

TLDNS Relay

1.0

Generated by Doxygen 1.11.0



---

1 TLDNS Relay	1
1.1 System Function Design	1
1.1.1 Basic Tasks	1
1.1.2 Additional Functions	2
1.1.3 Extra Features	2
1.2 Quick Start	2
1.2.1 Quick Start	2
1.2.2 Program Help	3
1.3 Reference	3
2 Data Structure Index	5
2.1 Data Structures	5
3 File Index	7
3.1 File List	7
4 Data Structure Documentation	9
4.1 cache__ Struct Reference	9
4.1.1 Detailed Description	11
4.1.2 Field Documentation	11
4.1.2.1 head	11
4.1.2.2 insert	11
4.1.2.3 query	11
4.1.2.4 size	12
4.1.2.5 tail	12
4.1.2.6 tree	12
4.2 dns_header Struct Reference	12
4.2.1 Detailed Description	13
4.2.2 Field Documentation	13
4.2.2.1 aa	13
4.2.2.2 ancount	13
4.2.2.3 arcount	13
4.2.2.4 id	13
4.2.2.5 nscount	13
4.2.2.6 opcode	13

---

4.2.2.7 qdcount . . . . .	13
4.2.2.8 qr . . . . .	13
4.2.2.9 ra . . . . .	14
4.2.2.10 rcode . . . . .	14
4.2.2.11 rd . . . . .	14
4.2.2.12 tc . . . . .	14
4.2.2.13 z . . . . .	14
4.3 dns_msg Struct Reference . . . . .	14
4.3.1 Detailed Description . . . . .	15
4.3.2 Field Documentation . . . . .	15
4.3.2.1 header . . . . .	15
4.3.2.2 que . . . . .	15
4.3.2.3 rr . . . . .	15
4.4 dns_query Struct Reference . . . . .	16
4.4.1 Detailed Description . . . . .	16
4.4.2 Field Documentation . . . . .	17
4.4.2.1 addr . . . . .	17
4.4.2.2 id . . . . .	17
4.4.2.3 msg . . . . .	17
4.4.2.4 prev_id . . . . .	17
4.4.2.5 timer . . . . .	17
4.5 dns_question Struct Reference . . . . .	18
4.5.1 Detailed Description . . . . .	18
4.5.2 Field Documentation . . . . .	18
4.5.2.1 next . . . . .	18
4.5.2.2 qclass . . . . .	18
4.5.2.3 qname . . . . .	18
4.5.2.4 qtype . . . . .	19
4.6 dns_rr Struct Reference . . . . .	19
4.6.1 Detailed Description . . . . .	19
4.6.2 Field Documentation . . . . .	20
4.6.2.1 class . . . . .	20
4.6.2.2 name . . . . .	20
4.6.2.3 next . . . . .	20

---

4.6.2.4 rdata . . . . .	20
4.6.2.5 rdlength . . . . .	20
4.6.2.6 ttl . . . . .	20
4.6.2.7 type . . . . .	20
4.7 dns_rr_linklist Struct Reference . . . . .	21
4.7.1 Detailed Description . . . . .	21
4.7.2 Field Documentation . . . . .	22
4.7.2.1 delete_next . . . . .	22
4.7.2.2 expire_time . . . . .	22
4.7.2.3 insert . . . . .	22
4.7.2.4 next . . . . .	22
4.7.2.5 query_next . . . . .	22
4.7.2.6 value . . . . .	23
4.8 index_ Struct Reference . . . . .	23
4.8.1 Detailed Description . . . . .	23
4.8.2 Field Documentation . . . . .	24
4.8.2.1 id . . . . .	24
4.8.2.2 prev_id . . . . .	24
4.9 index_pool Struct Reference . . . . .	24
4.9.1 Detailed Description . . . . .	25
4.9.2 Field Documentation . . . . .	25
4.9.2.1 count . . . . .	25
4.9.2.2 delete . . . . .	25
4.9.2.3 destroy . . . . .	26
4.9.2.4 full . . . . .	26
4.9.2.5 insert . . . . .	26
4.9.2.6 pool . . . . .	26
4.9.2.7 query . . . . .	26
4.9.2.8 queue . . . . .	27
4.10 linklist_rbtrees Struct Reference . . . . .	27
4.10.1 Detailed Description . . . . .	28
4.10.2 Field Documentation . . . . .	29
4.10.2.1 insert . . . . .	29
4.10.2.2 query . . . . .	29

---

4.10.2.3 root . . . . .	29
4.11 query_pool Struct Reference . . . . .	30
4.11.1 Detailed Description . . . . .	31
4.11.2 Field Documentation . . . . .	31
4.11.2.1 cache . . . . .	31
4.11.2.2 count . . . . .	31
4.11.2.3 delete . . . . .	31
4.11.2.4 finish . . . . .	32
4.11.2.5 full . . . . .	32
4.11.2.6 insert . . . . .	32
4.11.2.7 ipool . . . . .	33
4.11.2.8 loop . . . . .	33
4.11.2.9 pool . . . . .	33
4.11.2.10 queue . . . . .	33
4.12 queue Struct Reference . . . . .	33
4.12.1 Detailed Description . . . . .	34
4.12.2 Field Documentation . . . . .	34
4.12.2.1 destroy . . . . .	34
4.12.2.2 head . . . . .	34
4.12.2.3 pop . . . . .	34
4.12.2.4 push . . . . .	35
4.12.2.5 q . . . . .	35
4.12.2.6 tail . . . . .	35
4.13 rbtree_node Struct Reference . . . . .	36
4.13.1 Detailed Description . . . . .	37
4.13.2 Field Documentation . . . . .	37
4.13.2.1 color . . . . .	37
4.13.2.2 key . . . . .	37
4.13.2.3 left . . . . .	37
4.13.2.4 parent . . . . .	37
4.13.2.5 right . . . . .	37
4.13.2.6 rr_list . . . . .	38
4.14 rbtree_value Struct Reference . . . . .	38
4.14.1 Detailed Description . . . . .	39

---

4.14.2 Field Documentation . . . . .	39
4.14.2.1 ancount . . . . .	39
4.14.2.2 arcount . . . . .	39
4.14.2.3 nscount . . . . .	39
4.14.2.4 rr . . . . .	39
4.14.2.5 type . . . . .	39
5 File Documentation . . . . .	41
5.1 include/cache.h File Reference . . . . .	41
5.1.1 Macro Definition Documentation . . . . .	42
5.1.1.1 CACHE_SIZE . . . . .	42
5.1.2 Typedef Documentation . . . . .	43
5.1.2.1 Cache . . . . .	43
5.1.3 Function Documentation . . . . .	43
5.1.3.1 new_cache() . . . . .	43
5.2 cache.h . . . . .	44
5.3 include/config.h File Reference . . . . .	44
5.3.1 Function Documentation . . . . .	45
5.3.1.1 init_config() . . . . .	45
5.3.2 Variable Documentation . . . . .	45
5.3.2.1 CLIENT_PORT . . . . .	45
5.3.2.2 HOSTS_PATH . . . . .	46
5.3.2.3 LOG_MASK . . . . .	46
5.3.2.4 LOG_PATH . . . . .	46
5.3.2.5 REMOTE_HOST . . . . .	46
5.4 config.h . . . . .	46
5.5 include/dns.h File Reference . . . . .	47
5.5.1 Macro Definition Documentation . . . . .	48
5.5.1.1 DNS_CLASS_IN . . . . .	48
5.5.1.2 DNS_OPCODE_IQUERY . . . . .	49
5.5.1.3 DNS_OPCODE_QUERY . . . . .	49
5.5.1.4 DNS_OPCODE_STATUS . . . . .	49
5.5.1.5 DNS_QR_ANSWER . . . . .	49
5.5.1.6 DNS_QR_QUERY . . . . .	49

5.5.1.7 DNS_RCODE_NXDOMAIN . . . . .	49
5.5.1.8 DNS_RCODE_OK . . . . .	49
5.5.1.9 DNS_RCODE_SERVFAIL . . . . .	49
5.5.1.10 DNS_RR_NAME_MAX_SIZE . . . . .	49
5.5.1.11 DNS_STRING_MAX_SIZE . . . . .	50
5.5.1.12 DNS_TYPE_A . . . . .	50
5.5.1.13 DNS_TYPE_AAAA . . . . .	50
5.5.1.14 DNS_TYPE_CNAME . . . . .	50
5.5.1.15 DNS_TYPE_HINFO . . . . .	50
5.5.1.16 DNS_TYPE_MINFO . . . . .	50
5.5.1.17 DNS_TYPE_MX . . . . .	50
5.5.1.18 DNS_TYPE_NS . . . . .	50
5.5.1.19 DNS_TYPE_PTR . . . . .	50
5.5.1.20 DNS_TYPE_SOA . . . . .	51
5.5.1.21 DNS_TYPE_TXT . . . . .	51
5.5.2 Typedef Documentation . . . . .	51
5.5.2.1 Dns_Header . . . . .	51
5.5.2.2 Dns_Msg . . . . .	51
5.5.2.3 Dns_Queue . . . . .	51
5.5.2.4 Dns_RR . . . . .	51
5.6 dns.h . . . . .	52
5.7 include/dns_client.h File Reference . . . . .	53
5.7.1 Function Documentation . . . . .	54
5.7.1.1 init_client() . . . . .	54
5.7.1.2 send_to_remote() . . . . .	55
5.8 dns_client.h . . . . .	55
5.9 include/dns_parse.h File Reference . . . . .	56
5.9.1 Function Documentation . . . . .	57
5.9.1.1 copy_dnsmmsg() . . . . .	57
5.9.1.2 copy_dnsrr() . . . . .	57
5.9.1.3 destroy_dnsmmsg() . . . . .	58
5.9.1.4 destroy_dnsrr() . . . . .	59
5.9.1.5 dnsmmsg_to_string() . . . . .	59
5.9.1.6 string_to_dnsmmsg() . . . . .	61



---

5.10 dns_parse.h . . . . .	61
5.11 include/dns_print.h File Reference . . . . .	62
5.11.1 Function Documentation . . . . .	63
5.11.1.1 print_dns_message() . . . . .	63
5.11.1.2 print_dns_string() . . . . .	64
5.12 dns_print.h . . . . .	64
5.13 include/dns_server.h File Reference . . . . .	64
5.13.1 Function Documentation . . . . .	65
5.13.1.1 init_server() . . . . .	65
5.13.1.2 send_to_local() . . . . .	66
5.14 dns_server.h . . . . .	67
5.15 include/index_pool.h File Reference . . . . .	68
5.15.1 Macro Definition Documentation . . . . .	69
5.15.1.1 INDEX_POOL_MAX_SIZE . . . . .	69
5.15.2 Typedef Documentation . . . . .	69
5.15.2.1 Index . . . . .	69
5.15.2.2 Index_Pool . . . . .	69
5.15.3 Function Documentation . . . . .	70
5.15.3.1 new_ipool() . . . . .	70
5.16 index_pool.h . . . . .	71
5.17 include/linklist_rbtrees.h File Reference . . . . .	71
5.17.1 Typedef Documentation . . . . .	73
5.17.1.1 Dns_RR_LinkList . . . . .	73
5.17.1.2 Rbtrees . . . . .	73
5.17.1.3 Rbtrees_Node . . . . .	74
5.17.1.4 Rbtrees_Value . . . . .	74
5.17.2 Enumeration Type Documentation . . . . .	74
5.17.2.1 Color . . . . .	74
5.17.3 Function Documentation . . . . .	74
5.17.3.1 new_linklist() . . . . .	74
5.17.3.2 new_rbtrees() . . . . .	75
5.18 linklist_rbtrees.h . . . . .	76
5.19 include/log.h File Reference . . . . .	77
5.19.1 Macro Definition Documentation . . . . .	78

