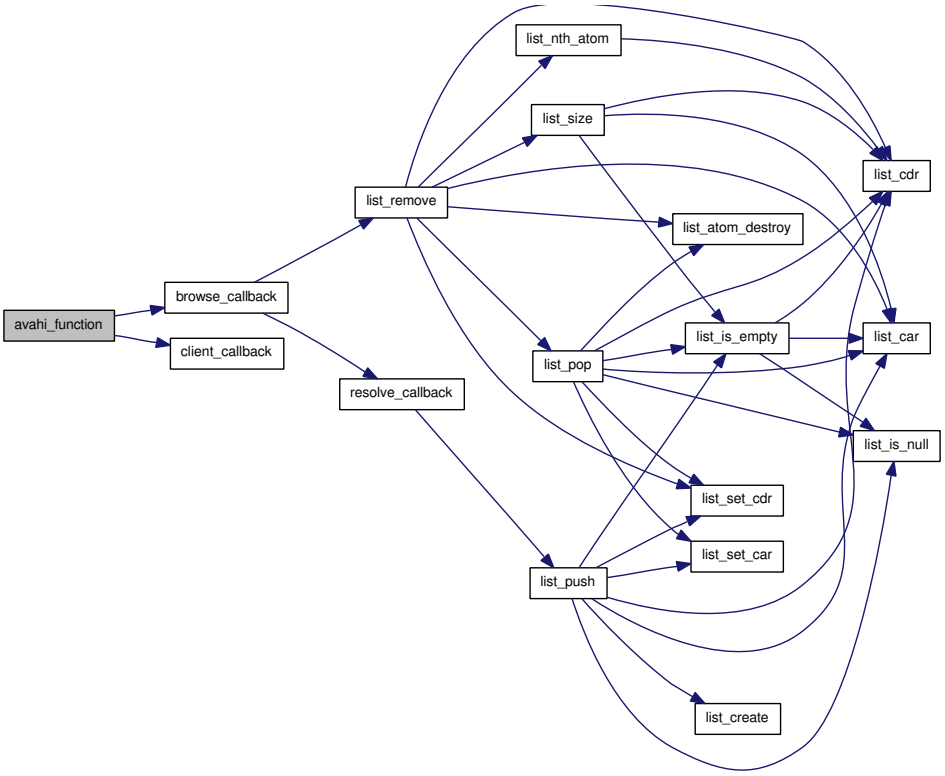
***args*** struct client_config, Relevant configuration settings.

**Returns**

Should not return, if so, error.

Definition at line 225 of file avahi_dns_handler.c.

Here is the call graph for this function:




#### 5.1.2.2 void browse_callback (AvahiServiceBrowser * *b*, AvahiIfIndex *interface*, AvahiProtocol *protocol*, AvahiBrowserEvent *event*, const char * *name*, const char * *type*, const char * *domain*, AvahiLookupResultFlags *flags*, void * *userdata*)

Called whenever a new services becomes available on the LAN or is removed from the LAN.

**See also**

simple_poll_reader.c in libavahi API

**Parameters**

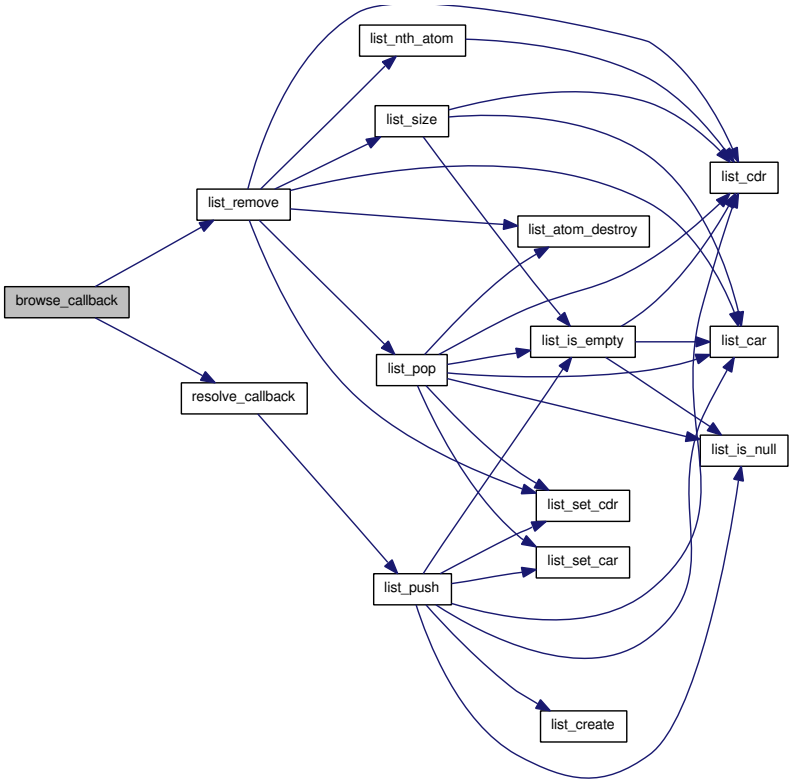
***userdata*** An avahi_callback_params pointer containing relevant data.

this is a hack, rather than coming up with a good loop

we just assume this can only match one. not ideal. we only compare against the name, with no consideration of domain, which is bad. should come up with a hash.

Definition at line 118 of file avahi_dns_handler.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 5.1.2.3 void client_callback (AvahiClient * *c*, AvahiClientState *state*, void * *userdata*)

Called whenever the client or server state changes.

**Parameters**

***c*** The current client.

***state*** The current state.

***userdata*** Relevant data, including configuration settings.

Definition at line 203 of file avahi_dns_handler.c.

Here is the caller graph for this function:

#### 5.1.2.4 void resolve_callback (AvahiServiceResolver * *r*, AvahiIfIndex *interface*, AvahiProtocol *protocol*, AvahiResolverEvent *event*, const char * *name*, const char * *type*, const char * *domain*, const char * *host_name*, const AvahiAddress * *address*, uint16_t *port*, AvahiStringList * *txt*, AvahiLookupResultFlags *flags*, void * *userdata*)

Called whenever a service has been resolved successfully or timed out.

**See also**

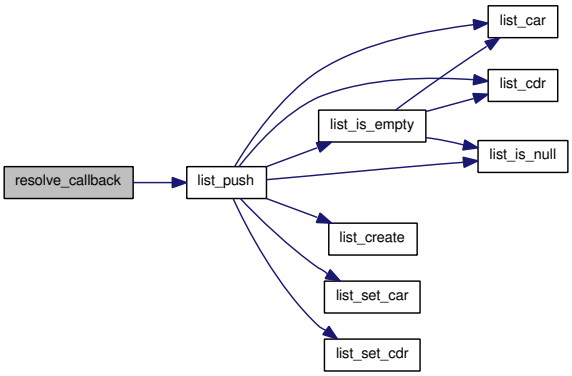
simple_poll_reader.c in libavahi API

**Parameters**

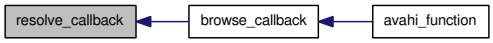
***userdata*** An avahi_callback_params pointer containing relevant data.

Definition at line 37 of file avahi_dns_handler.c.

Here is the call graph for this function:



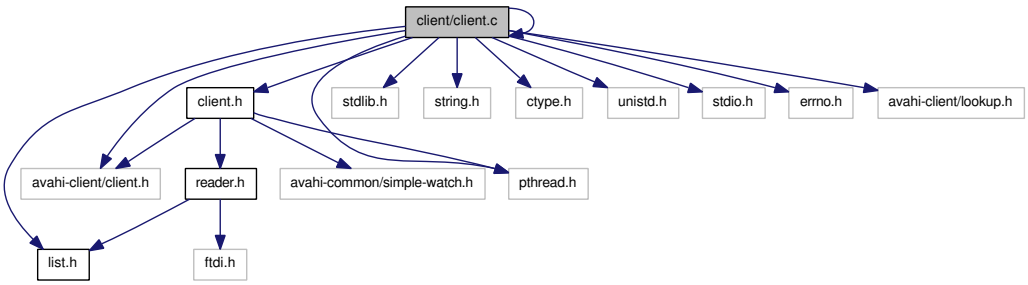

Here is the caller graph for this function:



## 5.2 client/client.c File Reference

```
#include "client.h"
#include "config.h"
#include "list.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>
#include <errno.h>
#include <avahi-client/client.h>
#include <avahi-client/lookup.h>
```

Include dependency graph for client.c:




This graph shows which files directly or indirectly include this file:


### Functions

- int read_config (struct client_config *client_config, const char *filename)

  *Configures a client given a configuration file.*

- list * read_config_servers (struct client_config *client_config)

  *Configures static servers from a configuration file.*

- int **main** (int argc, char **argv)

### 5.2.1 Detailed Description

This application reads RFID tags and queries a server with the tag ID. The server is discovered using Avahi/mDNS.

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file client.c.

### 5.2.2 Function Documentation

#### 5.2.2.1 int read_config (struct client_config * *client_config*, const char * *filename*)

Configures a client given a configuration file.

Configures a client given a configuration file.

**Parameters**

***client_config*** The destination configuration.

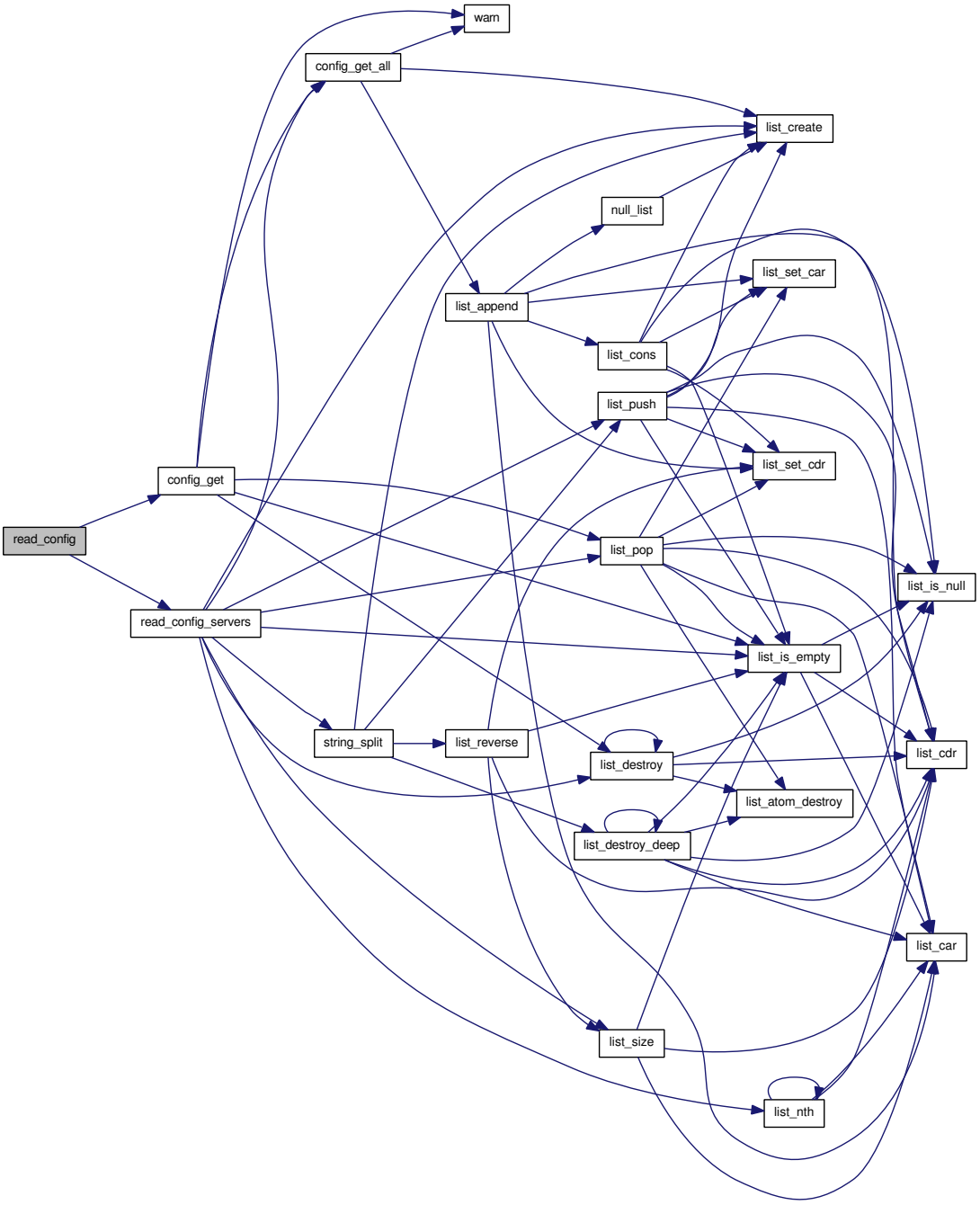
***filename*** The file from which to read the config.

**Returns**

0 on success. non-zero otherwise.

Definition at line 35 of file client.c.

Here is the call graph for this function:




#### 5.2.2.2 list* read_config_servers (struct client_config * *client_config*)

Configures static servers from a configuration file.

Configures static servers from a configuration file.

### Parameters

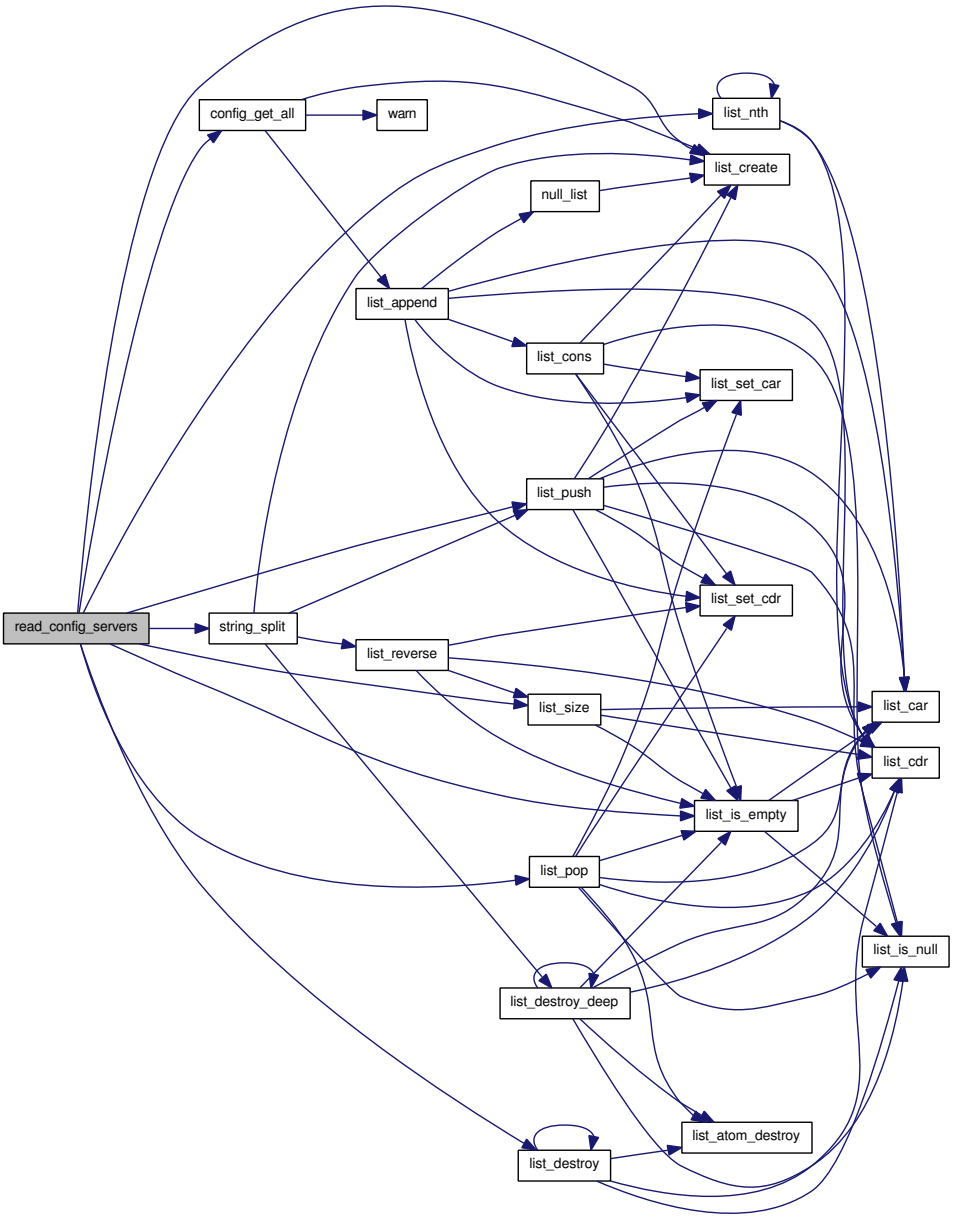
***client_config*** The config, including filename, from which to determine the servers.

### Returns

A list of servers, or NULL on failure.

Definition at line 79 of file client.c.

Here is the call graph for this function:

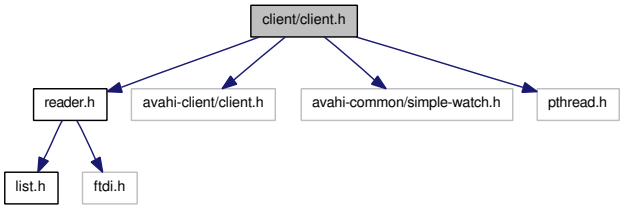


Here is the caller graph for this function:

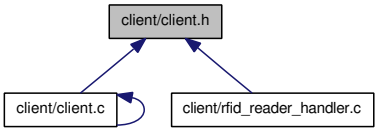
## 5.3 client/client.h File Reference

```
#include "reader.h"
#include <avahi-client/client.h>
#include <avahi-common/simple-watch.h>
#include <pthread.h>
```

Include dependency graph for client.h:




This graph shows which files directly or indirectly include this file:




### Data Structures

- struct rfid_server_info

  *Defines a relevant configuration of an RFID server.*

- struct client_config

  *Defines a client configuration context.*

- struct avahi_callback_params

  *Aggregation of various parameters to Avahi callbacks.*

### Functions

- void * reader_function (void *args)

  *RFID reading loop which polls for identifications. Look for rfid_reader_handler.c.*

- void * avahi_function (void *args)

  *mDNS handling loop which looks for RFIDentify servers. Look for avahi_dns_handler.c*

- int read_config (struct client_config *client_config, const char *filename)

  *Configures a client given a configuration file.*

- list * read_config_servers (struct client_config *client_config)

  *Configures static servers from a configuration file.*

### 5.3.1 Detailed Description

This application reads RFID tags and queries a server with the tag ID. The server is discovered using Avahi/mDNS.