---

---

5.19.1.1	<a href="#">log_debug</a>	78
5.19.1.2	<a href="#">log_error</a>	78
5.19.1.3	<a href="#">log_fatal</a>	79
5.19.1.4	<a href="#">log_info</a>	79
5.19.2	<a href="#">Variable Documentation</a>	79
5.19.2.1	<a href="#">log_file</a>	79
5.20	<a href="#">log.h</a>	79
5.21	<a href="#">include/query_pool.h File Reference</a>	80
5.21.1	<a href="#">Macro Definition Documentation</a>	82
5.21.1.1	<a href="#">QUERY_POOL_MAX_SIZE</a>	82
5.21.2	<a href="#">Typedef Documentation</a>	82
5.21.2.1	<a href="#">Dns_Query</a>	82
5.21.2.2	<a href="#">Query_Pool</a>	82
5.21.3	<a href="#">Function Documentation</a>	82
5.21.3.1	<a href="#">new_qpool()</a>	82
5.22	<a href="#">query_pool.h</a>	83
5.23	<a href="#">include/queue.h File Reference</a>	84
5.23.1	<a href="#">Macro Definition Documentation</a>	85
5.23.1.1	<a href="#">QUEUE_MAX_SIZE</a>	85
5.23.2	<a href="#">Typedef Documentation</a>	86
5.23.2.1	<a href="#">Queue</a>	86
5.23.3	<a href="#">Function Documentation</a>	86
5.23.3.1	<a href="#">new_queue()</a>	86
5.24	<a href="#">queue.h</a>	87
5.25	<a href="#">README.md File Reference</a>	87
5.26	<a href="#">src/cache.c File Reference</a>	87
5.26.1	<a href="#">Function Documentation</a>	88
5.26.1.1	<a href="#">BKDRHash()</a>	88
5.26.1.2	<a href="#">cache_insert()</a>	89
5.26.1.3	<a href="#">cache_query()</a>	90
5.26.1.4	<a href="#">get_min_ttl()</a>	91
5.26.1.5	<a href="#">new_cache()</a>	91
5.27	<a href="#">src/config.c File Reference</a>	92
5.27.1	<a href="#">Function Documentation</a>	93

---

---

5.27.1.1	<a href="#">init_config()</a>	93
5.27.2	<a href="#">Variable Documentation</a>	94
5.27.2.1	<a href="#">CLIENT_PORT</a>	94
5.27.2.2	<a href="#">HOSTS_PATH</a>	94
5.27.2.3	<a href="#">LOG_MASK</a>	94
5.27.2.4	<a href="#">LOG_PATH</a>	94
5.27.2.5	<a href="#">REMOTE_HOST</a>	95
5.28	<a href="#">src/dns_client.c File Reference</a>	95
5.28.1	<a href="#">Function Documentation</a>	96
5.28.1.1	<a href="#">alloc_buffer()</a>	96
5.28.1.2	<a href="#">init_client()</a>	96
5.28.1.3	<a href="#">on_read()</a>	97
5.28.1.4	<a href="#">on_send()</a>	98
5.28.1.5	<a href="#">send_to_remote()</a>	100
5.28.2	<a href="#">Variable Documentation</a>	101
5.28.2.1	<a href="#">client_socket</a>	101
5.28.2.2	<a href="#">local_addr</a>	101
5.28.2.3	<a href="#">qpool</a>	101
5.28.2.4	<a href="#">send_addr</a>	101
5.29	<a href="#">src/dns_parse.c File Reference</a>	102
5.29.1	<a href="#">Function Documentation</a>	103
5.29.1.1	<a href="#">copy_dnsmmsg()</a>	103
5.29.1.2	<a href="#">copy_dnsrr()</a>	104
5.29.1.3	<a href="#">destroy_dnsmmsg()</a>	105
5.29.1.4	<a href="#">destroy_dnsrr()</a>	106
5.29.1.5	<a href="#">dnshead_to_string()</a>	107
5.29.1.6	<a href="#">dnsmmsg_to_string()</a>	107
5.29.1.7	<a href="#">dnsque_to_string()</a>	108
5.29.1.8	<a href="#">dnsrr_to_string()</a>	109
5.29.1.9	<a href="#">read_uint16()</a>	110
5.29.1.10	<a href="#">read_uint32()</a>	111
5.29.1.11	<a href="#">rrname_to_string()</a>	112
5.29.1.12	<a href="#">string_to_dnshead()</a>	112
5.29.1.13	<a href="#">string_to_dnsmmsg()</a>	113

---

5.29.1.14	<a href="#">string_to_dnsque()</a>	114
5.29.1.15	<a href="#">string_to_dnsrr()</a>	115
5.29.1.16	<a href="#">string_to_rrname()</a>	116
5.29.1.17	<a href="#">write_uint16()</a>	117
5.29.1.18	<a href="#">write_uint32()</a>	118
5.30	<a href="#">src/dns_print.c File Reference</a>	119
5.30.1	<a href="#">Function Documentation</a>	120
5.30.1.1	<a href="#">print_dns_header()</a>	120
5.30.1.2	<a href="#">print_dns_message()</a>	120
5.30.1.3	<a href="#">print_dns_question()</a>	121
5.30.1.4	<a href="#">print_dns_rr()</a>	122
5.30.1.5	<a href="#">print_dns_string()</a>	123
5.30.1.6	<a href="#">print_rr_A()</a>	123
5.30.1.7	<a href="#">print_rr_AAAA()</a>	124
5.30.1.8	<a href="#">print_rr_CNAME()</a>	124
5.30.1.9	<a href="#">print_rr_MX()</a>	125
5.30.1.10	<a href="#">print_rr_SOA()</a>	126
5.31	<a href="#">src/dns_server.c File Reference</a>	126
5.31.1	<a href="#">Function Documentation</a>	128
5.31.1.1	<a href="#">alloc_buffer()</a>	128
5.31.1.2	<a href="#">init_server()</a>	128
5.31.1.3	<a href="#">on_read()</a>	129
5.31.1.4	<a href="#">on_send()</a>	130
5.31.1.5	<a href="#">send_to_local()</a>	132
5.31.2	<a href="#">Variable Documentation</a>	133
5.31.2.1	<a href="#">qpool</a>	133
5.31.2.2	<a href="#">recv_addr</a>	133
5.31.2.3	<a href="#">server_socket</a>	133
5.32	<a href="#">src/index_pool.c File Reference</a>	133
5.32.1	<a href="#">Function Documentation</a>	134
5.32.1.1	<a href="#">ipool_delete()</a>	134
5.32.1.2	<a href="#">ipool_destroy()</a>	135
5.32.1.3	<a href="#">ipool_full()</a>	135
5.32.1.4	<a href="#">ipool_insert()</a>	136

5.32.1.5	<code>ipool_query()</code>	137
5.32.1.6	<code>new_ipool()</code>	137
5.33	<code>src/linklist_rbtrees.c</code> File Reference	138
5.33.1	Function Documentation	140
5.33.1.1	<code>delete_case()</code>	140
5.33.1.2	<code>destroy_node()</code>	141
5.33.1.3	<code>grandparent()</code>	142
5.33.1.4	<code>insert_case()</code>	142
5.33.1.5	<code>linklist_delete_next()</code>	143
5.33.1.6	<code>linklist_insert()</code>	144
5.33.1.7	<code>linklist_query_next()</code>	144
5.33.1.8	<code>new_linklist()</code>	145
5.33.1.9	<code>new_rbtrees()</code>	146
5.33.1.10	<code>node_init()</code>	147
5.33.1.11	<code>rbtrees_delete()</code>	147
5.33.1.12	<code>rbtrees_find()</code>	148
5.33.1.13	<code>rbtrees_insert()</code>	149
5.33.1.14	<code>rbtrees_query()</code>	150
5.33.1.15	<code>rotate_left()</code>	151
5.33.1.16	<code>rotate_right()</code>	152
5.33.1.17	<code>sibling()</code>	153
5.33.1.18	<code>smallest_child()</code>	153
5.33.1.19	<code>uncle()</code>	154
5.33.2	Variable Documentation	155
5.33.2.1	<code>NIL</code>	155
5.34	<code>src/main.c</code> File Reference	155
5.34.1	Function Documentation	156
5.34.1.1	<code>main()</code>	156
5.34.2	Variable Documentation	157
5.34.2.1	<code>cache</code>	157
5.34.2.2	<code>log_file</code>	157
5.34.2.3	<code>loop</code>	158
5.34.2.4	<code>qpool</code>	158
5.35	<code>src/query_pool.c</code> File Reference	158

---

5.35.1 Function Documentation . . . . .	159
5.35.1.1 new_qpool() . . . . .	159
5.35.1.2 qpool_delete() . . . . .	160
5.35.1.3 qpool_finish() . . . . .	161
5.35.1.4 qpool_full() . . . . .	162
5.35.1.5 qpool_insert() . . . . .	163
5.35.1.6 qpool_query() . . . . .	164
5.35.1.7 timeout_cb() . . . . .	164
5.36 src/queue.c File Reference . . . . .	165
5.36.1 Function Documentation . . . . .	166
5.36.1.1 new_queue() . . . . .	166
5.36.1.2 queue_destroy() . . . . .	166
5.36.1.3 queue_pop() . . . . .	167
5.36.1.4 queue_push() . . . . .	167
<b>索引</b>	169

# Chapter 1

## TLDNS Relay

### 1.1 System Function Design

#### 1.1.1 Basic Tasks

Design a DNS relay server program that reads a "Domain Name - IP Address" mapping table. When a client queries the IP address corresponding to a domain name, the domain name is searched in the table, resulting in three possible outcomes:

- If the result is an IP address 0.0.0.0, return an error message "Domain name does not exist" to the client instead of returning the IP address 0.0.0.0, implementing a malicious website blocking function.
- If the result is a regular IP address, return this address to the client, implementing DNS server functionality.
- If the domain name is not found in the table, send the query to an Internet DNS server and return the result to the client, implementing DNS relay functionality.

The implementation must adhere to the DNS protocol specifications to ensure interoperability with Windows and other systems.

Notes:

1. Concurrent Clients: Allow concurrent queries from multiple clients (which may be on different computers). This means processing another client's query request even if the first query has not been answered yet (the role of the ID field in the DNS protocol header), requiring message ID translation.
2. Timeout Handling: Consider the unreliability of UDP, and handle situations where the external DNS server (relay) does not respond or responds late.

### 1.1.2 Additional Functions

- Implement LRU mechanism for Cache.
- Optimize the dictionary lookup algorithm.
- Ensure consistent performance across Windows/Linux source code.

### 1.1.3 Extra Features

- Support for IPv6.
- Cross-platform support for Windows/Linux/MacOS.
- Implement high-performance querying using an event-driven, non-blocking asynchronous I/O model.
- Implement query pools and index pools to support concurrent queries.
- Support multiple message types, including A, CNAME, SOA, MX, and AAAA.
- Provide command-line argument parsing and help documentation.

## 1.2 Quick Start

### 1.2.1 Quick Start

1. Clone the repository locally and navigate into it:  

```
git clone https://github.com/Word2VecT/TLDNS-Relay.git  
cd TLDNS-Relay
```
2. Download and install [libuv](#).
3. Import the project folder in CLion, compile, and run.
4. Set your DNS to 127.0.0.1.
5. Enjoy!



## 1.2.2 Program Help

Use the `-h` parameter to view the program help documentation.

Usage:

`[-a]` Use the specified name server  
`[-d]` Debug level mask, a 4-bit binary number, DEBUG, INFO, ERROR, FATAL in order  
`[-f]` Use the specified DNS hosts file  
`[-l]` Log information storage location  
`[-p]` Custom listening ports  
`[-h]` Helpful Information

Example:

`-d 1111 -a 192.168.0.1 -f c:\dns-table.txt`

Output all debugging information

Use the specified name server 192.168.0.1

Use the specified configuration file `c:\dns-table.txt`

`-d 1101 -l /Users/Code -p 53`

Output DEBUG, INFO, and FATAL information

Output debugging information to `/Users/Code` as a file

## 1.3 Reference

- [Domain names - concepts and facilities](#). RFC 1034, RFC Editor, November 1987, DOI: 10.17487/RFC1034. 55 pages. Abstract: This RFC is the revised basic definition of The Domain Name System. It obsoletes RFC-882. This memo describes the domain style names and their use for host address look up and electronic mail forwarding. It discusses the clients and servers in the domain name system and the protocol used between them.
- [Domain names - implementation and specification](#). RFC 1035, RFC Editor, November 1987, DOI: 10.17487/RFC1035. 55 pages. Abstract: This RFC is the revised specification of the protocol and format used in the implementation of the Domain Name System. It obsoletes RFC-883. This memo documents the details of the domain name client-server communication.
- Stroustrup, Bjarne. The C++ Programming Language. Pearson Education, 2013.
- Wikipedia. [红黑树](#) — [Wikipedia, The Free Encyclopedia](#). [Online; accessed 01-July-2024].



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">cache__</a>	Cash struct . . . . .	9
<a href="#">dns_header</a>	Header Section structure of DNS message . . . . .	12
<a href="#">dns_msg</a>	DNS message structure . . . . .	14
<a href="#">dns_query</a>	DNS query structure . . . . .	16
<a href="#">dns_question</a>	Question Section structure of DNS message, represented as a linked list . . . . .	18
<a href="#">dns_rr</a>	Resource Record structure of DNS message, represented as a linked list . . . . .	19
<a href="#">dns_rr_linklist</a>	Linked list of Red-Black Tree nodes . . . . .	21
<a href="#">index__</a>	Index structure . . . . .	23
<a href="#">index_pool</a>	Index pool . . . . .	24
<a href="#">linklist_rbtrees</a>	Red-Black Tree . . . . .	27
<a href="#">query_pool</a>	DNS query pool . . . . .	30
<a href="#">queue</a>	Circular queue . . . . .	33

[rbtree\\_node](#)

Node of the Red-Black Tree . . . . . 36

[rbtree\\_value](#)

Value of a Red-Black Tree node's linked list, corresponding to an answer for a  
specific query . . . . . 38

# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/ <a href="#">cache.h</a>	41
include/ <a href="#">config.h</a>	44
include/ <a href="#">dns.h</a>	47
include/ <a href="#">dns_client.h</a>	53
include/ <a href="#">dns_parse.h</a>	56
include/ <a href="#">dns_print.h</a>	62
include/ <a href="#">dns_server.h</a>	64
include/ <a href="#">index_pool.h</a>	68
include/ <a href="#">linklist_rbtrees.h</a>	71
include/ <a href="#">log.h</a>	77
include/ <a href="#">query_pool.h</a>	80
include/ <a href="#">queue.h</a>	84
src/ <a href="#">cache.c</a>	87
src/ <a href="#">config.c</a>	92
src/ <a href="#">dns_client.c</a>	95
src/ <a href="#">dns_parse.c</a>	102
src/ <a href="#">dns_print.c</a>	119
src/ <a href="#">dns_server.c</a>	126
src/ <a href="#">index_pool.c</a>	133
src/ <a href="#">linklist_rbtrees.c</a>	138
src/ <a href="#">main.c</a>	155
src/ <a href="#">query_pool.c</a>	158
src/ <a href="#">queue.c</a>	165



## Chapter 4

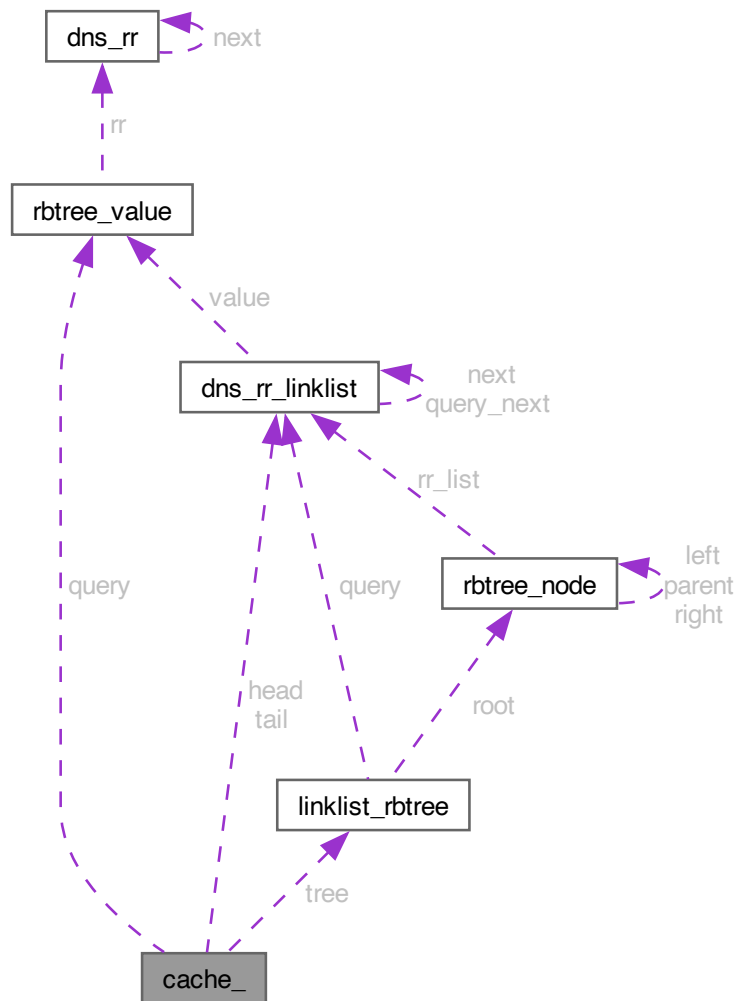
# Data Structure Documentation

### 4.1 cache\_\_ Struct Reference

Cash struct.

```
#include <cache.h>
```

Collaboration diagram for `cache_`:



## Data Fields

- `Dns_RR_LinkList * head`  
LRU header node.
- `Dns_RR_LinkList * tail`  
LRU tail node.
- `int size`  
LRU size.
- `Rbtree * tree`  
Red-black tree.
- `void(* insert)(struct cache_ *cache, const Dns_Msg *msg)`



Insert a DNS message into the cache.

- [Rbtree\\_Value](#) [\\*\(\\* query\)](#)(struct [cache\\_](#) \*cache, const [Dns\\_Que](#) \*que)

Query the cache for a DNS question.

### 4.1.1 Detailed Description

Cash struct.

### 4.1.2 Field Documentation

#### 4.1.2.1 head

[Dns\\_RR\\_LinkList](#)\* cache\_::head

LRU header node.

#### 4.1.2.2 insert

void(\* cache\_::insert) (struct [cache\\_](#) \*cache, const [Dns\\_Msg](#) \*msg)

Insert a DNS message into the cache.

Parameters

cache	The cache where the message will be inserted.
msg	The DNS message to be inserted.

#### 4.1.2.3 query

[Rbtree\\_Value](#) [\\*\(\\* cache\\_::query\)](#) (struct [cache\\_](#) \*cache, const [Dns\\_Que](#) \*que)

Query the cache for a DNS question.

Parameters

cache	The cache to query.
que	The DNS question.

Returns

The value found in the cache or NULL if not found.

#### 4.1.2.4 size

`int cache_::size`

LRU size.

#### 4.1.2.5 tail

`Dns_RR_LinkList* cache_::tail`

LRU tail node.

#### 4.1.2.6 tree

`Rbtree* cache_::tree`

Red-black tree.

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

- include/[cache.h](#)

## 4.2 dns\_header Struct Reference

Header Section structure of DNS message.

```
#include <dns.h>
```

### Data Fields

- `uint16_t id`
- `uint8_t qr: 1`
- `uint8_t opcode: 4`
- `uint8_t aa: 1`
- `uint8_t tc: 1`
- `uint8_t rd: 1`
- `uint8_t ra: 1`
- `uint8_t z: 3`
- `uint8_t rcode: 4`
- `uint16_t qdcount`
- `uint16_t ancourt`
- `uint16_t nscount`
- `uint16_t arcount`

### 4.2.1 Detailed Description

Header Section structure of DNS message.

### 4.2.2 Field Documentation

#### 4.2.2.1 aa

uint8\_t dns\_header::aa

#### 4.2.2.2 ancount

uint16\_t dns\_header::ancount

#### 4.2.2.3 arcount

uint16\_t dns\_header::arcount

#### 4.2.2.4 id

uint16\_t dns\_header::id

#### 4.2.2.5 nscount

uint16\_t dns\_header::nscount

#### 4.2.2.6 opcode

uint8\_t dns\_header::opcode

#### 4.2.2.7 qdcount

uint16\_t dns\_header::qdcount

#### 4.2.2.8 qr

uint8\_t dns\_header::qr

#### 4.2.2.9 ra

uint8\_t dns\_header::ra

#### 4.2.2.10 rcode

uint8\_t dns\_header::rcode

#### 4.2.2.11 rd

uint8\_t dns\_header::rd

#### 4.2.2.12 tc

uint8\_t dns\_header::tc

#### 4.2.2.13 z

uint8\_t dns\_header::z

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

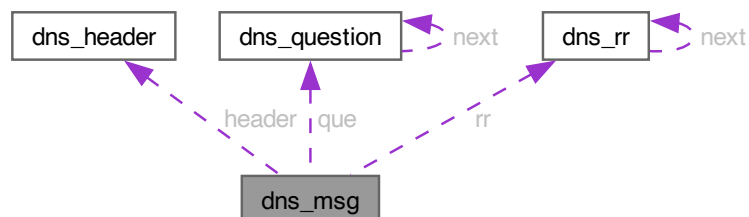
- [include/dns.h](#)

### 4.3 dns\_msg Struct Reference

DNS message structure.

```
#include <dns.h>
```

Collaboration diagram for dns\_msg:



## Data Fields

- [Dns\\_Header](#) \* [header](#)  
Pointer to the Header Section.
- [Dns\\_Que](#) \* [que](#)  
Pointer to the head node of the Question Section linked list.
- [Dns\\_RR](#) \* [rr](#)  
Pointer to the head node of the Resource Record linked list.

### 4.3.1 Detailed Description

DNS message structure.

### 4.3.2 Field Documentation

#### 4.3.2.1 header

[Dns\\_Header](#)\* dns\_msg::header

Pointer to the Header Section.

#### 4.3.2.2 que

[Dns\\_Que](#)\* dns\_msg::que

Pointer to the head node of the Question Section linked list.

#### 4.3.2.3 rr

[Dns\\_RR](#)\* dns\_msg::rr

Pointer to the head node of the Resource Record linked list.