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file client.h.

### 5.3.2 Function Documentation

#### 5.3.2.1 void* avahi_function (void * *args*)

mDNS handling loop which looks for RFIDentify servers. Look for avahi_dns_handler.c

PThread. Initializes an mDNS Avahi service, and listens for relevant services. Updates the server info configuration as necessary.

**Parameters**

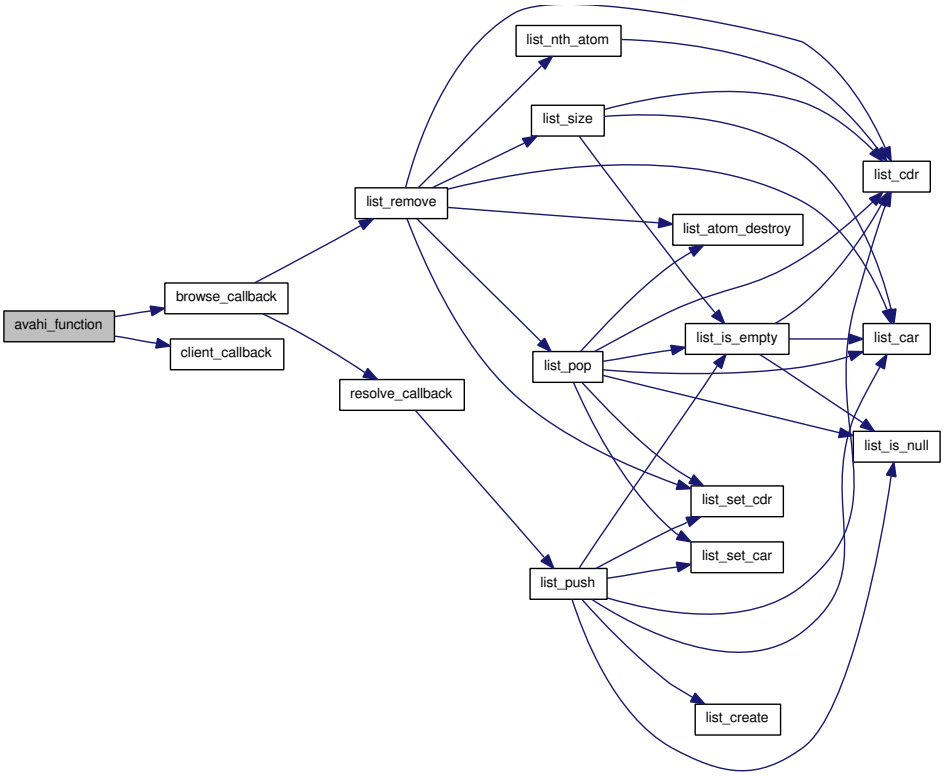
***args*** struct client_config, Relevant configuration settings.

**Returns**

Should not return, if so, error.

Definition at line 225 of file avahi_dns_handler.c.

Here is the call graph for this function:




#### 5.3.2.2 int read_config (struct client_config * *client_config*, const char * *filename*)

Configures a client given a configuration file.

Configures a client given a configuration file.

**Parameters**

***client_config*** The destination configuration.

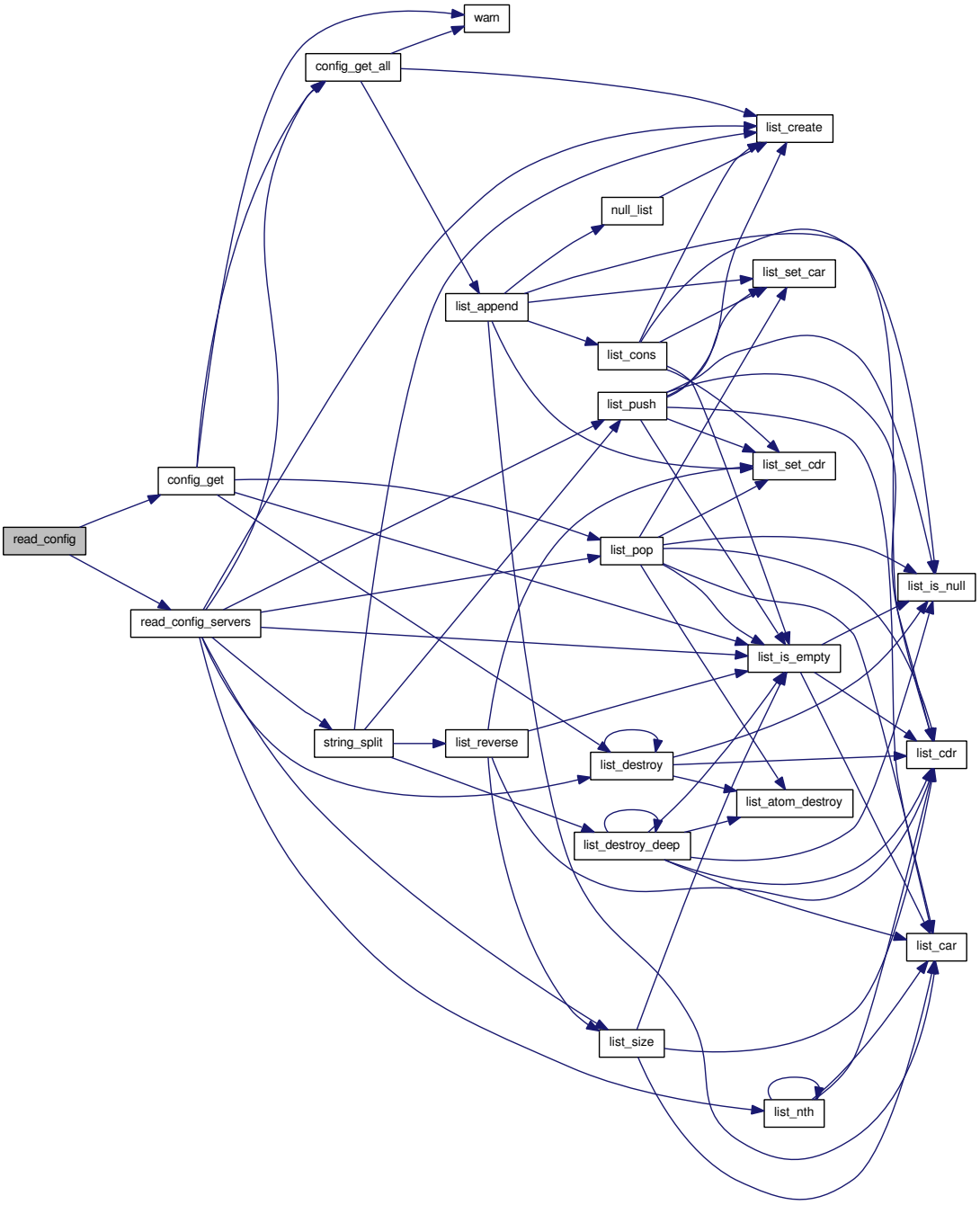
***filename*** The file from which to read the config.

**Returns**

0 on success. non-zero otherwise.

Definition at line 35 of file client.c.

Here is the call graph for this function:




#### 5.3.2.3 list* read_config_servers (struct client_config * *client_config*)

Configures static servers from a configuration file.

Configures static servers from a configuration file.

### Parameters

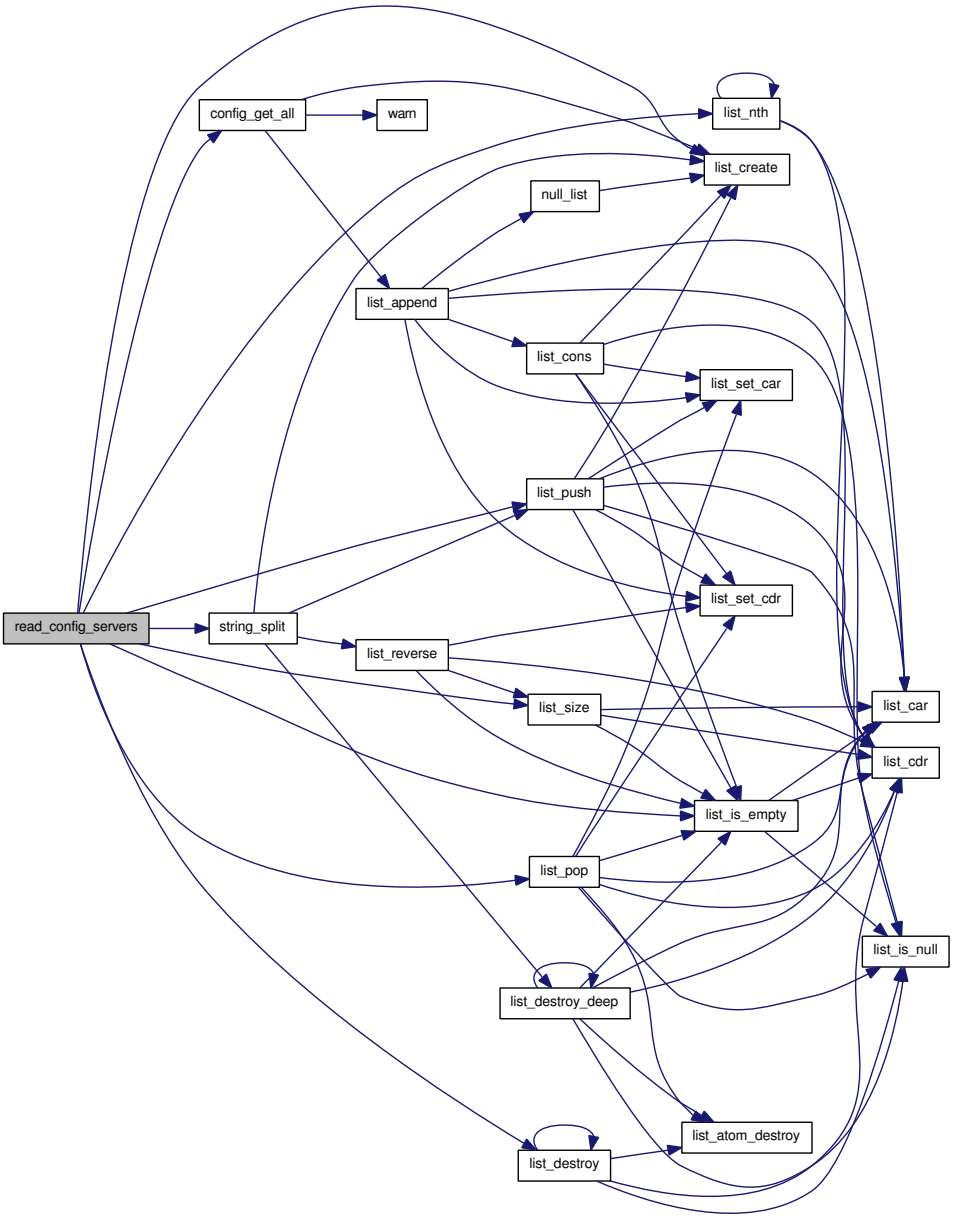
*client_config* The config, including filename, from which to determine the servers.

### Returns

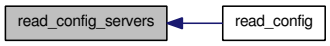
A list of servers, or NULL on failure.

Definition at line 79 of file client.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 5.3.2.4 void* reader_function (void * *args*)

RFID reading loop which polls for identifications. Look for rfid_reader_handler.c.

PThread. Initializes an RFID device, if it exists, and begins a tag reading loop.

**See also**

SLEEP_BTWN_SEARCH

**Parameters**

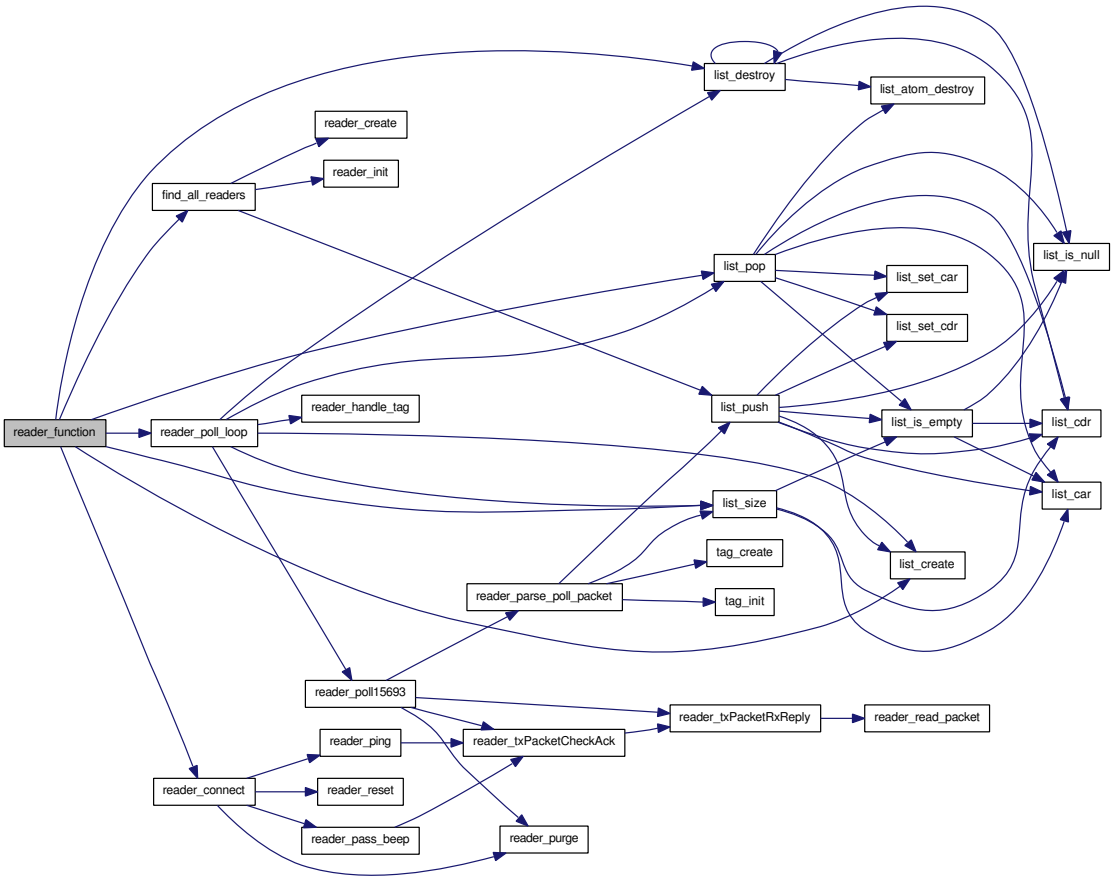
***args*** struct client_config - The current configuration of the client.

**Returns**

Should not return, if so, error. abstract RFID differences from this implementation

Definition at line 164 of file rfid_reader_handler.c.

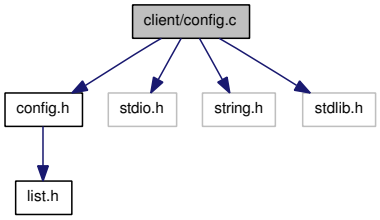
Here is the call graph for this function:



## 5.4 client/config.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for config.c:




### Defines

- #define DEBUG 1

  *Whether to print out debug messages to STDOUT.*

- #define WARN 1

  *Whether to print out warning messages to STDOUT.*

### Functions

- void warn (const char *warning)
- void debug (const char *debug)
- char * config_get (const char *filename, const char *key)

  *Get a value from a configuration file identified by a filename.*

- list * config_get_all (const char *filename, const char *key)

  *Gets all relevant values from a configuration file identifed by a filename.*

### 5.4.1 Detailed Description

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file config.c.

## 5.4.2 Function Documentation

### 5.4.2.1 char* config_get (const char * *filename*, const char * *key*)

Get a value from a configuration file identified by a filename.

Given a file, and key, return the first matching config value.

**Parameters**

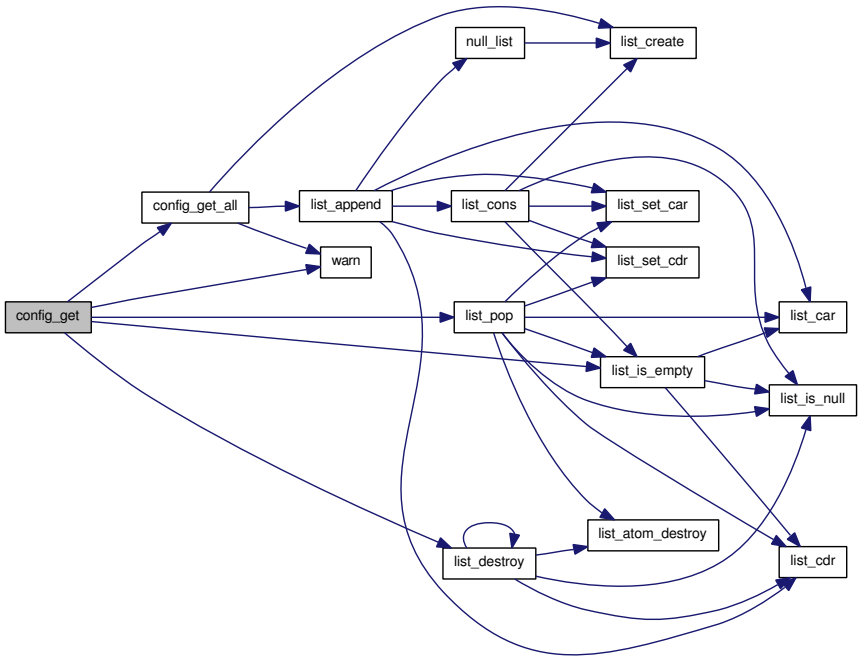
***filename*** The config file.
***key*** The key of which the value to find.