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

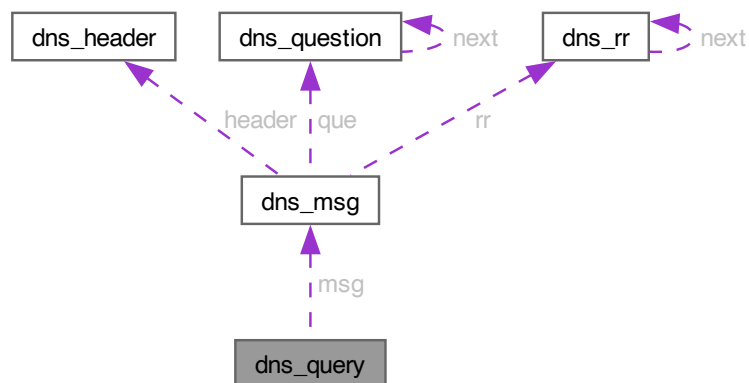
- include/[dns.h](#)

## 4.4 dns\_query Struct Reference

DNS query structure.

```
#include <query_pool.h>
```

Collaboration diagram for dns\_query:



### Data Fields

- `uint16_t id`  
Query ID.
- `uint16_t prev_id`  
Original DNS query message ID.
- `struct sockaddr addr`  
Address of the requester.
- `Dns_Msg * msg`  
DNS query message.
- `uv_timer_t timer`  
Timer.

### 4.4.1 Detailed Description

DNS query structure.

## 4.4.2 Field Documentation

### 4.4.2.1 addr

```
struct sockaddr dns_query::addr
```

Address of the requester.

### 4.4.2.2 id

```
uint16_t dns_query::id
```

Query ID.

### 4.4.2.3 msg

```
Dns_Msg* dns_query::msg
```

DNS query message.

### 4.4.2.4 prev\_id

```
uint16_t dns_query::prev_id
```

Original DNS query message ID.

### 4.4.2.5 timer

```
uv_timer_t dns_query::timer
```

Timer.

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

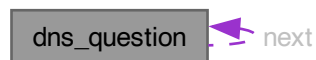
- [include/query\\_pool.h](#)

## 4.5 dns\_question Struct Reference

Question Section structure of DNS message, represented as a linked list.

```
#include <dns.h>
```

Collaboration diagram for dns\_question:



### Data Fields

- uint8\_t \* [qname](#)
- uint16\_t [qtype](#)
- uint16\_t [qclass](#)
- struct [dns\\_question](#) \* [next](#)

### 4.5.1 Detailed Description

Question Section structure of DNS message, represented as a linked list.

### 4.5.2 Field Documentation

#### 4.5.2.1 next

```
struct dns\_question* dns_question::next
```

#### 4.5.2.2 qclass

```
uint16_t dns_question::qclass
```

#### 4.5.2.3 qname

```
uint8_t* dns_question::qname
```



#### 4.5.2.4 qtype

uint16\_t dns\_question::qtype

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

- include/[dns.h](#)

## 4.6 dns\_rr Struct Reference

Resource Record structure of DNS message, represented as a linked list.

```
#include <dns.h>
```

Collaboration diagram for dns\_rr:



### Data Fields

- uint8\_t \* [name](#)
- uint16\_t [type](#)
- uint16\_t [class](#)
- uint32\_t [ttl](#)
- uint16\_t [rdlength](#)
- uint8\_t \* [rdata](#)
- struct [dns\\_rr](#) \* [next](#)

#### 4.6.1 Detailed Description

Resource Record structure of DNS message, represented as a linked list.

## 4.6.2 Field Documentation

### 4.6.2.1 class

`uint16_t dns_rr::class`

### 4.6.2.2 name

`uint8_t* dns_rr::name`

### 4.6.2.3 next

`struct dns\_rr* dns_rr::next`

### 4.6.2.4 rdata

`uint8_t* dns_rr::rdata`

### 4.6.2.5 rdlength

`uint16_t dns_rr::rdlength`

### 4.6.2.6 ttl

`uint32_t dns_rr::ttl`

### 4.6.2.7 type

`uint16_t dns_rr::type`

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

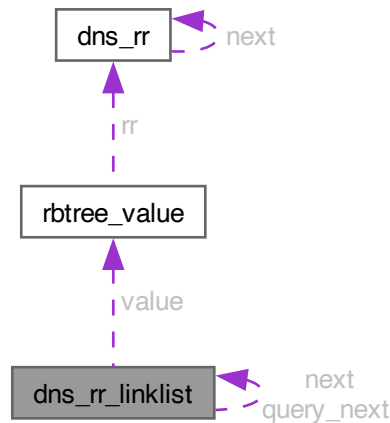
- `include/dns.h`

## 4.7 dns\_rr\_linklist Struct Reference

Linked list of Red-Black Tree nodes.

```
#include <linklist_rbtrees.h>
```

Collaboration diagram for dns\_rr\_linklist:



### Data Fields

- `Rbtrees_Value * value`  
Pointer to the value of the current linked list node.
- `time_t expire_time`  
Expiration time.
- `struct dns_rr_linklist * next`  
Pointer to the next node in the linked list.
- `void(* insert)(struct dns_rr_linklist *list, struct dns_rr_linklist *new_list_node)`  
Insert a key-value pair into the red-black tree.
- `void(* delete_next)(struct dns_rr_linklist *list)`  
Delete the next element in the linked list.
- `struct dns_rr_linklist *(* query_next)(struct dns_rr_linklist *list, const uint8_t *qname, const uint16_t qtype)`  
Query the next element in the linked list.

### 4.7.1 Detailed Description

Linked list of Red-Black Tree nodes.

## 4.7.2 Field Documentation

### 4.7.2.1 delete\_next

```
void(* dns_rr_linklist::delete_next) (struct dns\_rr\_linklist *list)
```

Delete the next element in the linked list.

Parameters

list	The linked list
------	-----------------

### 4.7.2.2 expire\_time

```
time_t dns_rr_linklist::expire_time
```

Expiration time.

### 4.7.2.3 insert

```
void(* dns_rr_linklist::insert) (struct dns\_rr\_linklist *list, struct dns\_rr\_linklist *new_list_node)
```

Insert a key-value pair into the red-black tree.

Parameters

tree	The red-black tree
key	The key
list	The value

### 4.7.2.4 next

```
struct dns\_rr\_linklist* dns_rr_linklist::next
```

Pointer to the next node in the linked list.

### 4.7.2.5 query\_next

```
struct dns\_rr\_linklist *(* dns_rr_linklist::query_next) (struct dns\_rr\_linklist *list, const uint8_t *qname, const uint16_t qtype)
```

Query the next element in the linked list.

## Parameters

list	The linked list
qname	The query name
qtype	The query type

## Returns

The queried element if found, otherwise NULL

## 4.7.2.6 value

[Rbtree\\_Value\\*](#) dns\_rr\_linklist::value

Pointer to the value of the current linked list node.

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

- include/[linklist\\_rbtrees.h](#)

## 4.8 index\_\_ Struct Reference

Index structure.

```
#include <index_pool.h>
```

## Data Fields

- [uint16\\_t](#) [id](#)  
The ID of the sent DNS query message.
- [uint16\\_t](#) [prev\\_id](#)  
The corresponding query ID.

## 4.8.1 Detailed Description

Index structure.

## 4.8.2 Field Documentation

### 4.8.2.1 id

```
uint16_t index_::id
```

The ID of the sent DNS query message.

### 4.8.2.2 prev\_id

```
uint16_t index_::prev_id
```

The corresponding query ID.

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

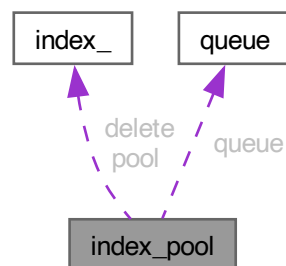
- [include/index\\_pool.h](#)

## 4.9 index\_pool Struct Reference

Index pool.

```
#include <index_pool.h>
```

Collaboration diagram for index\_pool:



## Data Fields

- [Index](#) \* [pool](#) [[INDEX\\_POOL\\_MAX\\_SIZE](#)]  
Index pool.
- unsigned short [count](#)  
Number of indices in the pool.
- [Queue](#) \* [queue](#)  
Queue of unallocated indices.
- [bool](#)(\* [full](#) )(struct [index\\_pool](#) \*ipool)  
Check if the index pool is full.
- [uint16\\_t](#)(\* [insert](#) )(struct [index\\_pool](#) \*ipool, [Index](#) \*req)  
Insert an index into the pool.
- [bool](#)(\* [query](#) )(struct [index\\_pool](#) \*ipool, [uint16\\_t](#) index)  
Query if an index exists in the pool.
- [Index](#) \*(\* [delete](#) )(struct [index\\_pool](#) \*ipool, [uint16\\_t](#) index)  
Delete an index from the pool.
- [void](#)(\* [destroy](#) )(struct [index\\_pool](#) \*ipool)  
Destroy the index pool.

## 4.9.1 Detailed Description

Index pool.

## 4.9.2 Field Documentation

## 4.9.2.1 count

unsigned short [index\\_pool::count](#)

Number of indices in the pool.

## 4.9.2.2 delete

[Index](#) \*(\* [index\\_pool::delete](#) )(struct [index\\_pool](#) \*ipool, [uint16\\_t](#) index)

Delete an index from the pool.

## Parameters

<a href="#">ipool</a>	The index pool
<a href="#">index</a>	The index to delete

## Returns

The deleted index

#### 4.9.2.3 destroy

```
void(* index_pool::destroy) (struct index\_pool *ipool)
```

Destroy the index pool.

Parameters

ipool	The index pool to destroy
-------	---------------------------

#### 4.9.2.4 full

```
bool(* index_pool::full) (struct index\_pool *ipool)
```

Check if the index pool is full.

Parameters

ipool	The index pool
-------	----------------

Returns

True if the index pool is full, false otherwise

#### 4.9.2.5 insert

```
uint16_t(* index_pool::insert) (struct index\_pool *ipool, Index *req)
```

Insert an index into the pool.

Parameters

ipool	The index pool
req	The index to insert

Returns

The ID of the inserted index

#### 4.9.2.6 pool

```
Index* index_pool::pool[INDEX\_POOL\_MAX\_SIZE]
```

Index pool.

#### 4.9.2.7 query

```
bool(* index_pool::query) (struct index\_pool *ipool, uint16_t index)
```

Query if an index exists in the pool.



## Parameters

ipool	The index pool
index	The index to query

## Returns

True if the index exists, false otherwise

## 4.9.2.8 queue

[Queue\\*](#) index\_pool::queue

Queue of unallocated indices.

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

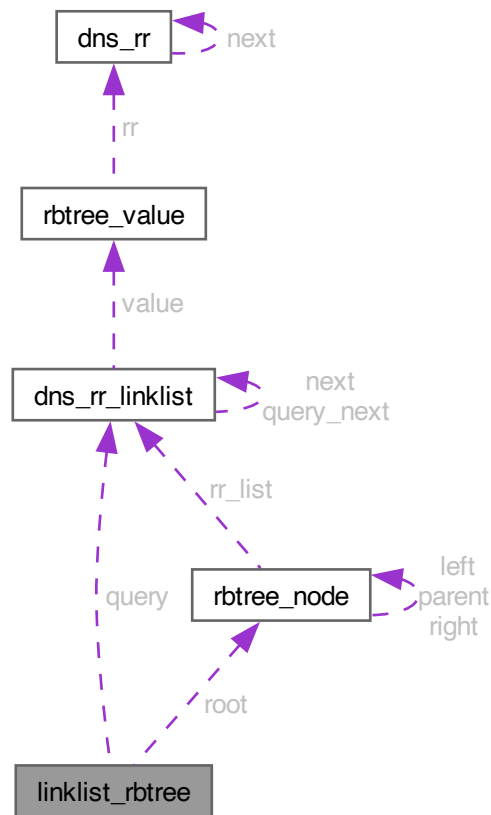
- include/[index\\_pool.h](#)

## 4.10 linklist\_rbtrees Struct Reference

Red-Black Tree.

```
#include <linklist_rbtrees.h>
```

Collaboration diagram for linklist\_rbtree:



## Data Fields

- [Rbtree\\_Node](#) \* **root**  
Pointer to the root node of the Red-Black Tree.
- void(\* [insert](#) )(struct [linklist\\_rbtree](#) \*tree, unsigned int key, [Dns\\_RR\\_LinkList](#) \*list)  
Insert a key-value pair into the red-black tree.
- [Dns\\_RR\\_LinkList](#) \*(\* [query](#) )(struct [linklist\\_rbtree](#) \*tree, unsigned int data)  
Query the red-black tree for a key.

### 4.10.1 Detailed Description

Red-Black Tree.

## 4.10.2 Field Documentation

### 4.10.2.1 insert

```
void(* linklist_rbtrees::insert) (struct linklist\_rbtrees *tree, unsigned int key, Dns\_RR\_LinkList *list)
```

Insert a key-value pair into the red-black tree.

Parameters

tree	The red-black tree
key	The key
list	The value

### 4.10.2.2 query

```
Dns\_RR\_LinkList *(* linklist_rbtrees::query) (struct linklist\_rbtrees *tree, unsigned int data)
```

Query the red-black tree for a key.

Parameters

tree	The red-black tree
key	The key to query

Returns

The linked list of the value if found, otherwise NULL

### 4.10.2.3 root

```
Rbtrees\_Node* linklist_rbtrees::root
```

Pointer to the root node of the Red-Black Tree.

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

- include/[linklist\\_rbtrees.h](#)



Event loop.

- [Cache](#) \* [cache](#)

Cache.

- [bool](#)(\* [full](#) )(struct [query\\_pool](#) \*[qpool](#))

Check if the query pool is full.

- [void](#)(\* [insert](#) )(struct [query\\_pool](#) \*[qpool](#), const struct sockaddr \*[addr](#), const [Dns\\_Msg](#) \*[msg](#))

Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.

- [void](#)(\* [finish](#) )(struct [query\\_pool](#) \*[qpool](#), const [Dns\\_Msg](#) \*[msg](#))

Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.

- [void](#)(\* [delete](#) )(struct [query\\_pool](#) \*[qpool](#), [uint16\\_t](#) [id](#))

Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

### 4.11.1 Detailed Description

DNS query pool.

### 4.11.2 Field Documentation

#### 4.11.2.1 [cache](#)

[Cache](#)\* [query\\_pool::cache](#)

Cache.

#### 4.11.2.2 [count](#)

unsigned short [query\\_pool::count](#)

Number of queries in the pool.

#### 4.11.2.3 [delete](#)

[void](#)(\* [query\\_pool::delete](#) ) (struct [query\\_pool](#) \*[qpool](#), [uint16\\_t](#) [id](#))

Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

## Parameters

qpool	The query pool
id	The ID of the query to be deleted

## 4.11.2.4 finish

```
void(* query_pool::finish) (struct query_pool *qpool, const Dns_Msg *msg)
```

Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.

## Parameters

qpool	The query pool
msg	The DNS message containing the response

## 4.11.2.5 full

```
bool(* query_pool::full) (struct query_pool *qpool)
```

Check if the query pool is full.

## Parameters

this	The query pool
------	----------------

## Returns

true if the query pool is full, false otherwise

## 4.11.2.6 insert

```
void(* query_pool::insert) (struct query_pool *qpool, const struct sockaddr *addr, const Dns_Msg *msg)
```

Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.

## Parameters

qpool	The query pool
addr	The address of the client
msg	The DNS message containing the query

## 4.11.2.7 ipool

[Index\\_Pool\\*](#) query\_pool::ipool

Index pool.

## 4.11.2.8 loop

[uv\\_loop\\_t\\*](#) query\_pool::loop

Event loop.

## 4.11.2.9 pool

[Dns\\_Query\\*](#) query\_pool::pool[[QUERY\\_POOL\\_MAX\\_SIZE](#)]

Query pool.

## 4.11.2.10 queue

[Queue\\*](#) query\_pool::queue

Queue of unassigned query IDs.

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

- [include/query\\_pool.h](#)

## 4.12 queue Struct Reference

Circular queue.

```
#include <queue.h>
```

## Data Fields

- `uint16_t q [QUEUE_MAX_SIZE]`  
The queue.
- unsigned short `head`  
The head of the queue.
- unsigned short `tail`  
The tail of the queue.
- `void(* push )(struct queue *queue, uint16_t num)`  
Push a number onto the queue.
- `uint16_t(* pop )(struct queue *queue)`  
Pop a number from the queue.
- `void(* destroy )(struct queue *queue)`  
Destroy the queue.

### 4.12.1 Detailed Description

Circular queue.

### 4.12.2 Field Documentation

#### 4.12.2.1 `destroy`

`void(* queue::destroy) (struct queue *queue)`

Destroy the queue.

#### Parameters

<code>queue</code>	The queue to destroy
--------------------	----------------------

#### 4.12.2.2 `head`

`unsigned short queue::head`

The head of the queue.

#### 4.12.2.3 `pop`

`uint16_t(* queue::pop) (struct queue *queue)`

Pop a number from the queue.



## Parameters

queue	The queue
-------	-----------

## Returns

The number popped from the queue

## 4.12.2.4 push

```
void(* queue::push) (struct queue *queue, uint16_t num)
```

Push a number onto the queue.

## Parameters

queue	The queue
num	The number to push

## 4.12.2.5 q

```
uint16_t queue::q[QUEUE\_MAX\_SIZE]
```

The queue.

## 4.12.2.6 tail

```
unsigned short queue::tail
```

The tail of the queue.

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

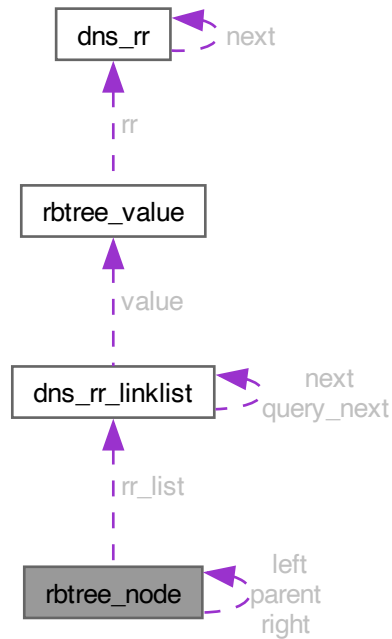
- [include/queue.h](#)

## 4.13 rbtree\_\_node Struct Reference

Node of the Red-Black Tree.

```
#include <linklist_rbtree.h>
```

Collaboration diagram for rbtree\_\_node:



### Data Fields

- unsigned int `key`  
Key of the Red-Black Tree node.
- `Dns_RR_LinkList * rr_list`  
Pointer to the linked list corresponding to the current node.
- `Color color`  
Color of the current node.
- `struct rbtree__node * left`  
Pointer to the left child of the current node.
- `struct rbtree__node * right`  
Pointer to the right child of the current node.
- `struct rbtree__node * parent`  
Pointer to the parent of the current node.

### 4.13.1 Detailed Description

Node of the Red-Black Tree.

### 4.13.2 Field Documentation

#### 4.13.2.1 color

[Color](#) rbtree\_\_node::color

Color of the current node.

#### 4.13.2.2 key

unsigned int rbtree\_\_node::key

Key of the Red-Black Tree node.

#### 4.13.2.3 left

struct [rbtree\\_\\_node\\*](#) rbtree\_\_node::left

Pointer to the left child of the current node.

#### 4.13.2.4 parent

struct [rbtree\\_\\_node\\*](#) rbtree\_\_node::parent

Pointer to the parent of the current node.

#### 4.13.2.5 right

struct [rbtree\\_\\_node\\*](#) rbtree\_\_node::right

Pointer to the right child of the current node.

#### 4.13.2.6 rr\_list

[Dns\\_RR\\_LinkList\\*](#) rbtree\_node::rr\_list

Pointer to the linked list corresponding to the current node.

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

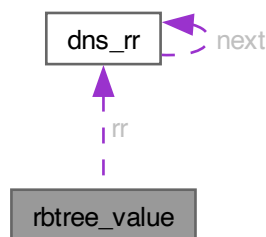
- include/[linklist\\_rbtree.h](#)

### 4.14 rbtree\_value Struct Reference

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

```
#include <linklist_rbtree.h>
```

Collaboration diagram for rbtree\_value:



#### Data Fields

- [Dns\\_RR](#) \* rr  
Pointer to a linked list of Dns\_RR.
- uint16\_t [ancount](#)  
Number of RRs in the Answer Section.
- uint16\_t [nscount](#)  
Number of RRs in the Authority Section.
- uint16\_t [arcount](#)  
Number of RRs in the Additional Section.
- uint8\_t [type](#)  
Type of the Question corresponding to the RR.

### 4.14.1 Detailed Description

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

### 4.14.2 Field Documentation

#### 4.14.2.1 `ancount`

```
uint16_t rbtree_value::ancount
```

Number of RRs in the Answer Section.

#### 4.14.2.2 `arcount`

```
uint16_t rbtree_value::arcount
```

Number of RRs in the Additional Section.

#### 4.14.2.3 `nscount`

```
uint16_t rbtree_value::nscount
```

Number of RRs in the Authority Section.

#### 4.14.2.4 `rr`

```
Dns_RR* rbtree_value::rr
```

Pointer to a linked list of Dns\_RR.

#### 4.14.2.5 `type`

```
uint8_t rbtree_value::type
```

Type of the Question corresponding to the RR.

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

- `include/linklist_rbtree.h`



## Chapter 5

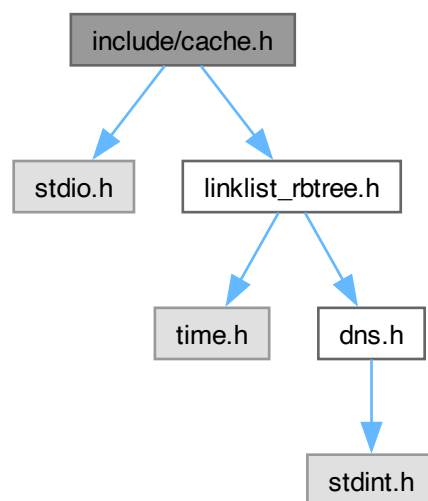
# File Documentation

### 5.1 include/cache.h File Reference

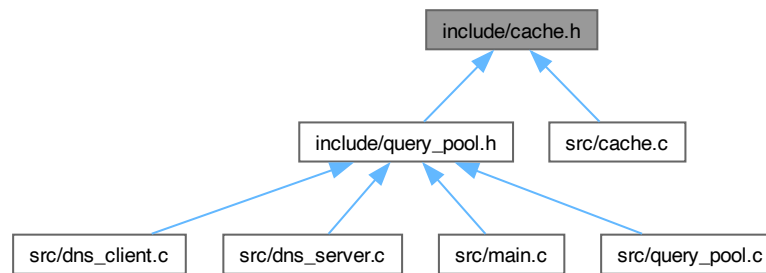
```
#include <stdio.h>
```

```
#include "linklist_rbtrees.h"
```

Include dependency graph for cache.h:



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



## Data Structures

- struct [cache\\_](#)  
Cash struct.

## Macros

- `#define` [CACHE\\_SIZE](#) 30

## Typedefs

- typedef struct [cache\\_](#) [Cache](#)  
Cash struct.

## Functions

- [Cache](#) \* [new\\_cache](#) (FILE \*hosts\_file)  
Create a new cache and initialize it with data from the hosts file.

## 5.1.1 Macro Definition Documentation

### 5.1.1.1 [CACHE\\_SIZE](#)

```
#define CACHE_SIZE 30
```



## 5.1.2 Typedef Documentation

### 5.1.2.1 Cache

typedef struct [cache\\_\\_](#) Cache

Cash struct.

## 5.1.3 Function Documentation

### 5.1.3.1 new\_cache()

```
Cache * new_cache (
    FILE * hosts_file)
```

Create a new cache and initialize it with data from the hosts file.