**Returns**

A newly allocated string, or NULL on failure.

Definition at line 50 of file config.c.

Here is the call graph for this function:




Here is the caller graph for this function:


### 5.4.2.2 list* config_get_all (const char * *filename*, const char * *key*)

Gets all relevant values from a configuration file identifed by a filename.

Given a filename and key, return all matching config values.

**Parameters**

***filename*** The filename of the configuration file.

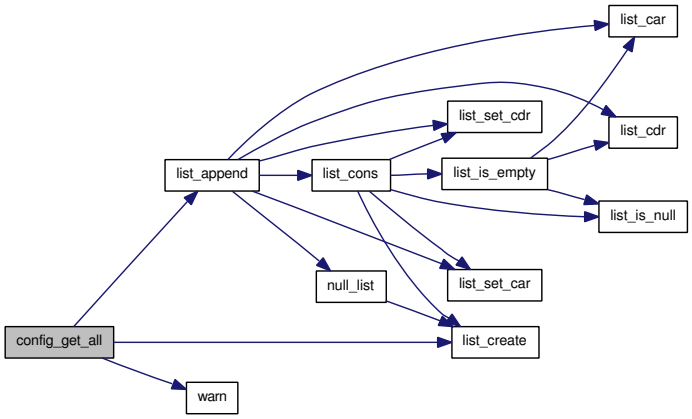
***key*** The key to which to match configuration definitions.

**Returns**

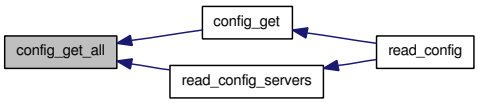
A newly allocated list of newly allocated values, or NULL on failure.

Definition at line 78 of file config.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.4.2.3 void debug (const char * *debug*)

Prints the given debug message to STDOUT if defined.

**Parameters**

***debug*** The message to print.

Definition at line 37 of file config.c.

#### 5.4.2.4 void warn (const char * *warning*)

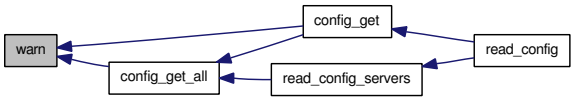
Prints the given warning message to STDOUT if defined.

**Parameters**

***warning*** The message to print.

Definition at line 25 of file config.c.

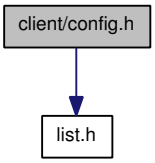
Here is the caller graph for this function:



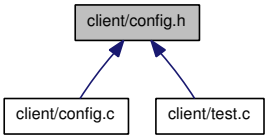
## 5.5 client/config.h File Reference

```
#include "list.h"
```

Include dependency graph for config.h:




This graph shows which files directly or indirectly include this file:




### Functions

- char * config_get (const char *filename, const char *key)

  *Get a value from a configuration file identified by a filename.*

- list * config_get_all (const char *filename, const char *key)

  *Gets all relevant values from a configuration file identifed by a filename.*

### 5.5.1 Detailed Description

A config file looks something like this:

a LINE is less than 1023 characters

```
STRING        --> any stream of characters, including nil
COMMENT       --> # STRING
KEY           --> WORD
VALUE         --> STRING
DEFINITION    --> KEY=VALUE
LINE          --> ( COMMENT | DEFINITON | nil )\n
CONFIGURATION --> LINE*


ACTION --> ( "/dev/kiosk?idTag=" | "/dev/gumstix?idTag=" )
SERVER --> URL:PORT
MODE   --> ( "client" | "kiosk" )
```

There should only be one ACTION. There should only be one MODE. There may be multiple SERVERs specified.

Definition in file config.h.

## 5.5.2 Function Documentation

### 5.5.2.1 char* config_get (const char * *filename*, const char * *key*)

Get a value from a configuration file identified by a filename.

Given a file, and key, return the first matching config value.

**Parameters**

***filename*** The config file.

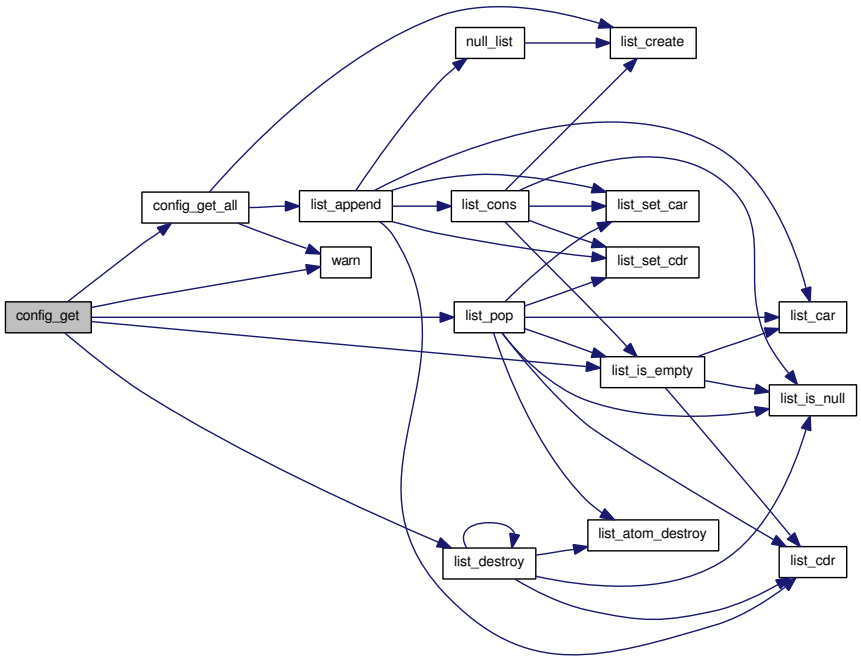
***key*** The key of which the value to find.

**Returns**

A newly allocated string, or NULL on failure.

Definition at line 50 of file config.c.

Here is the call graph for this function:




Here is the caller graph for this function:


### 5.5.2.2 list* config_get_all (const char * *filename*, const char * *key*)

Gets all relevant values from a configuration file identifed by a filename.

Given a filename and key, return all matching config values.

### Parameters

***filename*** The filename of the configuration file.

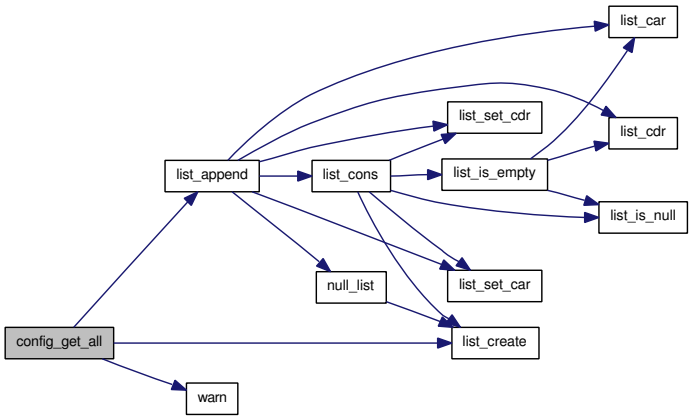
***key*** The key to which to match configuration definitions.

### Returns

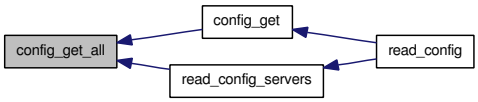
A newly allocated list of newly allocated values, or NULL on failure.

Definition at line 78 of file config.c.

Here is the call graph for this function:



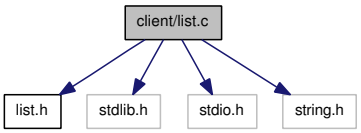

Here is the caller graph for this function:



## 5.6 client/list.c File Reference

```
#include "list.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for list.c:




### Functions

- list * list_create ()
- void * list_car (list *l)
- void * list_cdr (list *l)
- void list_set_car (list *l, void *v)
- void list_set_cdr (list *l, void *v)
- int list_is_null (list *l)
- int list_is_empty (list *l)
- int list_size (list *l)
- list * list_cons (void *v, list *l)
- list * list_reverse (list *l)
- void list_atom_destroy (list *l)
- void list_destroy (list *l)
- void list_destroy_deep (list *l)
- list * list_map (list *l, void *(*fn)(void *))
- list * list_filter (list *l, int(*fn)(void *, void *), void *args)
- void * list_simple_free_fn (void *v)
- void * **list_str_print_fn** (void *v)
- void * **list_int_print_fn** (void *v)
- list * null_list ()
- void * list_nth (list *l, int i)
- void ** list_array (list *l)
- int list_index_of_int (list *haystack, int needle)
- int list_index_of_str (list *haystack, const char *needle)
- list * list_append (list *l, void *v)
- list * list_push (list *l, void *v)
- void * list_pop (list *l)
- list * list_nth_atom (list *l, int i)
- void * list_remove (list *l, int index)

### Variables

- int **TRUE** = 1

### 5.6.1 Detailed Description

**Author**

Willi Ballenthin

This file contains the definition of a singly-linked list data structure. It was developed as I learned and explored LISP. The influence of LISP is clear in the API.

Definition in file list.c.

### 5.6.2 Function Documentation

#### 5.6.2.1 list* list_append (list * *l*, void * *v*)

Destructive.

**Parameters**

***l*** A list.

***v*** A value to insert at the last position.

**Postcondition**

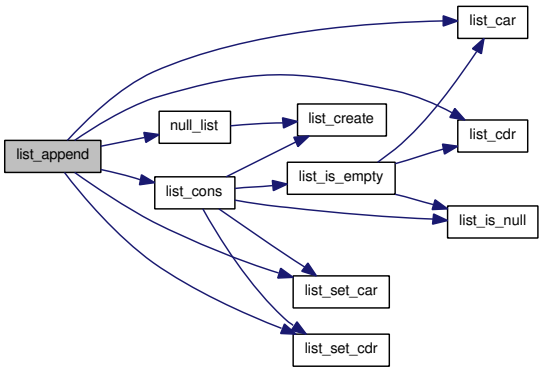
The last element of the list now points to a new node containing v.

**Returns**

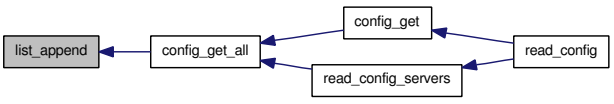
The newly created end node, or NULL on failure.

Definition at line 497 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



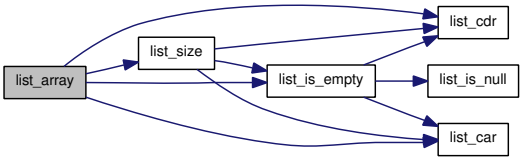
#### 5.6.2.2 void** list_array (list * *l*)

**Returns**

a new array, filled with the contents of the provided list the list is NULL terminated, but since the values of the array may be NULL, this should not be used to determine the length of the array. rather, list_size() should be used. NULL on failure or empty list.

Definition at line 410 of file list.c.

Here is the call graph for this function:

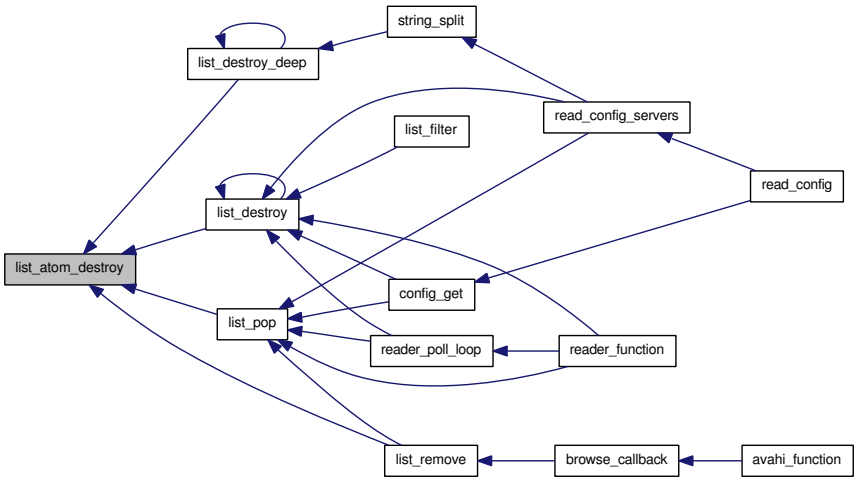



#### 5.6.2.3 void list_atom_destroy (list * *l*)

inverse of list_create() does not free contents

Definition at line 228 of file list.c.

Here is the caller graph for this function:




#### 5.6.2.4 void* list_car (list * *l*)

**Parameters**

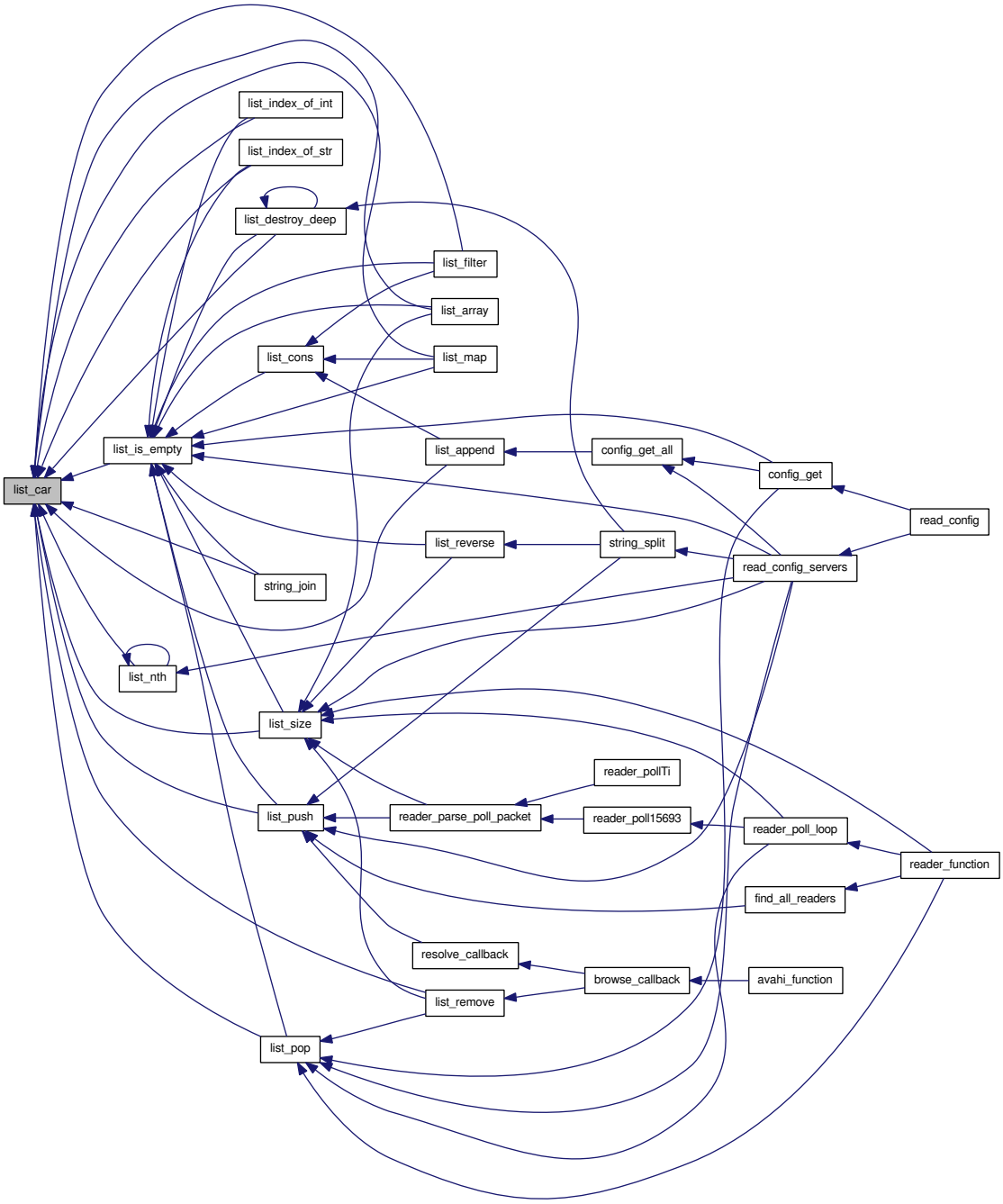
***l*** a list.

**Returns**

The car of the list.

Definition at line 41 of file list.c.

Here is the caller graph for this function:




### 5.6.2.5 void* list_cdr (list * *l*)

## Parameters

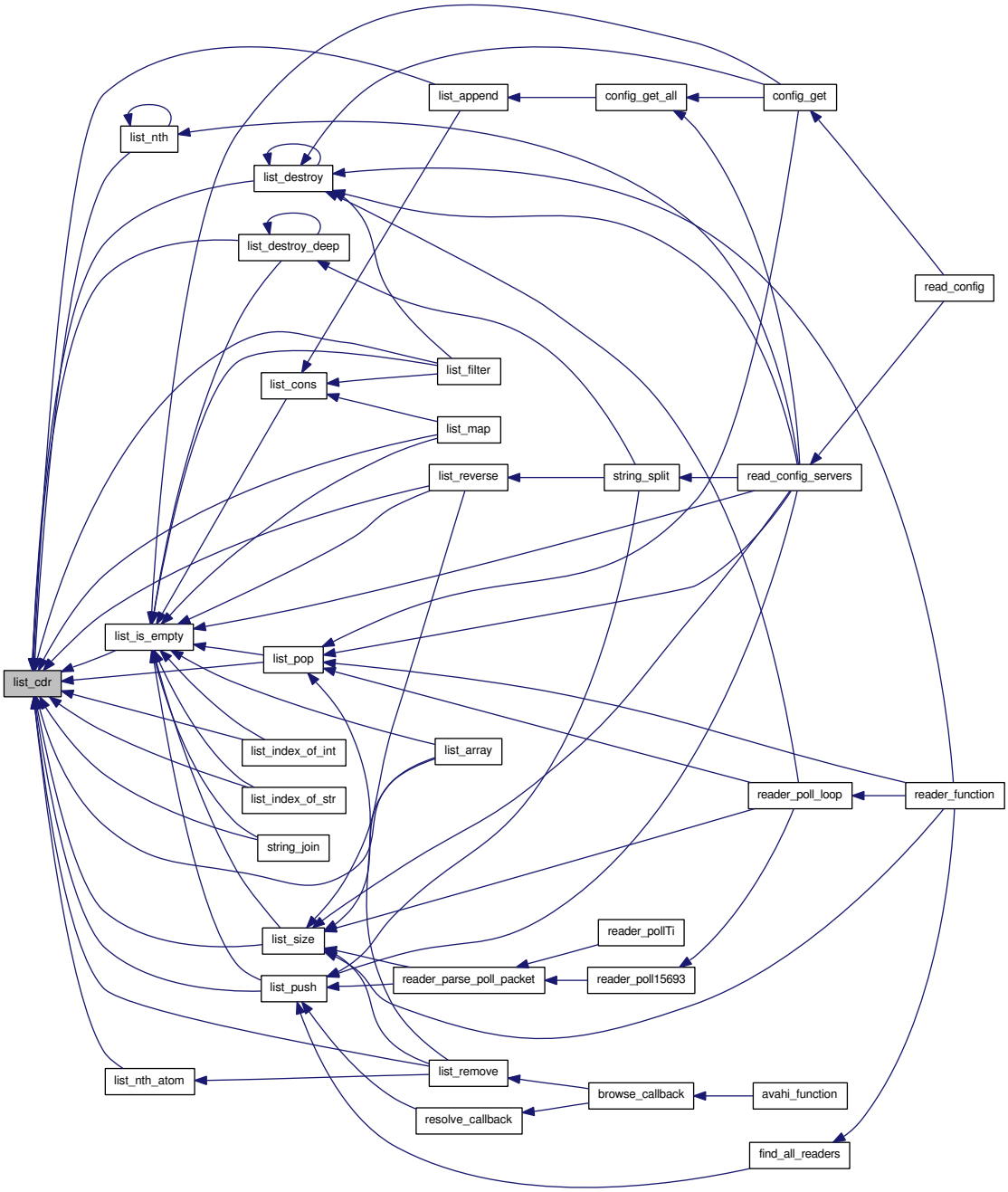
***a*** list

### Returns

The cdr of the list.

Definition at line 50 of file list.c.

Here is the caller graph for this function:



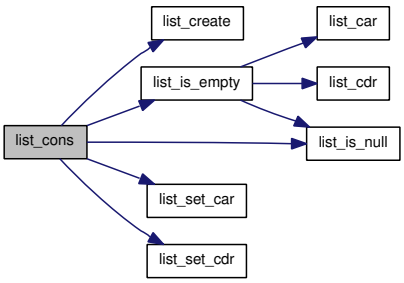
#### 5.6.2.6 list* list_cons (void * v, list * l)

**Returns**

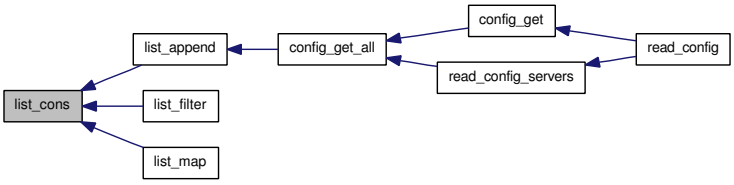
a list formed of the given value cons-ed with the given list. Allocates memory. Returns NULL on failure.

Definition at line 131 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



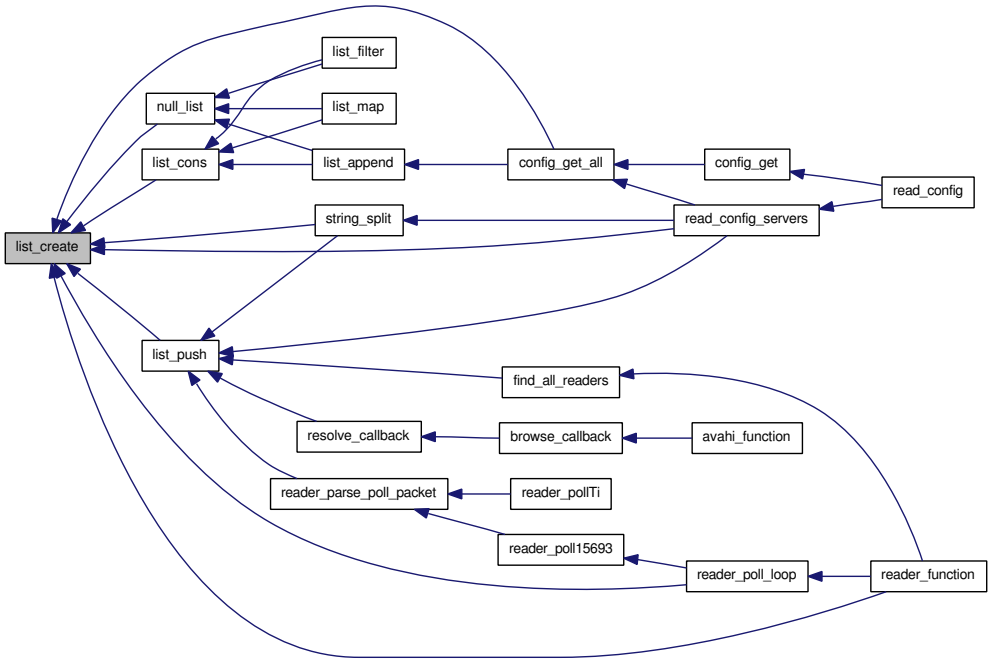

#### 5.6.2.7 list* list_create ()

**Returns**

A newly allocated list, or NULL on failure

Definition at line 23 of file list.c.

Here is the caller graph for this function:

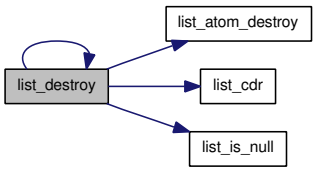



#### 5.6.2.8 void list_destroy (list * *l*)

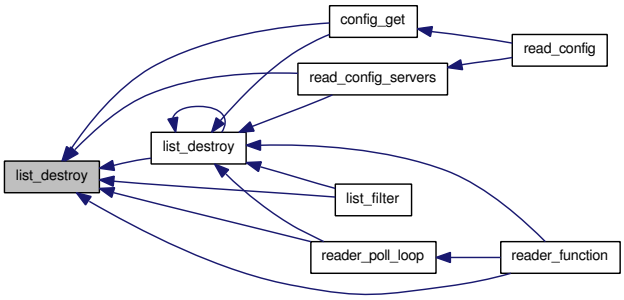
list_atom_destroy() chained together across an entire list. currently recursive, so probs not a good function for large lists... although i heard GCC can optimize tail recursion, so this may be fine

Definition at line 242 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:

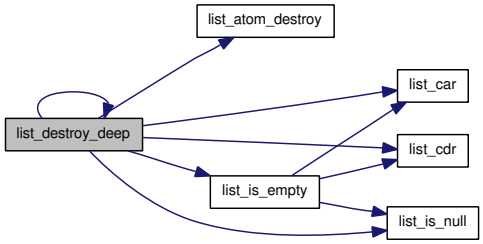



#### 5.6.2.9 void list_destroy_deep (list * *l*)

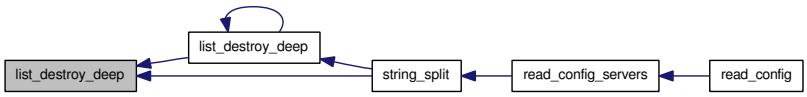
like list_destroy() but calls free on each car.

Definition at line 263 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.6.2.10 list* list_filter (list * *l*, int(*)(void *, void *) *fn*, void * *args*)

Given a filtering function,

**Parameters**

***l*** The list to filter.

***fn*** A function to apply to each element in the list. If the application is TRUE, then the element is retained, else it is not included in the new list.

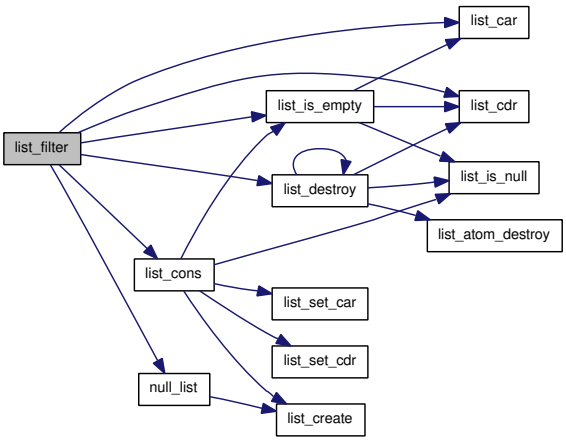
***args*** Arguments to be passed to the filtering function.

**Returns**

a new list consisting of elements of the old list that pass the test or NULL on failure.

Definition at line 326 of file list.c.

Here is the call graph for this function:



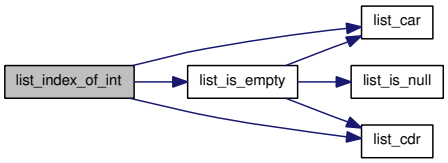

#### 5.6.2.11 int list_index_of_int (list * *haystack*, int *needle*)

**Returns**

index of first eleemnt encountered matching needle, ENOTFND otherwise.

Definition at line 446 of file list.c.

Here is the call graph for this function:



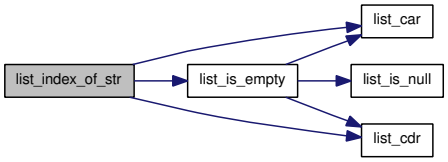

#### 5.6.2.12 int list_index_of_str (list * *haystack*, const char * *needle*)

**Returns**

index of first eleemnt encountered matching needle, ENOTFND otherwise.

Definition at line 467 of file list.c.

Here is the call graph for this function:



#### 5.6.2.13 int list_is_empty (list * *l*)

**Parameters**

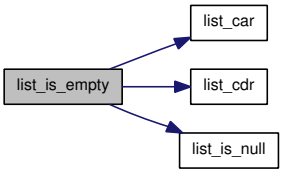
***l*** a list.

**Returns**

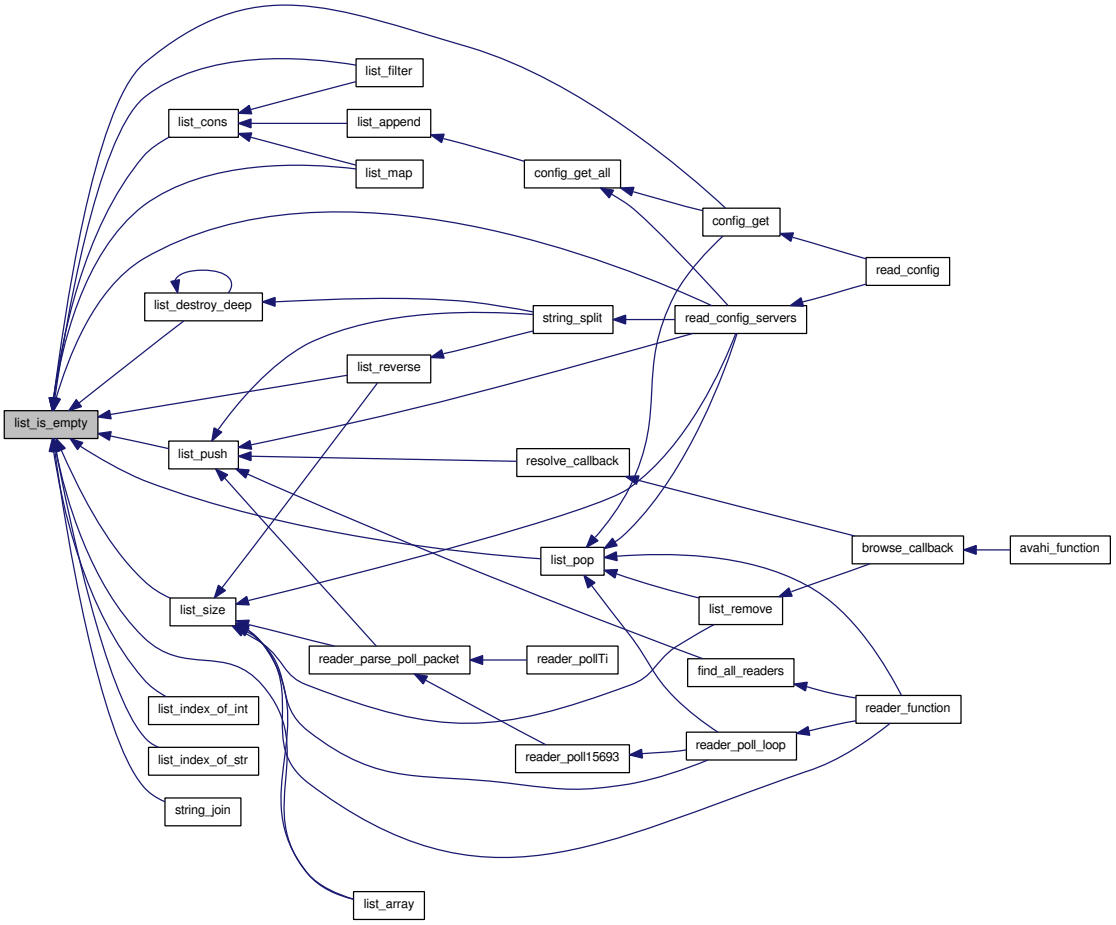
true if the parameter is empty