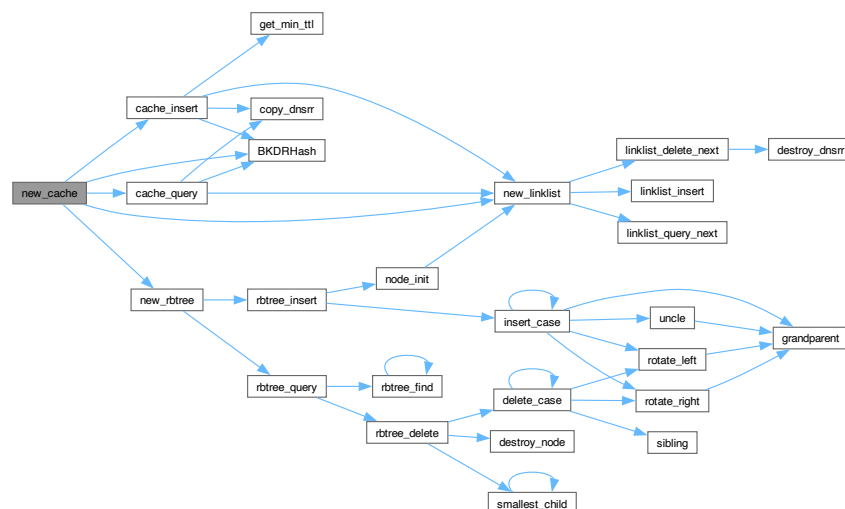
Parameters

hosts_file	The file containing hosts data.
------------	---------------------------------

Returns

The newly created cache.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.2 cache.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DNSR_CACHE_H
00002 #define DNSR_CACHE_H
00003
00004 #include <stdio.h>
00005
00006 #include "linklist_rbtrees.h"
00007
00008 #define CACHE_SIZE 30
00009
00011 typedef struct cache_ {
00012     Dns_RR_LinkList * head;
00013     Dns_RR_LinkList * tail;
00014     int size;
00015     Rbtrees * tree;
00016
00022     void (* insert)(struct cache_ * cache, const Dns_Msg * msg);
00023
00030     Rbtrees_Value * (* query)(struct cache_ * cache, const Dns_Queue * que);
00031 } Cache;
00032
00038 Cache * new_cache(FILE * hosts_file);
00039
00040 #endif //DNSR_CACHE_H
  
```

## 5.3 include/config.h File Reference

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



### Functions

- `void init_config(int argc, char *const *argv)`  
Parse command line arguments.

## Variables

- char \* [REMOTE\\_HOST](#)  
Remote DNS server address.
- int [LOG\\_MASK](#)  
Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.
- int [CLIENT\\_PORT](#)  
Local DNS client port.
- char \* [HOSTS\\_PATH](#)  
Hosts file path.
- char \* [LOG\\_PATH](#)  
Log file path.

## 5.3.1 Function Documentation

### 5.3.1.1 init\_config()

```
void init_config (  
    int argc,  
    char *const * argv)
```

Parse command line arguments.

#### Parameters

argc	Number of arguments
argv	Array of argument strings

Here is the caller graph for this function:



## 5.3.2 Variable Documentation

### 5.3.2.1 CLIENT\_PORT

```
int CLIENT_PORT [extern]
```

Local DNS client port.

### 5.3.2.2 HOSTS\_PATH

char\* HOSTS\_PATH [extern]

Hosts file path.

### 5.3.2.3 LOG\_MASK

int LOG\_MASK [extern]

Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.

### 5.3.2.4 LOG\_PATH

char\* LOG\_PATH [extern]

Log file path.

### 5.3.2.5 REMOTE\_HOST

char\* REMOTE\_HOST [extern]

Remote DNS server address.

## 5.4 config.h

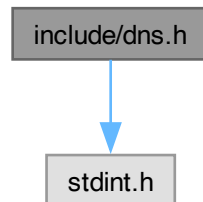
[Go to the documentation of this file.](#)

```
00001 #ifndef DNSR_CONFIG_H
00002 #define DNSR_CONFIG_H
00003
00004 extern char * REMOTE_HOST;
00005 extern int LOG_MASK;
00006 extern int CLIENT_PORT;
00007 extern char * HOSTS_PATH;
00008 extern char * LOG_PATH;
00009
00015 void init_config(int argc, char * const * argv);
00016
00017 #endif //DNSR_CONFIG_H
```

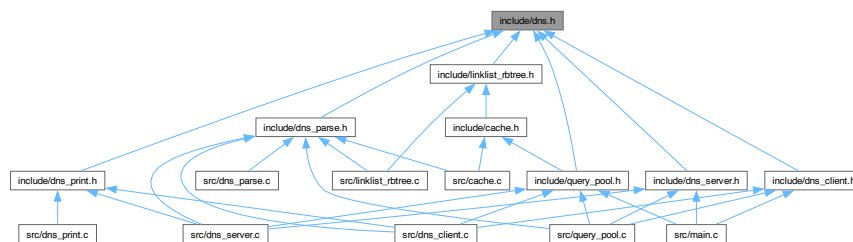
## 5.5 include/dns.h File Reference

```
#include <stdint.h>
```

Include dependency graph for dns.h:



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



### Data Structures

- struct [dns\\_header](#)  
Header Section structure of DNS message.
- struct [dns\\_question](#)  
Question Section structure of DNS message, represented as a linked list.
- struct [dns\\_rr](#)  
Resource Record structure of DNS message, represented as a linked list.
- struct [dns\\_msg](#)  
DNS message structure.

## Macros

- `#define DNS_STRING_MAX_SIZE 8192`
- `#define DNS_RR_NAME_MAX_SIZE 512`
- `#define DNS_QR_QUERY 0`
- `#define DNS_QR_ANSWER 1`
- `#define DNS_OPCODE_QUERY 0`
- `#define DNS_OPCODE_IQUERY 1`
- `#define DNS_OPCODE_STATUS 2`
- `#define DNS_TYPE_A 1`
- `#define DNS_TYPE_NS 2`
- `#define DNS_TYPE_CNAME 5`
- `#define DNS_TYPE_SOA 6`
- `#define DNS_TYPE_PTR 12`
- `#define DNS_TYPE_HINFO 13`
- `#define DNS_TYPE_MINFO 15`
- `#define DNS_TYPE_MX 15`
- `#define DNS_TYPE_TXT 16`
- `#define DNS_TYPE_AAAA 28`
- `#define DNS_CLASS_IN 1`
- `#define DNS_RCODE_OK 0`
- `#define DNS_RCODE_NXDOMAIN 3`
- `#define DNS_RCODE_SERVFAIL 2`

## Typedefs

- typedef struct `dns_header` `Dns_Header`  
Header Section structure of DNS message.
- typedef struct `dns_question` `Dns_Que`  
Question Section structure of DNS message, represented as a linked list.
- typedef struct `dns_rr` `Dns_RR`  
Resource Record structure of DNS message, represented as a linked list.
- typedef struct `dns_msg` `Dns_Msg`  
DNS message structure.

### 5.5.1 Macro Definition Documentation

#### 5.5.1.1 DNS\_CLASS\_IN

```
#define DNS_CLASS_IN 1
```

#### 5.5.1.2 DNS\_OPCODE\_IQUERY

```
#define DNS_OPCODE_IQUERY 1
```

#### 5.5.1.3 DNS\_OPCODE\_QUERY

```
#define DNS_OPCODE_QUERY 0
```

#### 5.5.1.4 DNS\_OPCODE\_STATUS

```
#define DNS_OPCODE_STATUS 2
```

#### 5.5.1.5 DNS\_QR\_ANSWER

```
#define DNS_QR_ANSWER 1
```

#### 5.5.1.6 DNS\_QR\_QUERY

```
#define DNS_QR_QUERY 0
```

#### 5.5.1.7 DNS\_RCODE\_NXDOMAIN

```
#define DNS_RCODE_NXDOMAIN 3
```

#### 5.5.1.8 DNS\_RCODE\_OK

```
#define DNS_RCODE_OK 0
```

#### 5.5.1.9 DNS\_RCODE\_SERVFAIL

```
#define DNS_RCODE_SERVFAIL 2
```

#### 5.5.1.10 DNS\_RR\_NAME\_MAX\_SIZE

```
#define DNS_RR_NAME_MAX_SIZE 512
```

#### 5.5.1.11 DNS\_STRING\_MAX\_SIZE

```
#define DNS_STRING_MAX_SIZE 8192
```

#### 5.5.1.12 DNS\_TYPE\_A

```
#define DNS_TYPE_A 1
```

#### 5.5.1.13 DNS\_TYPE\_AAAA

```
#define DNS_TYPE_AAAA 28
```

#### 5.5.1.14 DNS\_TYPE\_CNAME

```
#define DNS_TYPE_CNAME 5
```

#### 5.5.1.15 DNS\_TYPE\_HINFO

```
#define DNS_TYPE_HINFO 13
```

#### 5.5.1.16 DNS\_TYPE\_MINFO

```
#define DNS_TYPE_MINFO 15
```

#### 5.5.1.17 DNS\_TYPE\_MX

```
#define DNS_TYPE_MX 15
```

#### 5.5.1.18 DNS\_TYPE\_NS

```
#define DNS_TYPE_NS 2
```

#### 5.5.1.19 DNS\_TYPE\_PTR

```
#define DNS_TYPE_PTR 12
```



#### 5.5.1.20 DNS\_\_TYPE\_\_SOA

```
#define DNS__TYPE__SOA 6
```

#### 5.5.1.21 DNS\_\_TYPE\_\_TXT

```
#define DNS__TYPE__TXT 16
```

### 5.5.2 Typedef Documentation

#### 5.5.2.1 Dns\_\_Header

```
typedef struct dns\_header Dns__Header
```

Header Section structure of DNS message.

#### 5.5.2.2 Dns\_\_Msg

```
typedef struct dns\_msg Dns__Msg
```

DNS message structure.

#### 5.5.2.3 Dns\_\_Que

```
typedef struct dns\_question Dns__Que
```

Question Section structure of DNS message, represented as a linked list.

#### 5.5.2.4 Dns\_\_RR

```
typedef struct dns\_rr Dns__RR
```

Resource Record structure of DNS message, represented as a linked list.

## 5.6 dns.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DNSR_DNS_H
00002 #define DNSR_DNS_H
00003
00004 #include <stdint.h>
00005
00006 #define DNS_STRING_MAX_SIZE 8192
00007 #define DNS_RR_NAME_MAX_SIZE 512
00008
00009 #define DNS_QR_QUERY 0
00010 #define DNS_QR_ANSWER 1
00011
00012 #define DNS_OPCODE_QUERY 0
00013 #define DNS_OPCODE_IQUERY 1
00014 #define DNS_OPCODE_STATUS 2
00015
00016 #define DNS_TYPE_A 1
00017 #define DNS_TYPE_NS 2
00018 #define DNS_TYPE_CNAME 5
00019 #define DNS_TYPE_SOA 6
00020 #define DNS_TYPE_PTR 12
00021 #define DNS_TYPE_HINFO 13
00022 #define DNS_TYPE_MINFO 15
00023 #define DNS_TYPE_MX 15
00024 #define DNS_TYPE_TXT 16
00025 #define DNS_TYPE_AAAA 28
00026
00027 #define DNS_CLASS_IN 1
00028
00029 #define DNS_RCODE_OK 0
00030 #define DNS_RCODE_NXDOMAIN 3
00031 #define DNS_RCODE_SERVFAIL 2
00032
00034 typedef struct dns_header {
00035     uint16_t id;
00036     uint8_t qr: 1;
00037     uint8_t opcode: 4;
00038     uint8_t aa: 1;
00039     uint8_t tc: 1;
00040     uint8_t rd: 1;
00041     uint8_t ra: 1;
00042     uint8_t z: 3;
00043     uint8_t rcode: 4;
00044     uint16_t qdcount;
00045     uint16_t ancount;
00046     uint16_t nscount;
00047     uint16_t arcount;
00048 } Dns_Header;
00049
00051 typedef struct dns_question {
00052     uint8_t * qname;
00053     uint16_t qtype;
00054     uint16_t qclass;
00055     struct dns_question * next;
00056 } Dns_Que;
00057
00059 typedef struct dns_rr {
00060     uint8_t * name;
00061     uint16_t type;
00062     uint16_t class;
00063     uint32_t ttl;
00064     uint16_t rdlength;
00065     uint8_t * rdata;
00066     struct dns_rr * next;
00067 } Dns_RR;

```

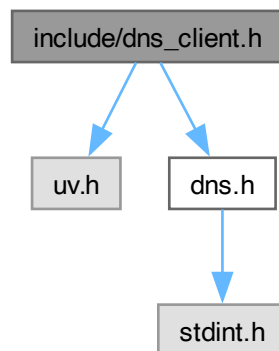
```
00068
00070 typedef struct dns_msg {
00071     Dns_Header * header;
00072     Dns_Queue * que;
00073     Dns_RR * rr;
00074 } Dns_Msg;
00075
00076 #endif //DNSR_DNS_H
```

## 5.7 include/dns\_client.h File Reference

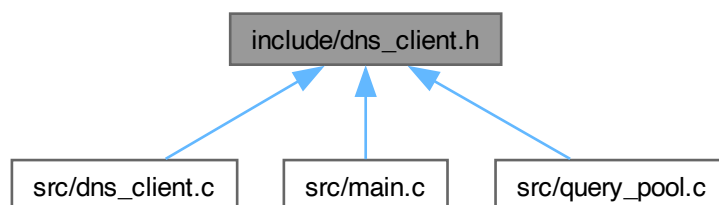
```
#include <uv.h>
```

```
#include "dns.h"
```

Include dependency graph for dns\_client.h:



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



## Functions

- void `init_client` (`uv_loop_t *loop`)  
Initialize the DNS client.
- void `send_to_remote` (`const Dns_Msg *msg`)  
Send a DNS query message to the remote server.

### 5.7.1 Function Documentation

#### 5.7.1.1 `init_client()`

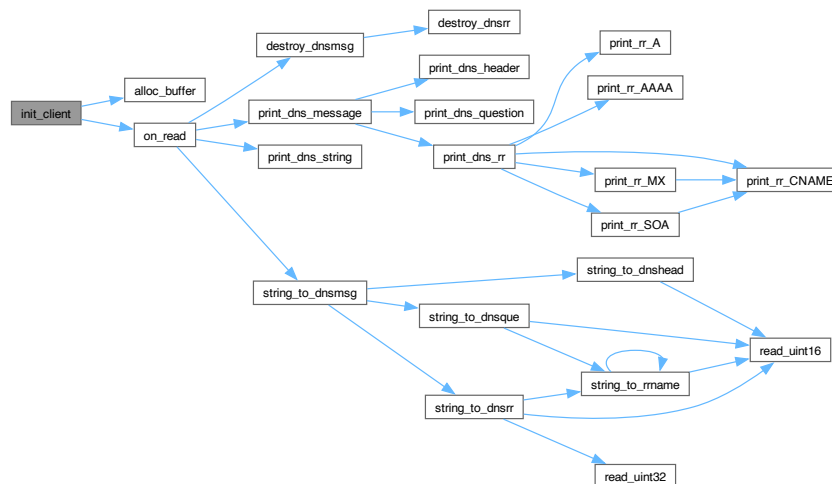
```
void init_client (
    uv_loop_t * loop)
```

Initialize the DNS client.

Parameters

loop	The libuv event loop
------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.7.1.2 send\_to\_remote()

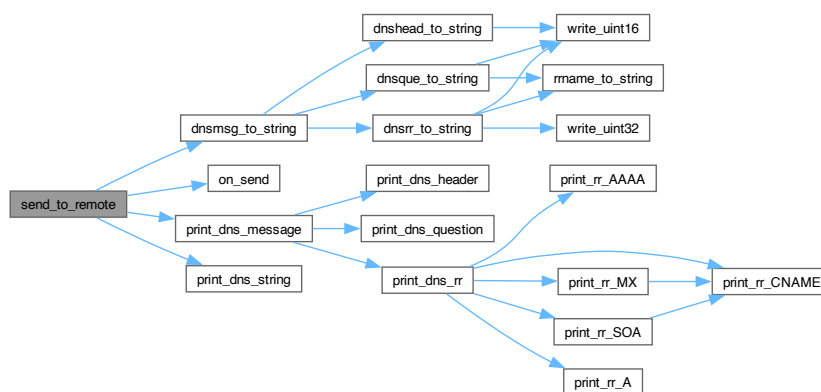
```
void send_to_remote (
    const Dns_Msg * msg)
```

Send a DNS query message to the remote server.

Parameters

msg	The DNS message to be sent
-----	----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.8 dns\_client.h

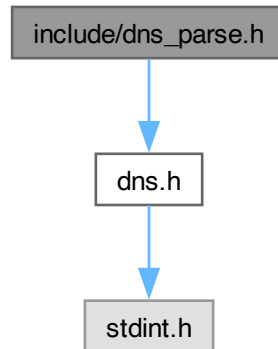
[Go to the documentation of this file.](#)

```
00001 #ifndef DNSR_DNS_CLIENT_H
00002 #define DNSR_DNS_CLIENT_H
00003
00004 #include <uv.h>
00005
00006 #include "dns.h"
00007
00012 void init_client(uv_loop_t * loop);
00013
00018 void send_to_remote(const Dns_Msg * msg);
00019
00020 #endif //DNSR_DNS_CLIENT_H
```

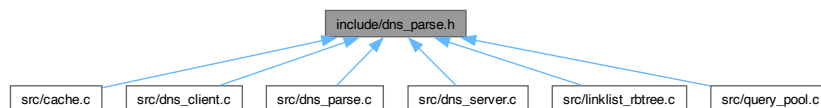
## 5.9 include/dns\_parse.h File Reference

#include "dns.h"

Include dependency graph for dns\_parse.h:



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



### Functions

- void [string\\_to\\_dnsmmsg](#) ([Dns\\_Msg](#) \*pmsg, const char \*pstring)  
Convert a byte stream to a DNS message structure.
- unsigned [dnsmmsg\\_to\\_string](#) (const [Dns\\_Msg](#) \*pmsg, char \*pstring)  
Write a NAME field to a byte stream.
- void [destroy\\_dnsrr](#) ([Dns\\_RR](#) \*prr)  
Release memory allocated for a Resource Record.
- void [destroy\\_dnsmmsg](#) ([Dns\\_Msg](#) \*pmsg)  
Release memory allocated for a DNS message.
- [Dns\\_RR](#) \* [copy\\_dnsrr](#) (const [Dns\\_RR](#) \*src)  
Copy a Resource Record.
- [Dns\\_Msg](#) \* [copy\\_dnsmmsg](#) (const [Dns\\_Msg](#) \*src)  
Copy a DNS message.

## 5.9.1 Function Documentation

### 5.9.1.1 copy\_dnsmsg()

```
Dns_Msg * copy_dnsmsg (  
    const Dns_Msg * src)
```

Copy a DNS message.

Parameters

src	The DNS message to copy
-----	-------------------------

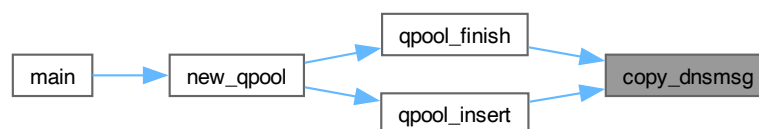
Returns

A copy of the DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.1.2 copy\_dnsrr()

```
Dns_RR * copy_dnsrr (  
    const Dns_RR * src)
```

Copy a Resource Record.

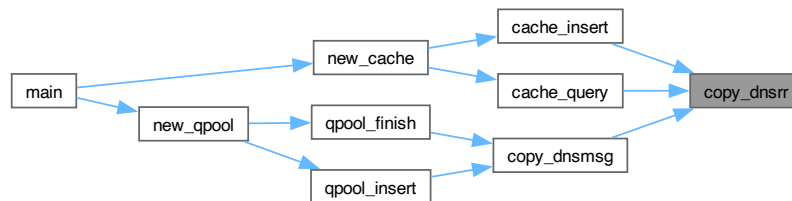
## Parameters

src	The Resource Record to copy
-----	-----------------------------

## Returns

A copy of the Resource Record

Here is the caller graph for this function:

5.9.1.3 `destroy_dnsmmsg()`

```
void destroy_dnsmmsg (
    Dns_Msg * pmsg)
```

Release memory allocated for a DNS message.

## Parameters

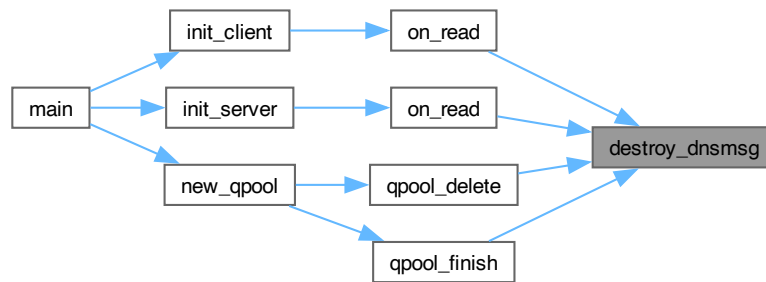
pmsg	The DNS message to release
------	----------------------------

Here is the call graph for this function:





Here is the caller graph for this function:



#### 5.9.1.4 destroy\_dnsrr()

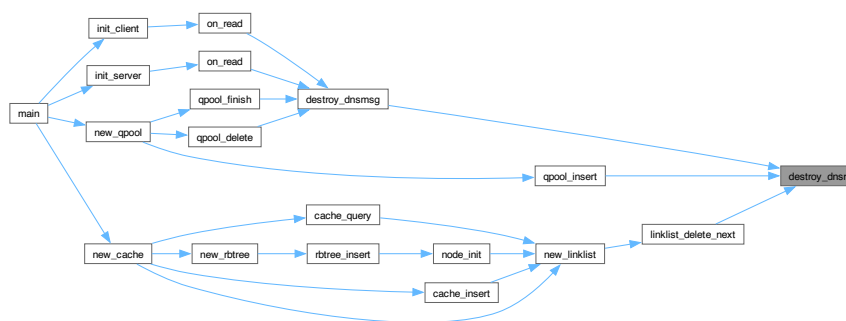
```
void destroy_dnsrr (
    Dns_RR * prr)
```

Release memory allocated for a Resource Record.

Parameters

prr	The Resource Record to release
-----	--------------------------------

Here is the caller graph for this function:



#### 5.9.1.5 dnsmmsg\_to\_string()

```
unsigned dnsmmsg_to_string (
    const Dns_Msg * pmsg,
    char * pstring)
```

Write a NAME field to a byte stream.

## Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

## Note

After writing, the offset increases to the position after the NAME field

Write a NAME field to a byte stream.