Definition at line 96 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 5.6.2.14 int list_is_null (list * *l*)

**Parameters**

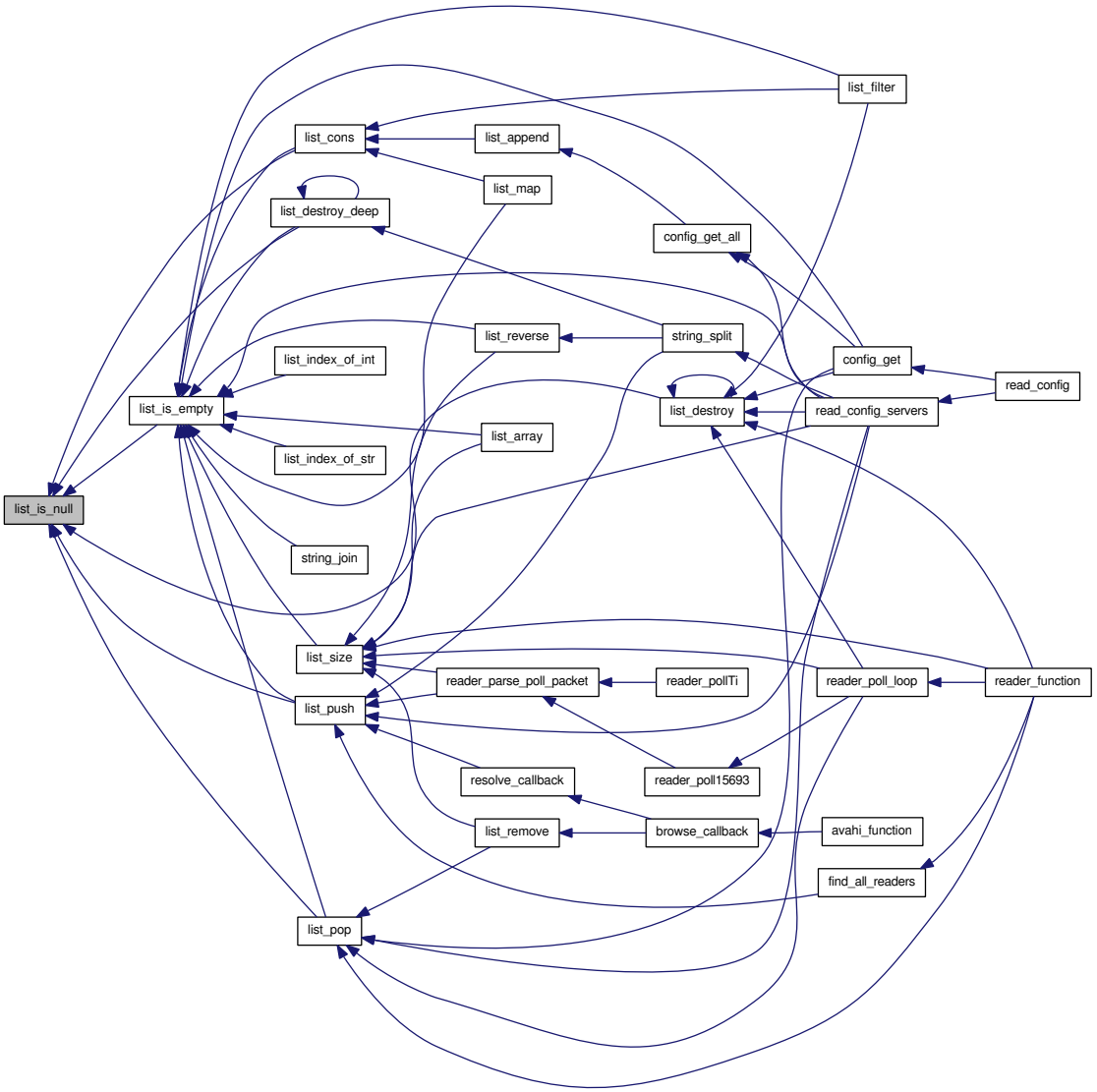
*l* a list.

**Returns**

true if the parameter is null.

Definition at line 88 of file list.c.

Here is the caller graph for this function:




#### 5.6.2.15 list* list_map (list * *l*, void *(*)(void *) *fn*)

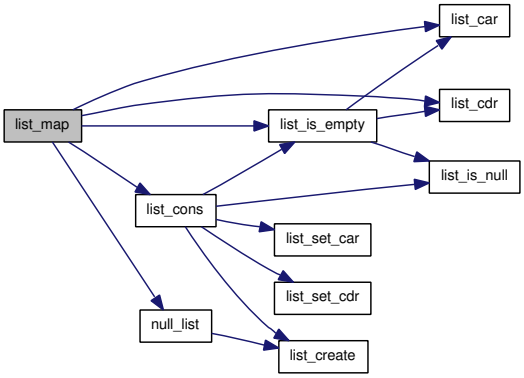
Maps a function across all elements of a list

**Returns**

a new list of the results, or NULL on failure

Definition at line 290 of file list.c.

Here is the call graph for this function:



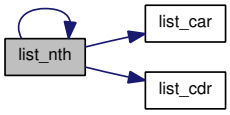

#### 5.6.2.16 void* list_nth (list * *l*, int *i*)

**Returns**

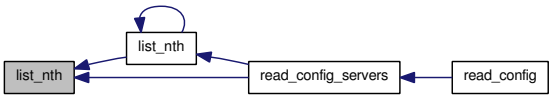
the value at the nth position of the list. NULL on failure, however accessing past the end of the list is undefined.

Definition at line 394 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.6.2.17 list* list_nth_atom (list * *l*, int *i*)

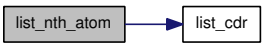
Private

**Returns**

nth atom in list.

Definition at line 599 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 5.6.2.18 void* list_pop (list * *l*)

Destructive. Convenience function. Opposite of list_push().

**Parameters**

***l*** A list.

**Returns**

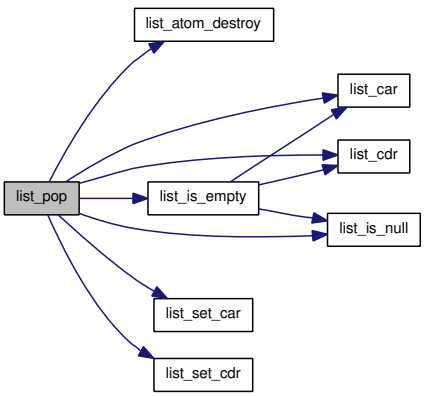
The car of the old list.

**Postcondition**

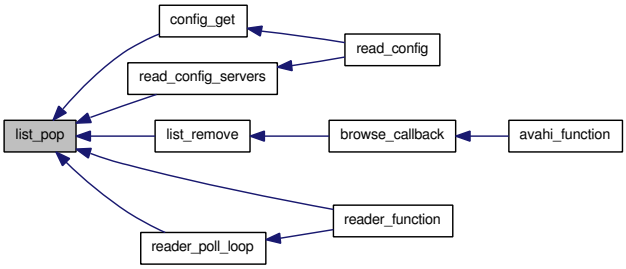
The list is now the cdr of the original list.

Definition at line 569 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 5.6.2.19 list* list_push (list * *l*, void * *v*)

Alias of: l = list_cons(v, l);

**Parameters**

***l*** A list to become the cdr.

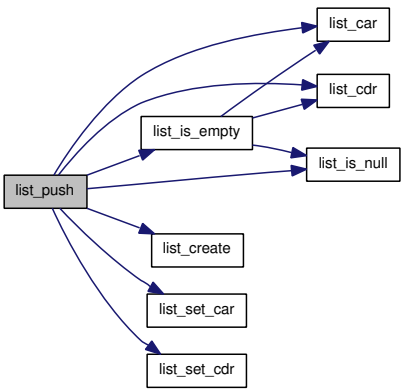
***v*** A value to become the car.

**Returns**

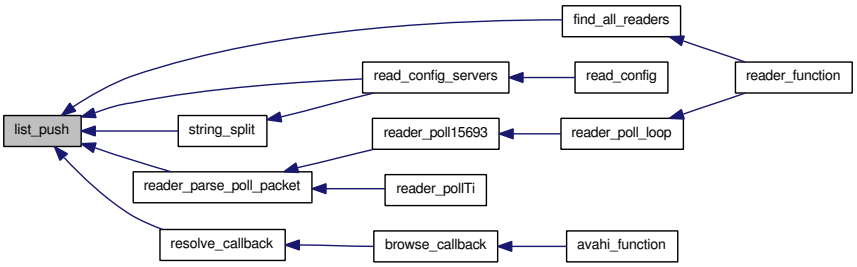
A list composed of v and l or NULL on failure

Definition at line 535 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 5.6.2.20 void* list_remove (list * *l*, int *index*)

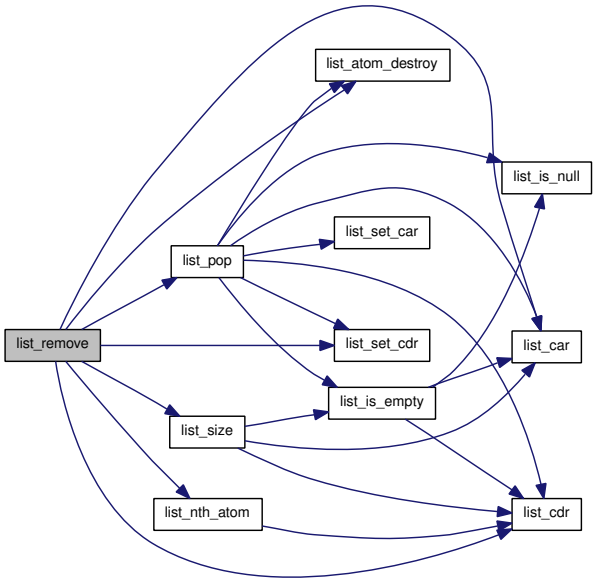
Removes the element at the given index

**Returns**

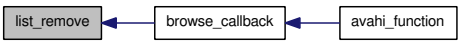
the value of the element removed, or NULL on failure.

Definition at line 611 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 5.6.2.21 list* list_reverse (list * *l*)

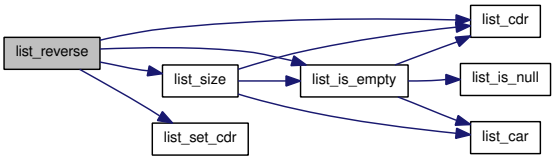
destructively modifies the current list. note, you must assign the result back in place, for this method is unable to modify in place, at the current time.

**Returns**

the list reversed. pointer to new head.

Definition at line 163 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 5.6.2.22 void list_set_car (list * *l*, void * *v*)

Sets the car of a list.

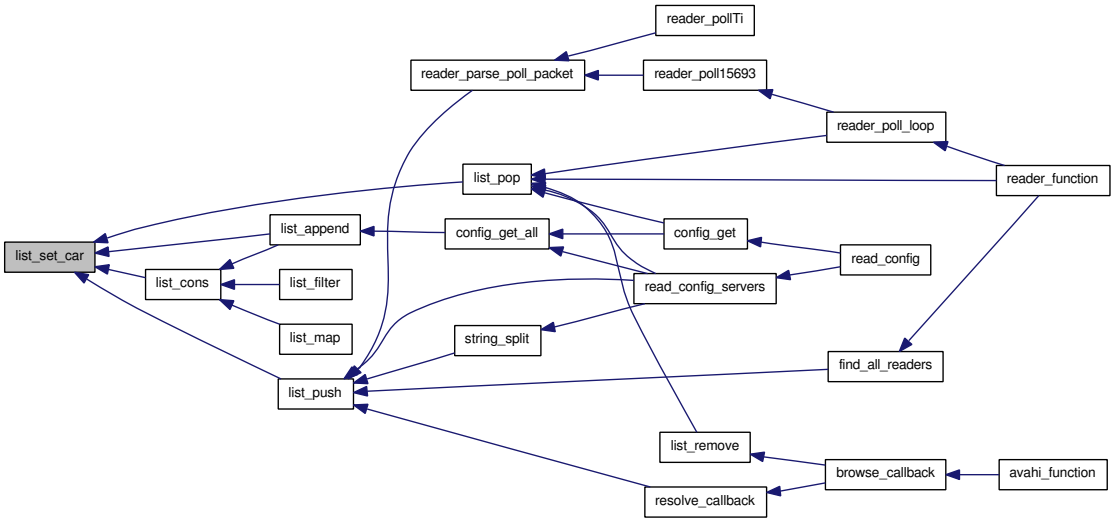
**Parameters**

***l*** list.

***v*** the value.

Definition at line 61 of file list.c.

Here is the caller graph for this function:




### 5.6.2.23 void list_set_cdr (list * *l*, void * *v*)

Sets the cdr of a list.

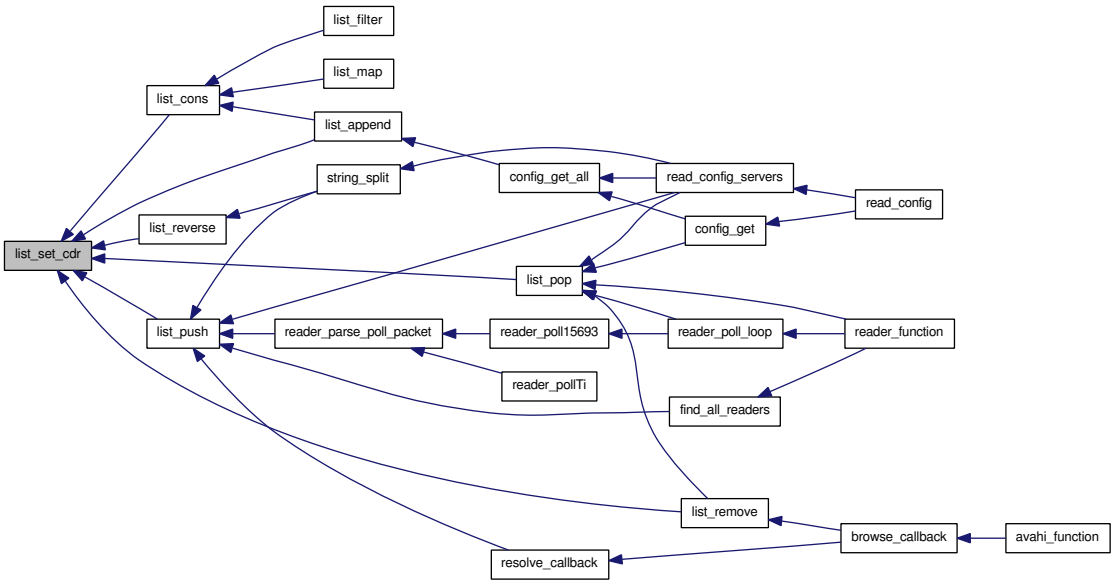
**Parameters**

*l* list.

*v* the value.

Definition at line 75 of file list.c.

Here is the caller graph for this function:



#### 5.6.2.24 void* list_simple_free_fn (void * *v*)

the following 3 functions conform to the function signatures accepted by list_map and list_filter

Definition at line 365 of file list.c.

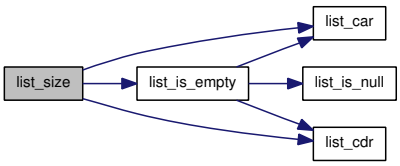
#### 5.6.2.25 int list_size (list * *l*)

**Returns**

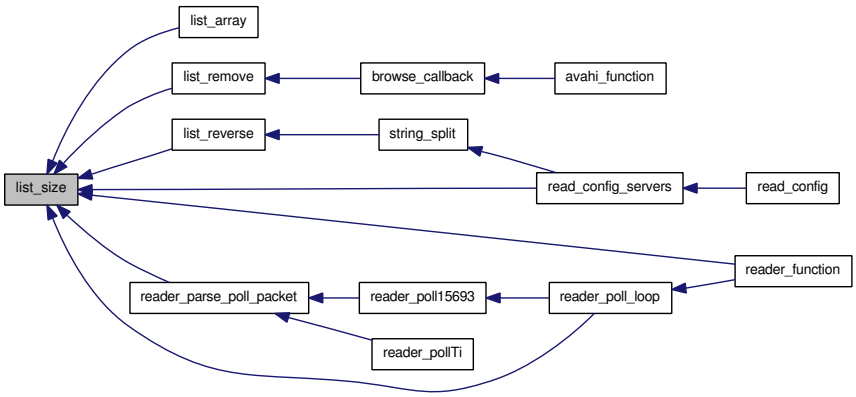
size of list.

Definition at line 104 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



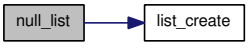

#### 5.6.2.26 list* null_list ()

**Returns**

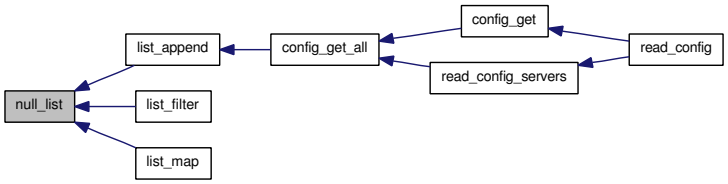
a new empty list, or NULL on failure.

Definition at line 385 of file list.c.

Here is the call graph for this function:

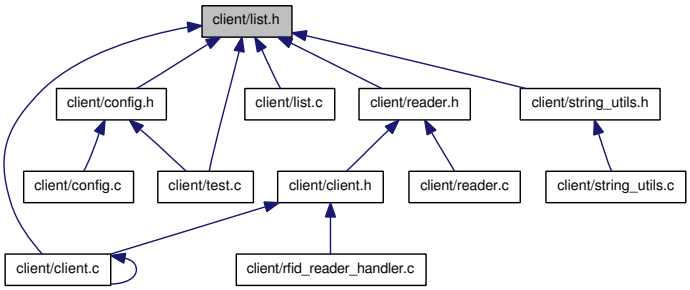


Here is the caller graph for this function:



## 5.7 client/list.h File Reference

This graph shows which files directly or indirectly include this file:




### Data Structures

- struct list

### Defines

- #define list_foreach(l, t) for (t = l; ! list_is_empty(t); t = list_cdr(t))
- #define list_foreach_entry(l, t, type, e)
- #define **ENOTFND** -1

### Typedefs

- typedef struct list list
- typedef list **list_t**

### Functions

- list * list_create ()
- void list_atom_destroy (list *l)
- void list_destroy (list *l)
- void list_destroy_deep (list *l)
- void * list_car (list *l)
- void * list_cdr (list *l)
- void list_set_car (list *l, void *v)
- void list_set_cdr (list *l, void *v)
- int list_is_empty (list *l)
- int list_size (list *l)
- list * list_cons (void *v, list *l)
- list * list_map (list *l, void *(*fn)(void *))
- list * list_filter (list *l, int(*fn)(void *, void *), void *args)
- list * **list_copy** (list *l)
- list * list_reverse (list *l)
- list * list_append (list *l, void *v)
- void * list_remove (list *l, int index)

- void * list_simple_free_fn (void *v)
- void * **list_str_print_fn** (void *)
- void * **list_int_print_fn** (void *)
- list * null_list ()
- list * list_push (list *l, void *v)
- void * list_pop (list *l)
- void * list_nth (list *l, int i)
- void ** list_array (list *l)
- int list_index_of_int (list *haystack, int needle)
- int list_index_of_str (list *haystack, const char *needle)

### 5.7.1 Detailed Description

**Author**

Willi Ballenthin

This file contains the declaration of a singly-linked list data structure.

Definition in file list.h.

### 5.7.2 Define Documentation

#### 5.7.2.1 #define list_foreach(l, t) for (t = l; ! list_is_empty(t); t = list_cdr(t))

**Parameters**

***l*** list *, the list to enumerate

***t*** list *, a temporary list

Definition at line 82 of file list.h.

#### 5.7.2.2 #define list_foreach_entry(l, t, type, e)

**Value:**

```
for (t = l, e = (type)list_car(t); \
      ! list_is_empty(t); \
      t = list_cdr(t), e = (type)list_car(t))
```

**Parameters**

***l*** list *l - the list to enumerate

***t*** list *t - a temporary list

***type*** the type of the entry

***e*** - entry of type TYPE

Definition at line 91 of file list.h.

## 5.7.3 Typedef Documentation

### 5.7.3.1 typedef struct list list

standardization:

a list ends with a cell that looks like:

```
     +---+---+
---> | v | 0 |
     +---+---+
```

and empty list then looks like:

```
+---+---+
| 0 | 0 |
+---+---+
```

it should look like this:

```
0
```

but this doesnt work, because we would need to be able to modify the pointers but of course they are passed by value...

so this implementation needs to be careful on edge cases, checking when there are fewer than 2 elements and whatnot

## 5.7.4 Function Documentation

### 5.7.4.1 list* list_append (list * *l*, void * *v*)

Destructive.

**Parameters**

***l*** A list.

***v*** A value to insert at the last position.

**Postcondition**

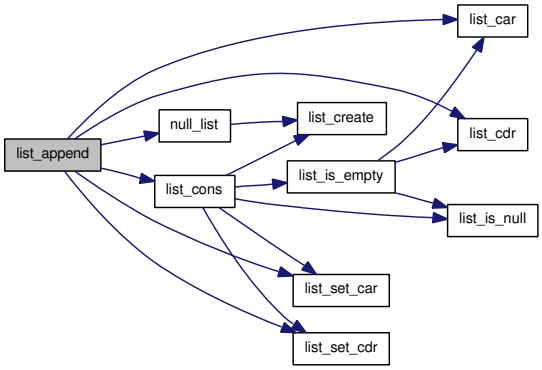
The last element of the list now points to a new node containing v.

**Returns**

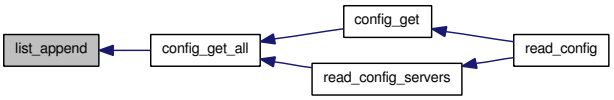
The newly created end node, or NULL on failure.

Definition at line 497 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



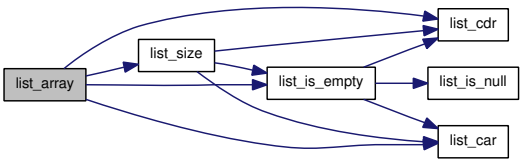

#### 5.7.4.2 void** list_array (list * *l*)

**Returns**

a new array, filled with the contents of the provided list the list is NULL terminated, but since the values of the array may be NULL, this should not be used to determine the length of the array. rather, list_size() should be used. NULL on failure or empty list.

Definition at line 410 of file list.c.

Here is the call graph for this function:

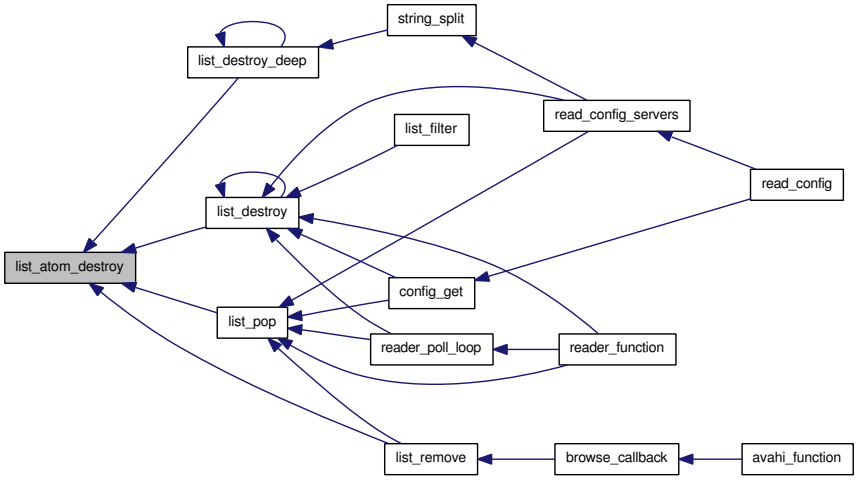



#### 5.7.4.3 void list_atom_destroy (list * *l*)

inverse of list_create() does not free contents

Definition at line 228 of file list.c.

Here is the caller graph for this function:




#### 5.7.4.4 void* list_car (list * *l*)

**Parameters**

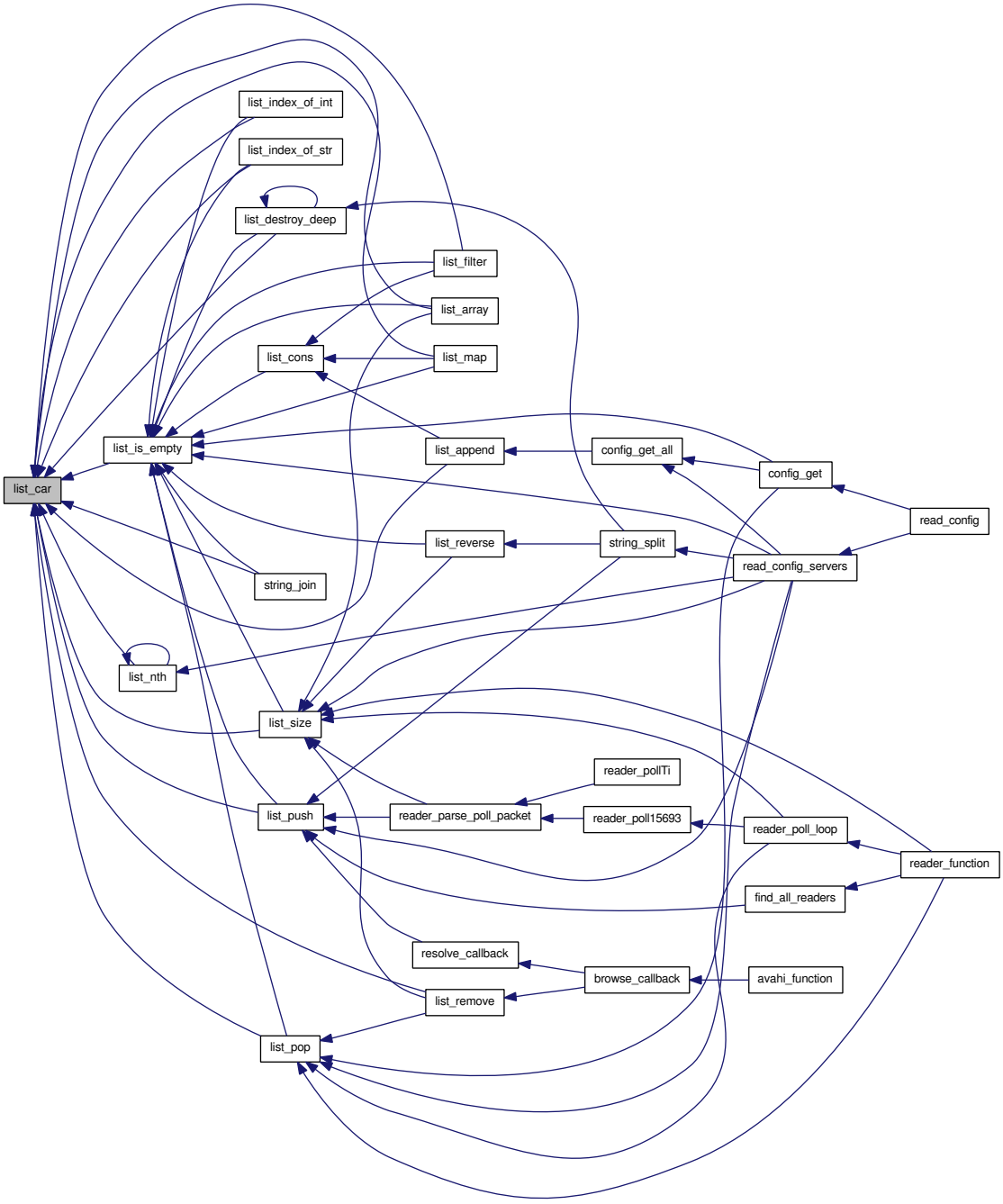
***l*** a list.

**Returns**

The car of the list.

Definition at line 41 of file list.c.

Here is the caller graph for this function:




### 5.7.4.5 void* list_cdr (list * *l*)

## Parameters

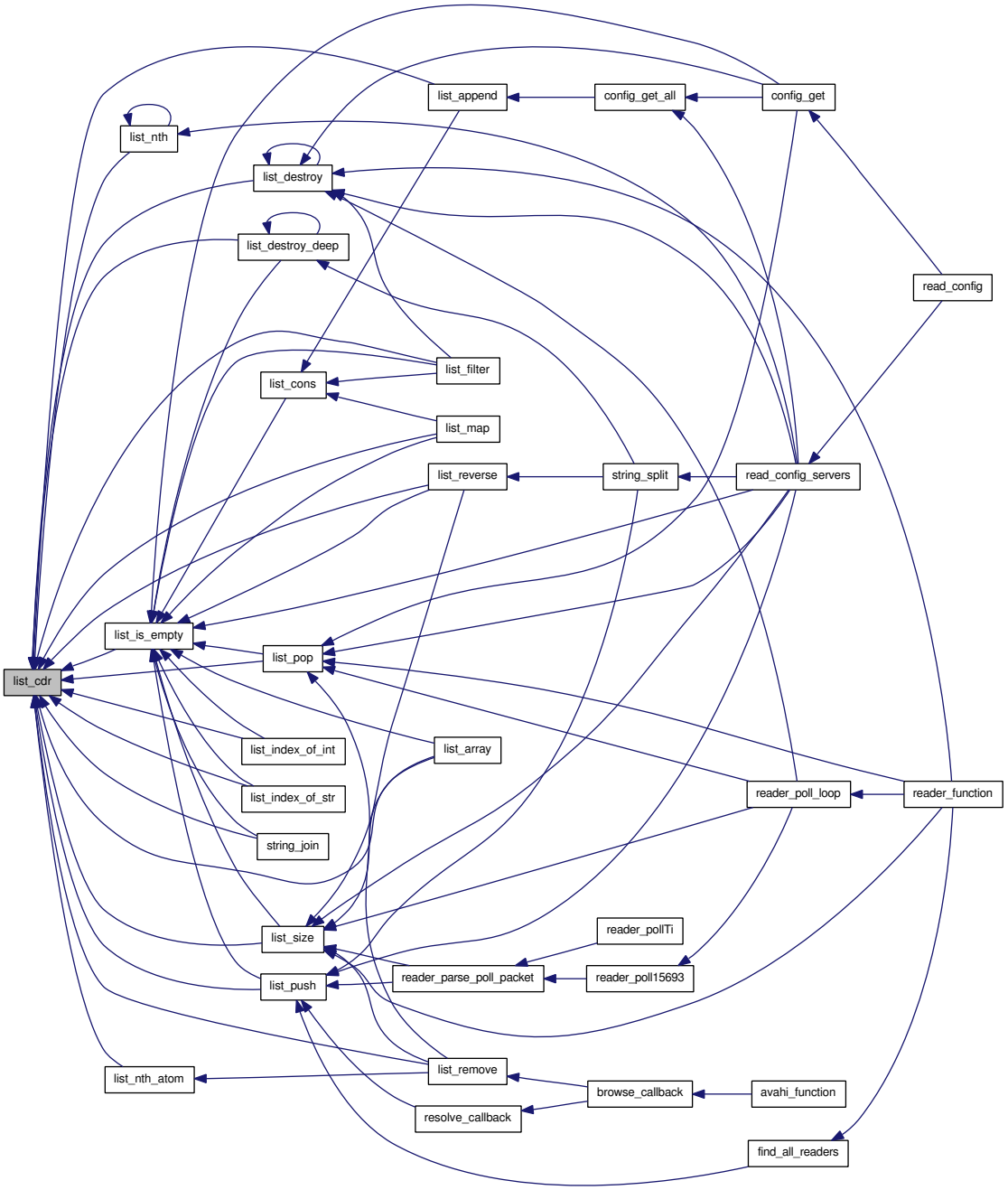
***a*** list

### Returns

The cdr of the list.

Definition at line 50 of file list.c.

Here is the caller graph for this function:



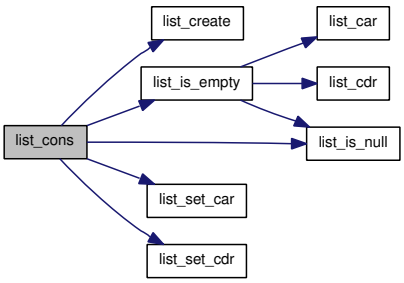
#### 5.7.4.6 list* list_cons (void * v, list * l)

**Returns**

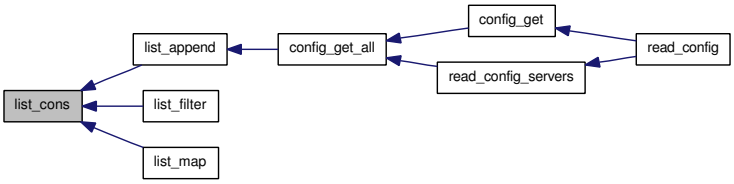
a list formed of the given value cons-ed with the given list. Allocates memory. Returns NULL on failure.

Definition at line 131 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



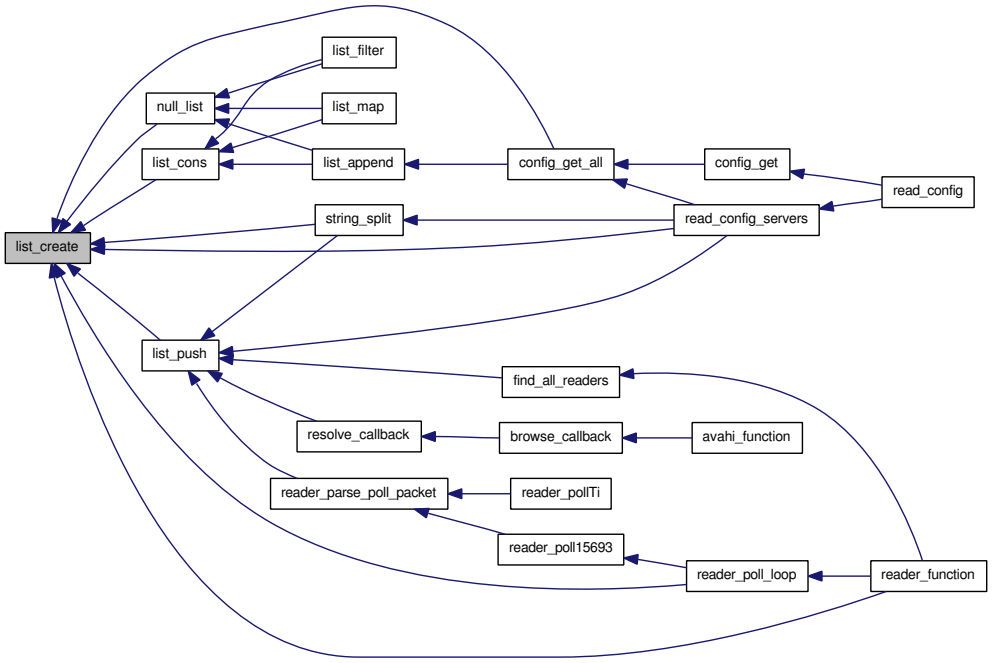

#### 5.7.4.7 list* list_create ()

**Returns**

A newly allocated list, or NULL on failure

Definition at line 23 of file list.c.

Here is the caller graph for this function:

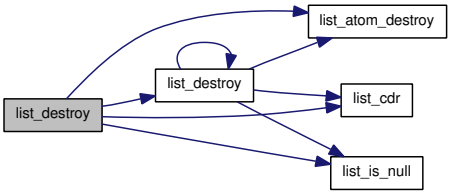



#### 5.7.4.8 void list_destroy (list * *l*)

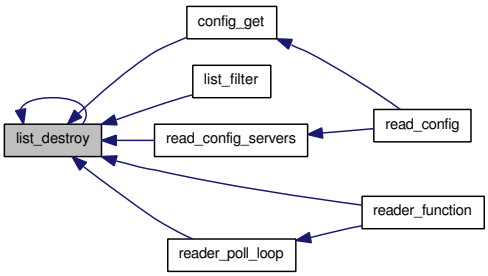
list_atom_destroy() chained together across an entire list. currently recursive, so probs not a good function for large lists... although i heard GCC can optimize tail recursion, so this may be fine

Definition at line 242 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:

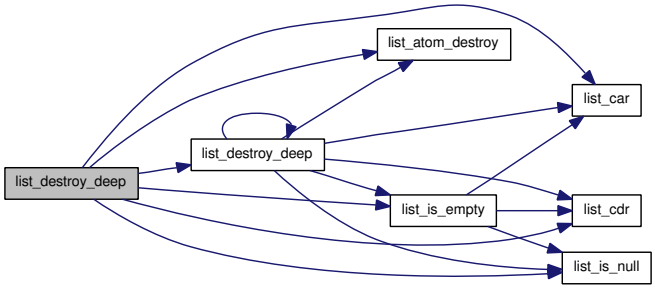



#### 5.7.4.9 void list_destroy_deep (list * *l*)

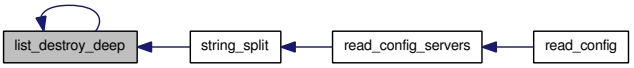
like list_destroy() but calls free on each car.

Definition at line 263 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.4.10 list* list_filter (list * *l*, int(*)(void *, void *) *fn*, void * *args*)

Given a filtering function,

**Parameters**

***l*** The list to filter.

***fn*** A function to apply to each element in the list. If the application is TRUE, then the element is retained, else it is not included in the new list.

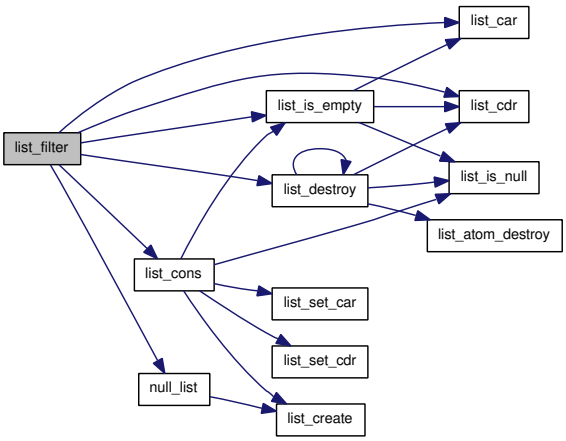
***args*** Arguments to be passed to the filtering function.

**Returns**

a new list consisting of elements of the old list that pass the test or NULL on failure.

Definition at line 326 of file list.c.

Here is the call graph for this function:



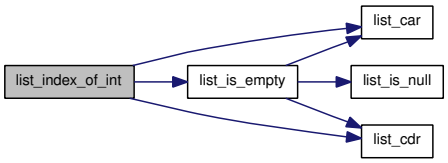

#### 5.7.4.11 int list_index_of_int (list * *haystack*, int *needle*)

**Returns**

index of first eleemnt encountered matching needle, ENOTFND otherwise.

Definition at line 446 of file list.c.

Here is the call graph for this function:



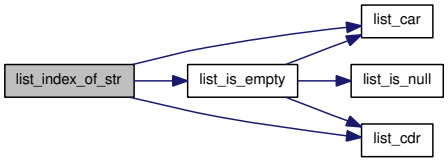

#### 5.7.4.12 int list_index_of_str (list * *haystack*, const char * *needle*)

**Returns**

index of first eleemnt encountered matching needle, ENOTFND otherwise.

Definition at line 467 of file list.c.

Here is the call graph for this function:



#### 5.7.4.13 int list_is_empty (list * *l*)

**Parameters**

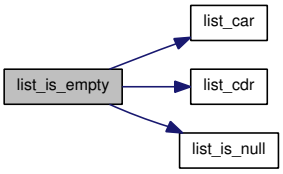
***l*** a list.

**Returns**

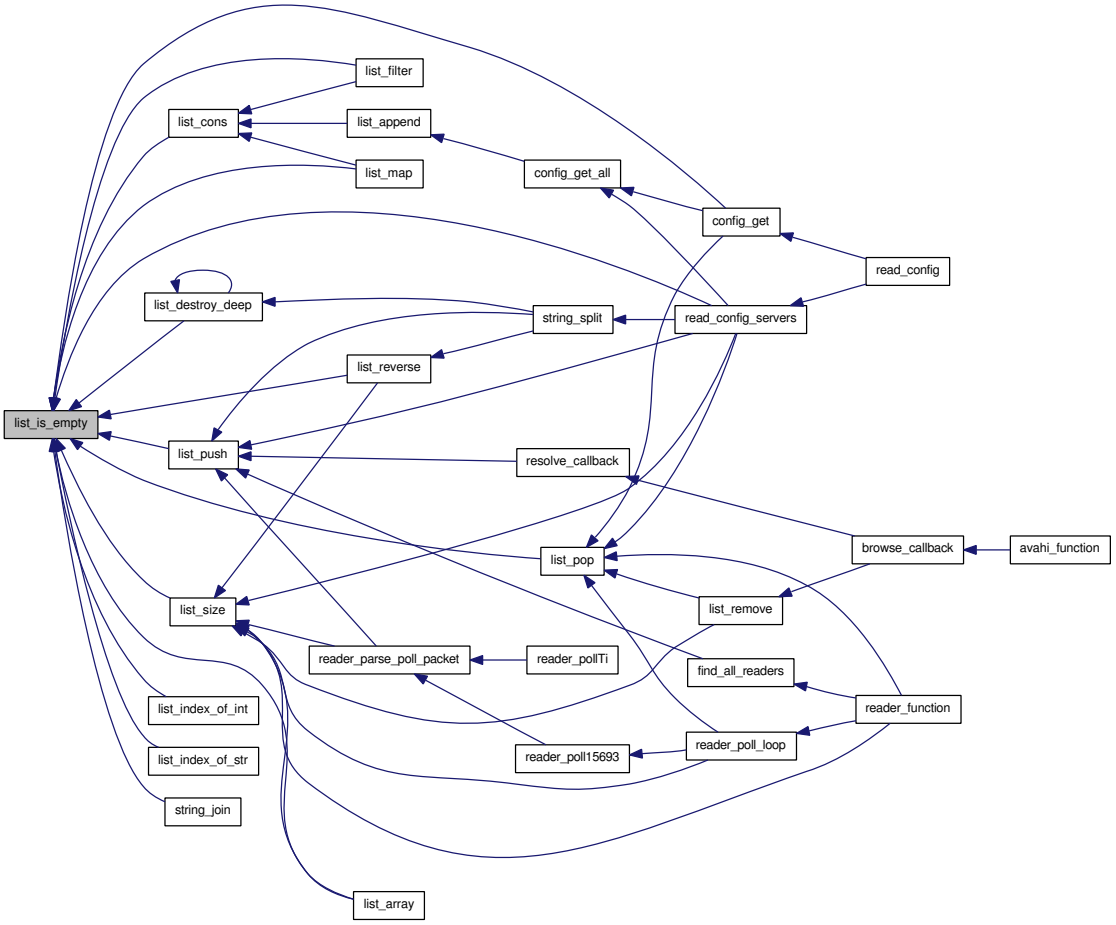
true if the parameter is empty

Definition at line 96 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 5.7.4.14 list* list_map (list * *l*, void *(*)(void *) *fn*)

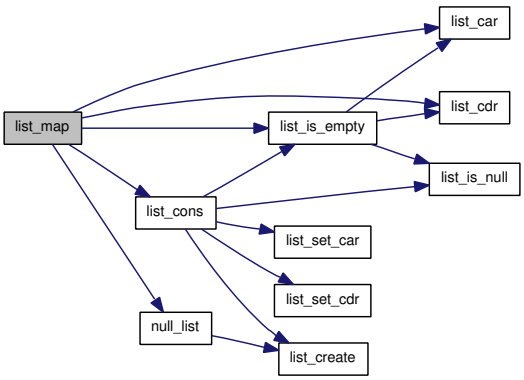
Maps a function across all elements of a list

**Returns**

a new list of the results, or NULL on failure

Definition at line 290 of file list.c.

Here is the call graph for this function:



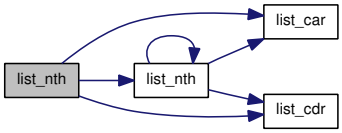

#### 5.7.4.15 void* list_nth (list * *l*, int *i*)

**Returns**

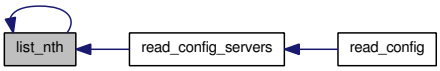
the value at the nth position of the list. NULL on failure, however accessing past the end of the list is undefined.

Definition at line 394 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.4.16 void* list_pop (list * *l*)

Destructive. Convenience function. Opposite of list_push().

### Parameters

***l*** A list.

### Returns

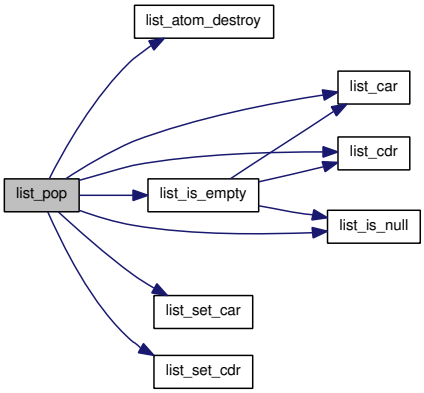
The car of the old list.

### Postcondition

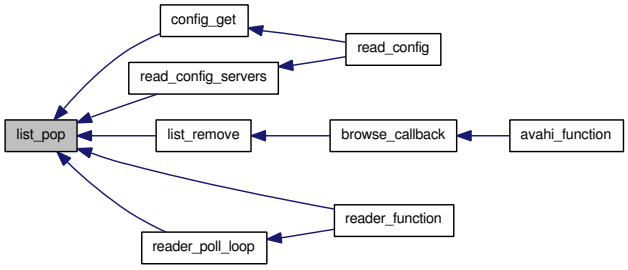
The list is now the cdr of the original list.

Definition at line 569 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.4.17 list* list_push (list * *l*, void * *v*)

Alias of: l = list_cons(v, l);

### Parameters

***l*** A list to become the cdr.

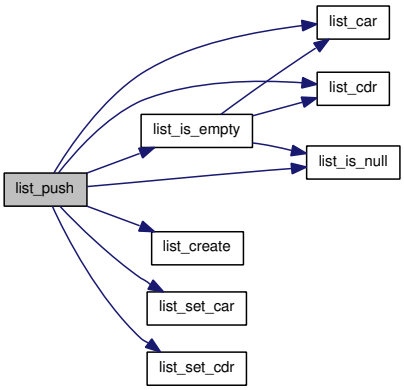
***v*** A value to become the car.

### Returns

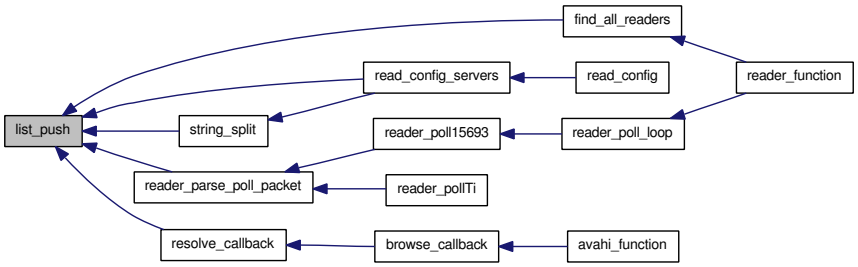
A list composed of v and l or NULL on failure

Definition at line 535 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.4.18 void* list_remove (list * *l*, int *index*)

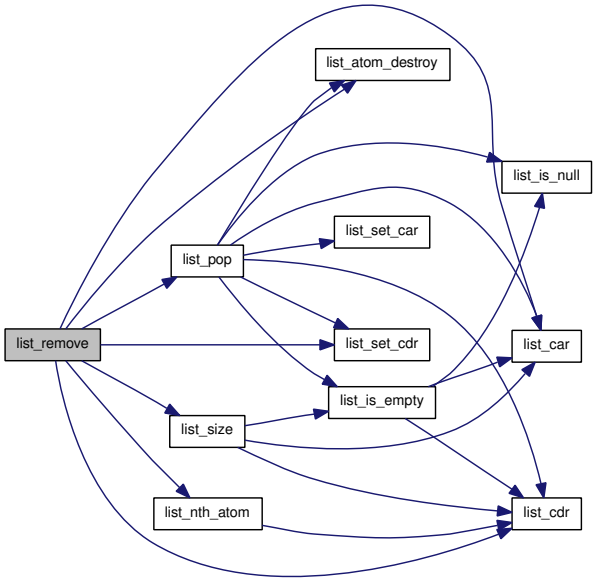
Removes the element at the given index

**Returns**

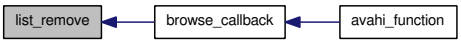
the value of the element removed, or NULL on failure.

Definition at line 611 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.4.19 list* list_reverse (list * *l*)

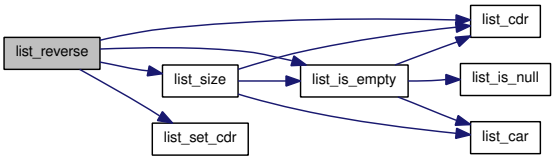
destructively modifies the current list. note, you must assign the result back in place, for this method is unable to modify in place, at the current time.

**Returns**

the list reversed. pointer to new head.

Definition at line 163 of file list.c.

Here is the call graph for this function:




Here is the caller graph for this function:

#### 5.7.4.20 void list_set_car (list * *l*, void * *v*)

Sets the car of a list.

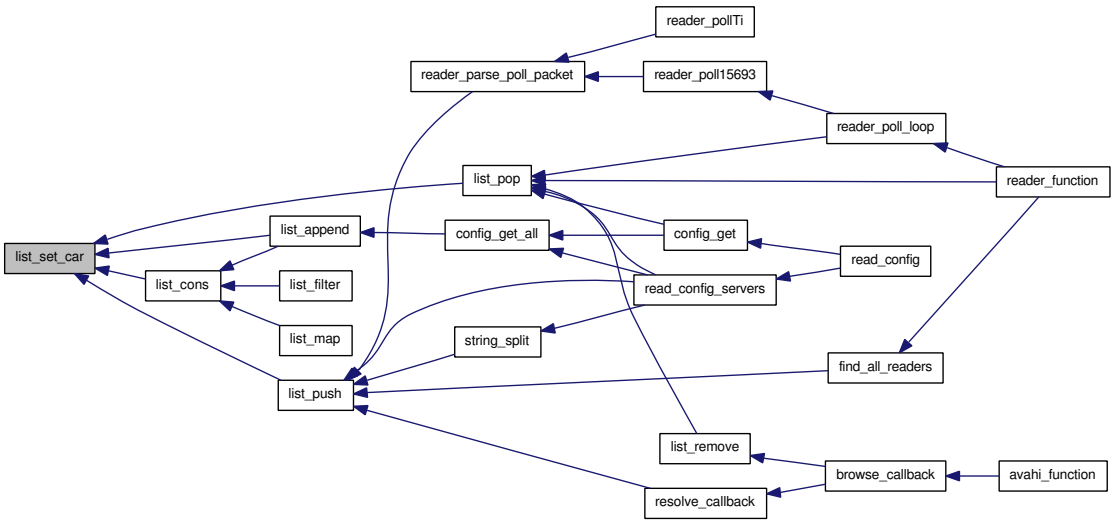
**Parameters**

***l*** list.

***v*** the value.

Definition at line 61 of file list.c.

Here is the caller graph for this function:




#### 5.7.4.21 void list_set_cdr (list * *l*, void * *v*)

Sets the cdr of a list.

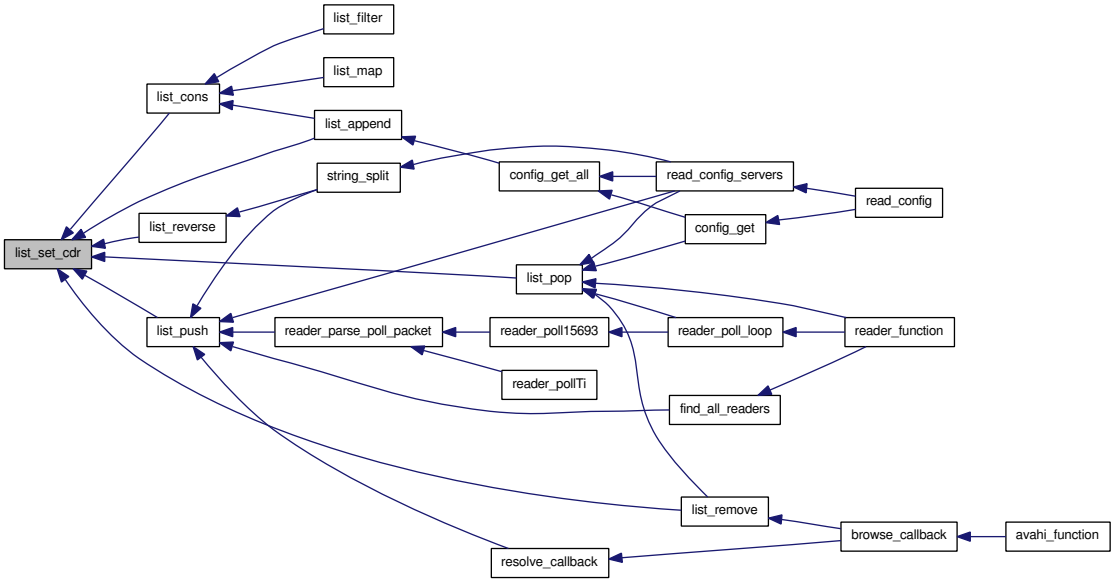
**Parameters**

***l*** list.

***v*** the value.

Definition at line 75 of file list.c.

Here is the caller graph for this function:




#### 5.7.4.22 void* list_simple_free_fn (void * *v*)

the following 3 functions conform to the function signatures accepted by list_map and list_filter

Definition at line 365 of file list.c.

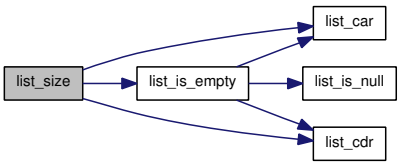
#### 5.7.4.23 int list_size (list * *l*)

**Returns**

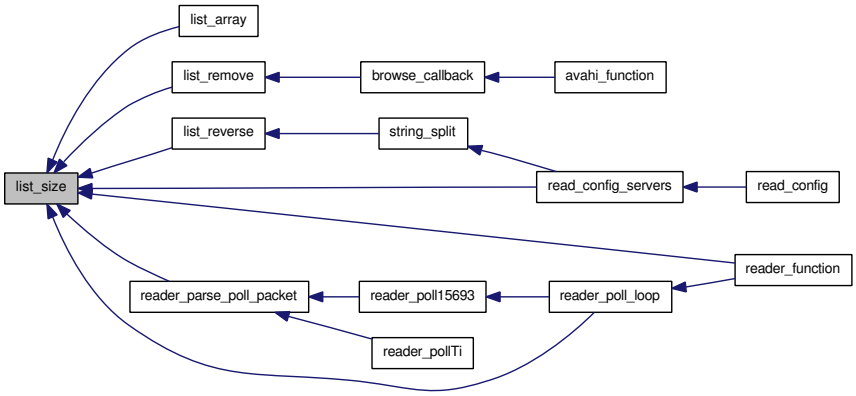
size of list.

Definition at line 104 of file list.c.

Here is the call graph for this function:



Here is the caller graph for this function:



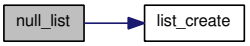

#### 5.7.4.24 list* null_list ()

**Returns**

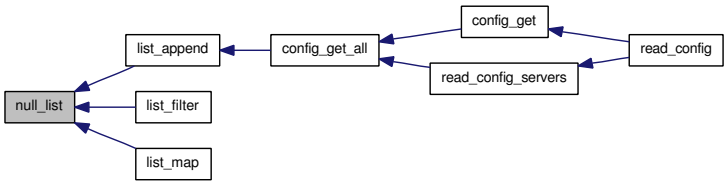
a new empty list, or NULL on failure.

Definition at line 385 of file list.c.

Here is the call graph for this function:



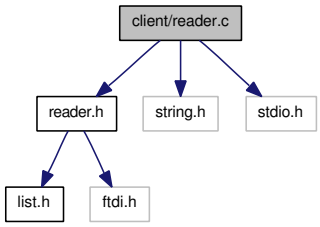

Here is the caller graph for this function:



## 5.8 client/reader.c File Reference

```
#include "reader.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for reader.c:




### Functions

- struct reader * reader_create ()
  *Return a newly allocated reader.*
- void reader_init (struct reader *reader, struct ftdi_context *ftdic, struct usb_device *dev)
  *Initialize a newly allocated reader.*
- struct tag * tag_create ()
  *Returns a newly allocated tag.*
- void tag_init (struct tag *tag, char *id, struct reader *parent)
  *Initializes a newly allocated tag.*
- list * find_all_readers (struct list *readers)
  *Returns a list of all connected readers.*
- ReturnCode reader_connect (struct reader *reader, int beep)
  *Connects to a given reader.*
- ReturnCode reader_disconnect (struct reader *reader)
  *Disconnects from a given reader.*
- ReturnCode reader_reset (struct reader *reader)
  *Resets a given reader.*
- ReturnCode reader_pass_beep (struct reader *reader)
  *Causes the reader to emit the PASS beep.*
- ReturnCode reader_fail_beep (struct reader *reader)
  *Causes the reader to emit the FAIL beep.*
- ReturnCode reader_ping (struct reader *reader)

*Pings a given reader.*

- ReturnCode reader_purge (struct reader *reader)

  *Purges a given reader.*

- int reader_read_packet (struct reader *reader, char *buf, int maxlen)

  *Reads a packet from a reader into a buffer.*

- int reader_txPacketRxReply (struct reader *reader, char *tx_packet, int tx_packet_len, char *rx_buf, int rx_buf_len, int *payload_index)

  *TODO.*

- ReturnCode reader_txPacketCheckAck (struct reader *reader, char *tx_packet, int tx_packet_len)

  *Returns if successfully acknowledged.*

- int reader_parse_poll_packet (struct reader *reader, struct list *tagList, char *packetPayload, int packetPayload_len)

  *Parses a packet returned from a reader poll.*

- int reader_poll15693 (struct reader *reader, list *tagList)

  *Causes the reader to poll for and ISO15693 tag.*

- int reader_pollTi (struct reader *reader, list *tagList)

  *Attempts to poll the reader for various tags.*

## 5.8.1 Detailed Description

Device library for an RFID reader. Particularly the DLP-RFID1.

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file reader.c.

## 5.8.2 Function Documentation

### 5.8.2.1 list* find_all_readers (struct list * *readers*)

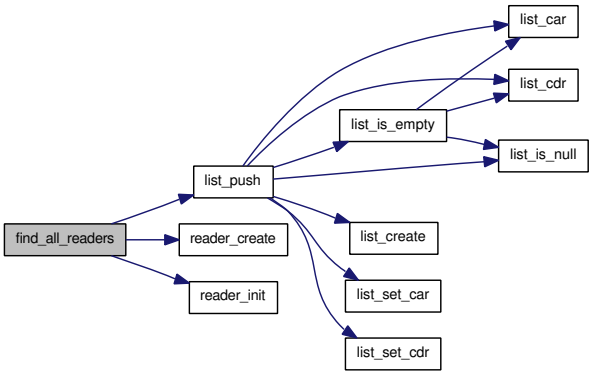
Returns a list of all connected readers.

**Parameters**

***readers*** Probably should be an empty list.

Definition at line 40 of file reader.c.

Here is the call graph for this function:

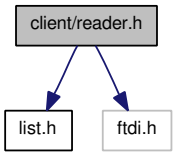



Here is the caller graph for this function:

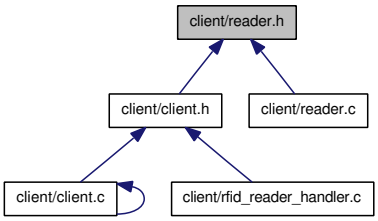
## 5.9 client/reader.h File Reference

```
#include "list.h"
#include <ftdi.h>
```

Include dependency graph for reader.h:




This graph shows which files directly or indirectly include this file:




### Data Structures

- struct reader

  *Completely identifies a currently connected RFID reader.*

- struct tag

  *An RFID identification.*

### Defines

- #define EDEVERR -1

  *Device error.*

- #define BEEP 1

  *True.*

- #define NOBEEP 0

  *False.*

- #define **RFID1_VID** 0x0403
- #define **RFID1_PID** 0xfbfc
- #define **RFID1_DESC** "DLP-RFID1"
- #define **RFID1_BAUDRATE** 115200
- #define **RFID1_PKT_PING** "0108000304FF0000"

- #define **RFID1_PKT_AGCTGL** "0109000304F0000000"
- #define **RFID1_PKT_AMPMTGL** "0109000304F1FF0000"
- #define **RFID1_PKT_PASSBEEP** "010900030419F00000"
- #define **RFID1_PKT_FAILBEEP** "010900030420F00000"
- #define **RFID1_PKT_15693INVRQ** "010B000304140601000000"
- #define **RFID1_PKT_TISIDPOLL** "010B000304340050000000"
- #define **RFID1_PKT_MODEMEM** "010C000304100021010000000"
- #define **RFID1_PKT_MODEUID** "010C00030410002101020000"
- #define **RFID1_PKT_TAGIT** "010C00030410002101130000"
- #define **RFID1_EOP_CHAR** '>'
- #define **RFID_ID_LEN** 20

## Enumerations

- enum ReturnCode {
  **RC_SUCCESS** = 0, **RC_ERROR**, **RC_NULL_ERROR**, **RC_IO_ERROR**,
  **RC_NO_PING**, **RC_NOT_CONNECTED** }
  *Possible return codes from reader.*

## Functions

- struct reader * reader_create ()
  *Return a newly allocated reader.*
- void reader_init (struct reader *reader, struct ftdi_context *ftdic, struct usb_device *dev)
  *Initialize a newly allocated reader.*
- struct tag * tag_create ()
  *Returns a newly allocated tag.*
- void tag_init (struct tag *tag, char *id, struct reader *parent)
  *Initializes a newly allocated tag.*
- list * find_all_readers (list *readers)
  *Returns a list of all connected readers.*
- ReturnCode reader_connect (struct reader *reader, int beep)
  *Connects to a given reader.*
- ReturnCode reader_disconnect (struct reader *reader)
  *Disconnects from a given reader.*
- ReturnCode reader_reset (struct reader *reader)
  *Resets a given reader.*
- ReturnCode reader_pass_beep (struct reader *reader)
  *Causes the reader to emit the PASS beep.*

- ReturnCode reader_fail_beep (struct reader *reader)

  *Causes the reader to emit the FAIL beep.*

- ReturnCode reader_ping (struct reader *reader)

  *Pings a given reader.*

- ReturnCode reader_purge (struct reader *reader)

  *Purges a given reader.*

- int reader_read_packet (struct reader *r, char *buf, int maxlen)

  *Reads a packet from a reader into a buffer.*

- int reader_txPacketRxReply (struct reader *reader, char *tx_packet, int tx_packet_len, char *rx_buf, int rx_buf_len, int *payload_index)

  *TODO.*

- ReturnCode reader_txPacketCheckAck (struct reader *reader, char *tx_packet, int tx_packet_len)

  *Returns if successfully acknowledged.*

- int reader_parse_poll_packet (struct reader *reader, struct list *tagList, char *packetPayload, int packetPayload_len)

  *Parses a packet returned from a reader poll.*

- int reader_poll15693 (struct reader *reader, list *tagList)

  *Causes the reader to poll for and ISO15693 tag.*

- int reader_pollTi (struct reader *reader, list *tagList)

  *Attempts to poll the reader for various tags.*

## 5.9.1 Detailed Description

Device library for an RFID reader. Particularly the DLP-RFID1.

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file reader.h.

## 5.9.2 Function Documentation

### 5.9.2.1 list* find_all_readers (struct list * *readers*)

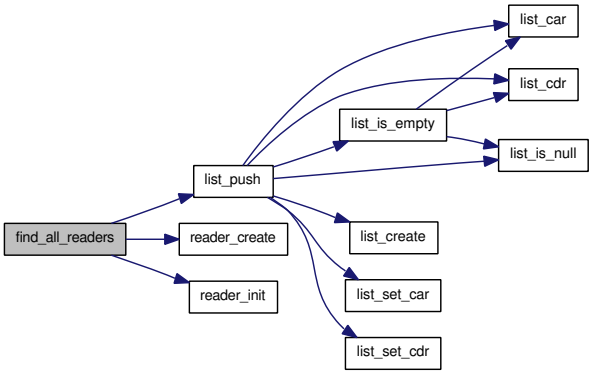
Returns a list of all connected readers.

### Parameters

***readers*** Probably should be an empty list.

Definition at line 40 of file reader.c.

Here is the call graph for this function:

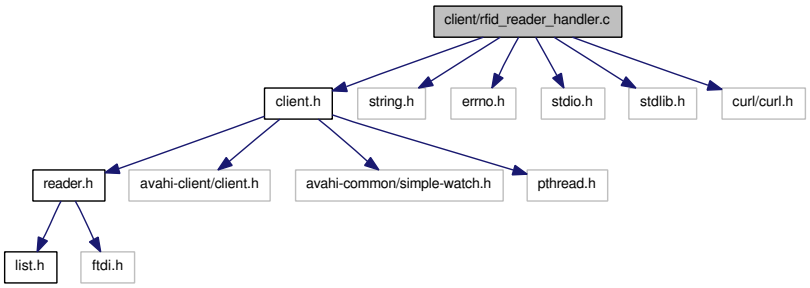



Here is the caller graph for this function:

# 5.10 client/rfid_reader_handler.c File Reference

```
#include "client.h"
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <curl/curl.h>
```

Include dependency graph for rfid_reader_handler.c:




## Defines

- #define SLEEP_BTWN_POLL 1

  *Amount of time in seconds between polls to RFID reader.*

- #define SLEEP_BTWN_SEARCH 1

  *Amount of time in seconds between polls to USB for RFID reader.*

## Functions

- int reader_handle_tag (const char *tag, struct client_config *config)
- int reader_poll_loop (struct reader *reader, struct client_config *config)
- void * reader_function (void *args)

  *RFID reading loop which polls for identifications. Look for rfid_reader_handler.c.*

### 5.10.1 Detailed Description

The RFID reader is discovered, initialized, and polled every few seconds. Upon reading a tag, a query is made to a server.

#### Author

Willi Ballenthin

**Date**

Spring, 2010

Definition in file rfid_reader_handler.c.

### 5.10.2 Function Documentation

#### 5.10.2.1 void* reader_function (void * *args*)

RFID reading loop which polls for identifications. Look for rfid_reader_handler.c.

PThread. Initializes an RFID device, if it exists, and begins a tag reading loop.

**See also**

SLEEP_BTWN_SEARCH

**Parameters**

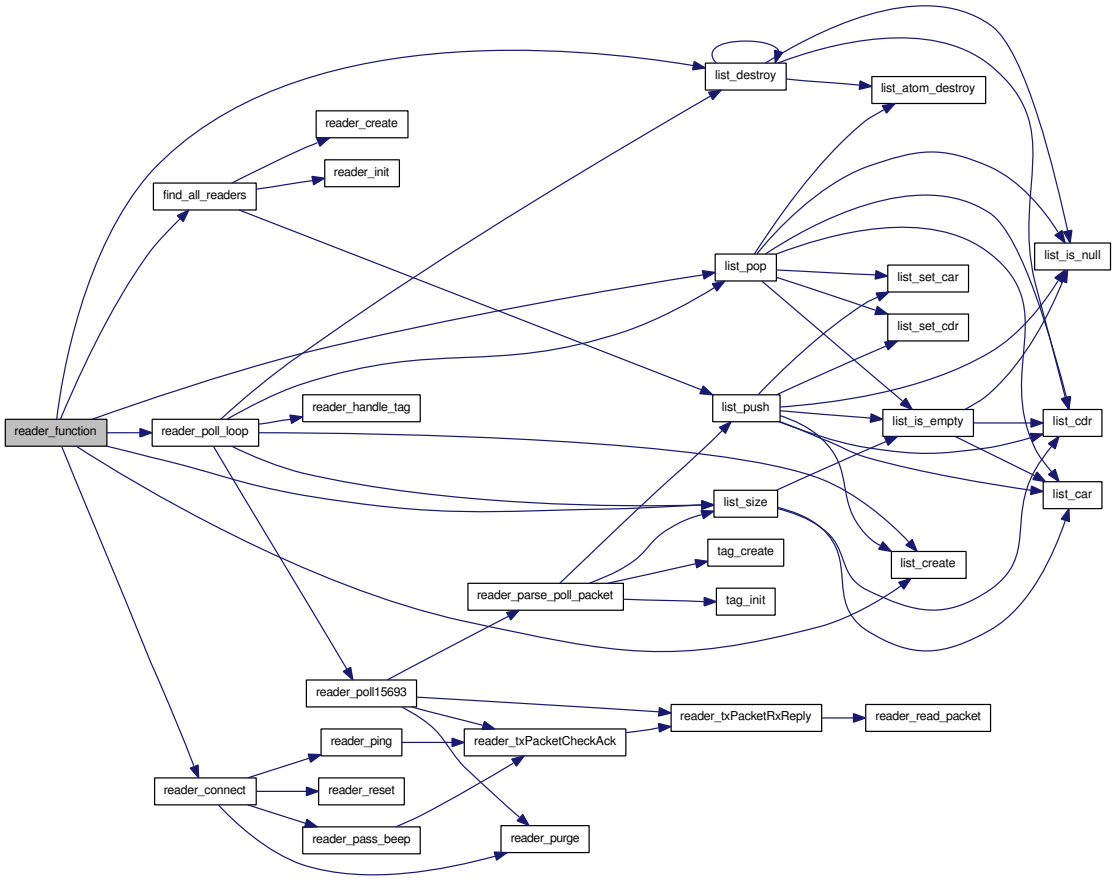
***args*** struct client_config - The current configuration of the client.

**Returns**

Should not return, if so, error. abstract RFID differences from this implementation

Definition at line 164 of file rfid_reader_handler.c.

Here is the call graph for this function:




### 5.10.2.2 int reader_handle_tag (const char * *tag*, struct client_config * *config*)

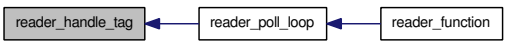
This function is invoked as a callback upon the reading of a tag.

**Parameters**

***tag*** A string containing the tag id.
***config*** The current configuration of the client.
***0*** on ok, negative integer otherwise.

Definition at line 34 of file rfid_reader_handler.c.

Here is the caller graph for this function:




### 5.10.2.3 int reader_poll_loop (struct reader * *reader*, struct client_config * *config*)

Repeatedly queries the device for RFID tags.

### See also

SLEEP_BTWN_POLL

### Parameters

***reader*** The RFID reading device.

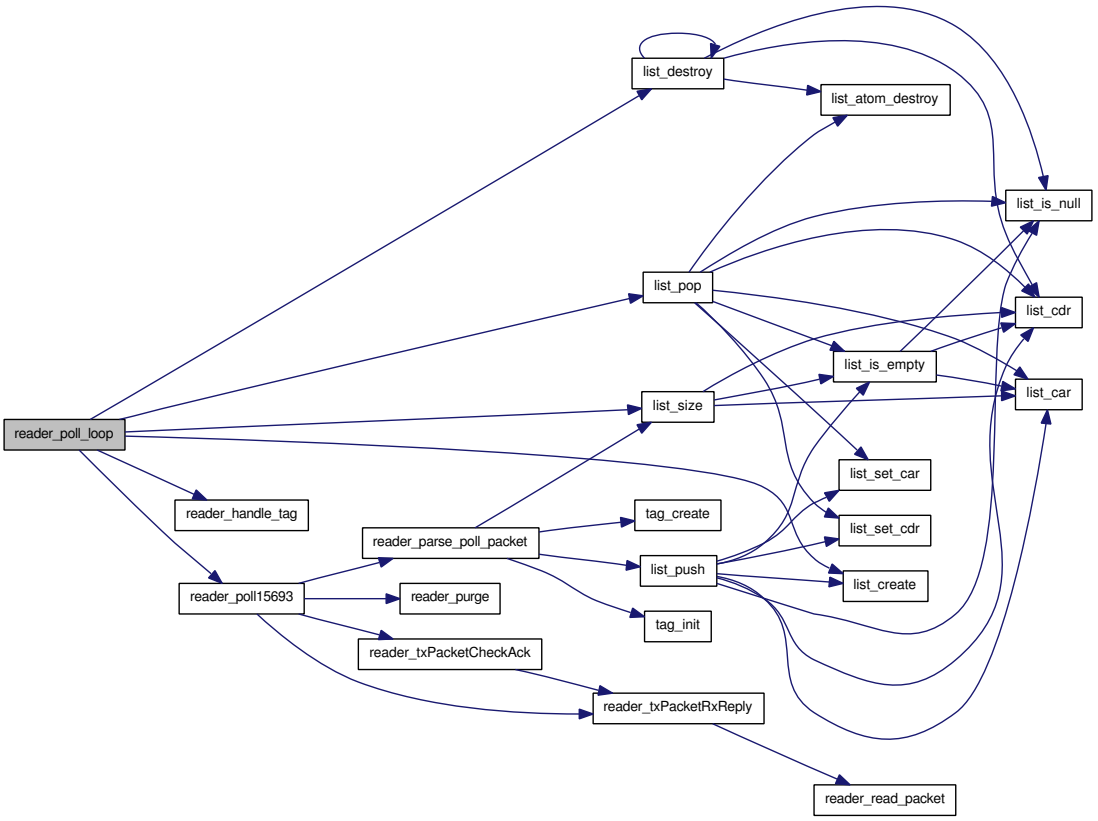
***config*** The current configuration of the client.

### Returns

Well it shouldnt. If it does, there's an error.

Definition at line 119 of file rfid_reader_handler.c.

Here is the call graph for this function:

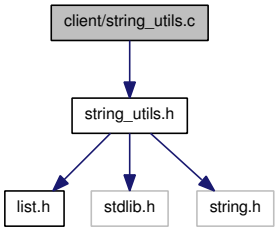



Here is the caller graph for this function:

# 5.11 client/string_utils.c File Reference

```
#include "string_utils.h"
```

Include dependency graph for string_utils.c:




## Functions

- char * string_join (list *strings, char *glue)

  *Given a list of strings, return a string glued together by GLUE.*

- list * string_split (char *line, char *delim)

## 5.11.1 Detailed Description

**Author**

Willi Ballenthin

**Date**

Spring, 2010

Definition in file string_utils.c.

## 5.11.2 Function Documentation

### 5.11.2.1 char* string_join (list * *strings*, char * *glue*)

Given a list of strings, return a string glued together by GLUE.

Given a string an set of delimiting characters, return a list of string.

**Parameters**

***strings*** A list of strings to join.

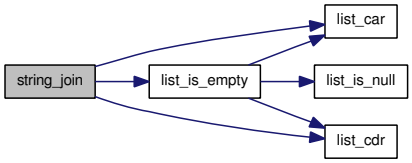
***glue*** A string to be used as glue.

**Returns**

A new string composed of the parts provided in the list glued together by the join string, or NULL on failure.

Definition at line 18 of file string_utils.c.

Here is the call graph for this function:




#### 5.11.2.2 list* string_split (char * *line*, char * *delim*)

split a string into a list of new substrings delimited by some set of characters the resulting list should be deep freed.

**Parameters**

***line*** The string to split.

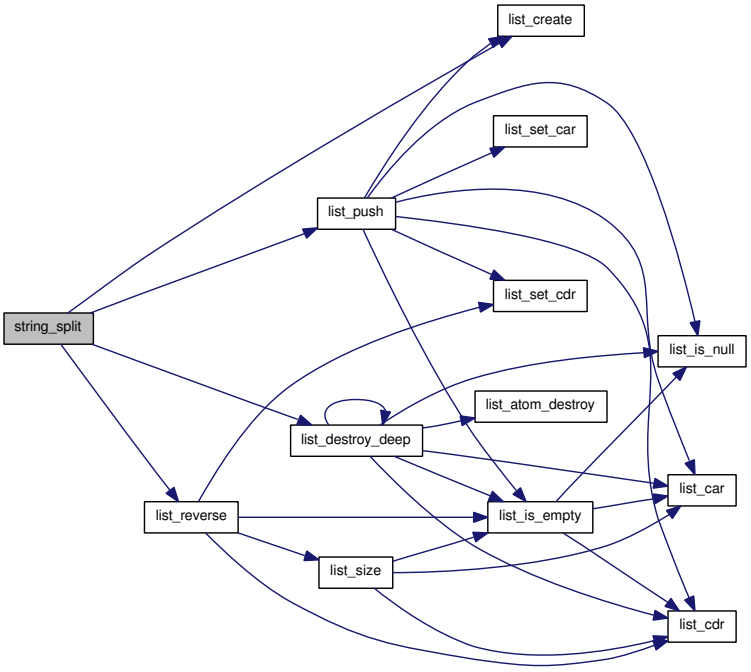
***delim*** A set of characters, each of which should cause a split.

**Returns**

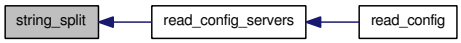
New list containing the fragments, or NULL on failure.

Definition at line 66 of file string_utils.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# Index