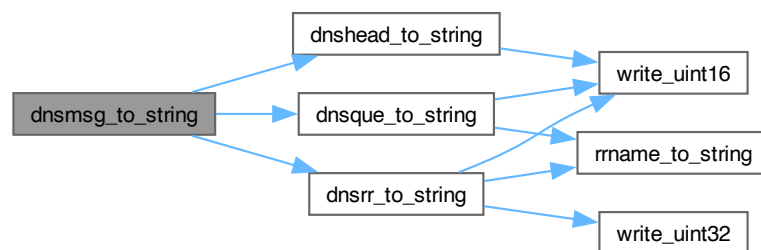
## Parameters

pmsg	The DNS message structure to convert
pstring	The byte stream to write to

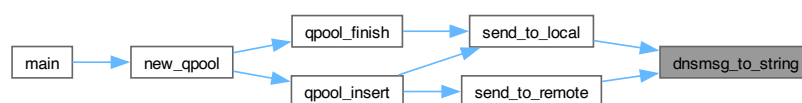
## Returns

The total length of the byte stream

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.9.1.6 string\_to\_dnsmsg()

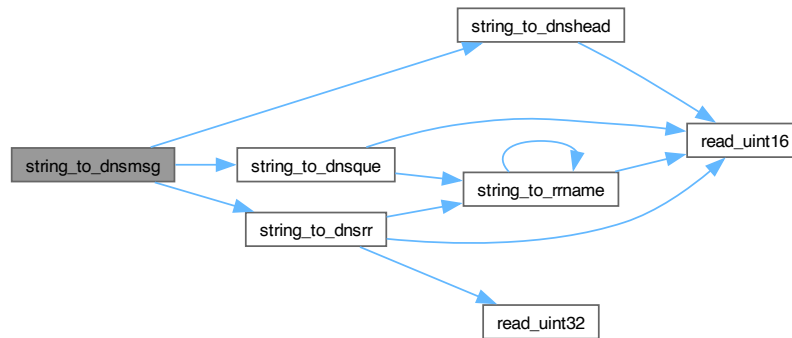
```
void string_to_dnsmsg (
    Dns_Msg * pmsg,
    const char * pstring)
```

Convert a byte stream to a DNS message structure.

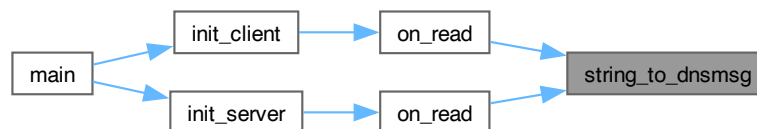
Parameters

pmsg	The DNS message structure to populate
pstring	The byte stream to read from

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.10 dns\_parse.h

[Go to the documentation of this file.](#)

```
00001 #ifndef DNSR_DNS_PARSE_H
```

```

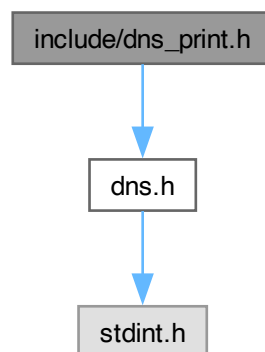
00002 #define DNSR_DNS_PARSE_H
00003
00004 #include "dns.h"
00005
00011 void string_to_dnsmsg(Dns_Msg * pmsg, const char * pstring);
00012
00020 unsigned dnsmsg_to_string(const Dns_Msg * pmsg, char * pstring);
00021
00026 void destroy_dnsrr(Dns_RR * prr);
00027
00032 void destroy_dnsmsg(Dns_Msg * pmsg);
00033
00039 Dns_RR * copy_dnsrr(const Dns_RR * src);
00040
00046 Dns_Msg * copy_dnsmsg(const Dns_Msg * src);
00047
00048 #endif //DNSR_DNS_PARSE_H

```

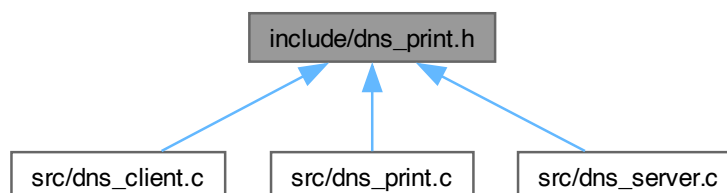
## 5.11 include/dns\_print.h File Reference

#include "dns.h"

Include dependency graph for dns\_print.h:



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



## Functions

- void [print\\_dns\\_string](#) (const char \*pstring, unsigned int len)  
Print DNS message byte stream.
- void [print\\_dns\\_message](#) (const [Dns\\_Msg](#) \*pmsg)  
Print the entire DNS message.

## 5.11.1 Function Documentation

5.11.1.1 [print\\_dns\\_message\(\)](#)

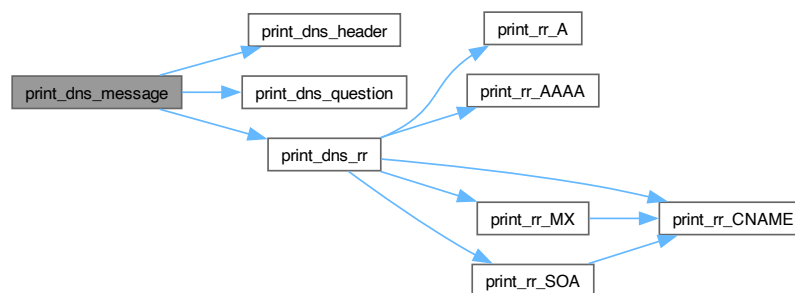
```
void print_dns_message (
    const Dns\_Msg * pmsg)
```

Print the entire DNS message.

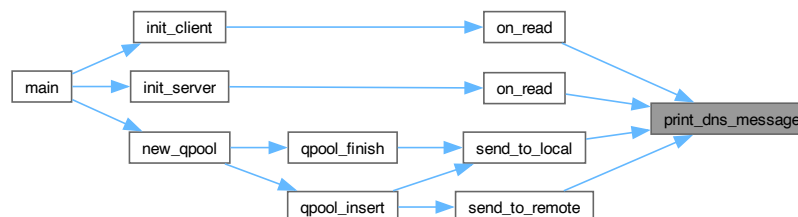
## Parameters

pmsg	The DNS message
------	-----------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.11.1.2 print\_dns\_string()

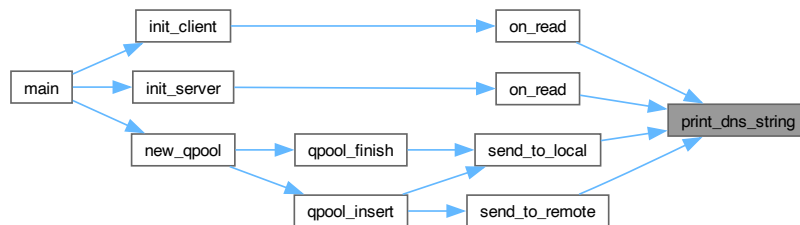
```
void print_dns_string (
    const char * pstring,
    unsigned int len)
```

Print DNS message byte stream.

Parameters

pstring	The byte stream
len	The length of the byte stream

Here is the caller graph for this function:



## 5.12 dns\_print.h

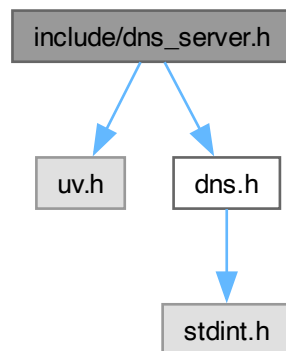
[Go to the documentation of this file.](#)

```
00001 #ifndef DNSR_DNS_PRINT_H
00002 #define DNSR_DNS_PRINT_H
00003
00004 #include "dns.h"
00005
00011 void print_dns_string(const char * pstring, unsigned int len);
00012
00017 void print_dns_message(const Dns_Msg * pmsg);
00018
00019 #endif //DNSR_DNS_PRINT_H
```

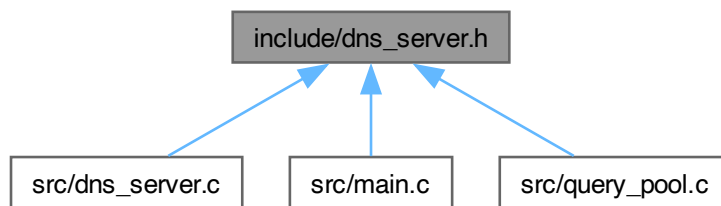
## 5.13 include/dns\_server.h File Reference

```
#include <uv.h>
#include "dns.h"
```

Include dependency graph for dns\_server.h:



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



## Functions

- void `init_server` (`uv_loop_t *loop`)  
Initialize the DNS server.
- void `send_to_local` (`const struct sockaddr *addr`, `const Dns_Msg *msg`)  
Send a DNS response message to local clients.

### 5.13.1 Function Documentation

#### 5.13.1.1 `init_server()`

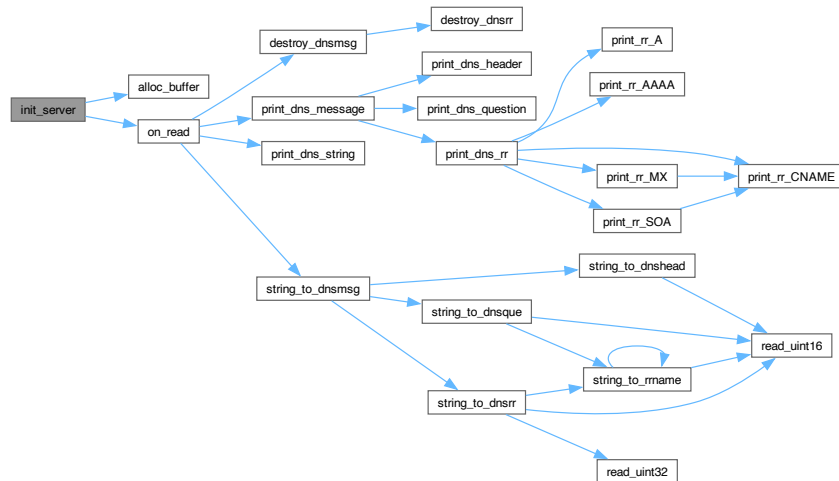
```
void init_server (  
    uv_loop_t * loop)
```

Initialize the DNS server.

## Parameters

loop	The libuv event loop
------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.13.1.2 send\_to\_local()

```

void send_to_local (
    const struct sockaddr * addr,
    const Dns_Msg * msg)

```

Send a DNS response message to local clients.

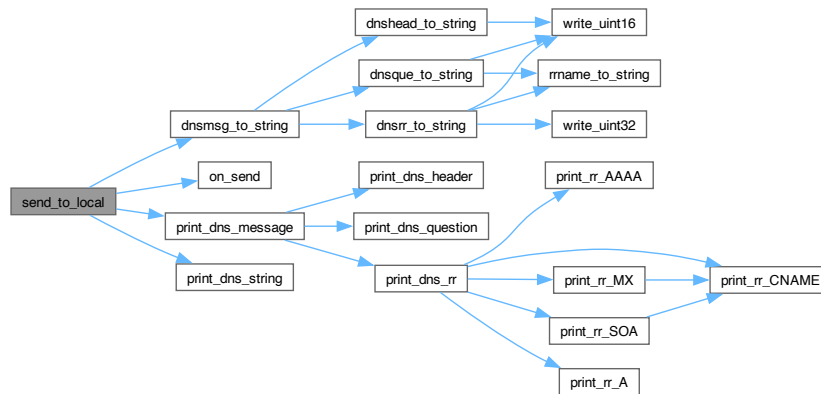
## Parameters

addr	The address of the local client
------	---------------------------------

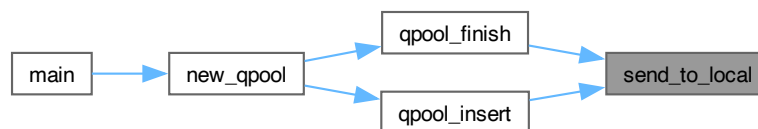


msg	The DNS message to be sent
-----	----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.14 dns\_server.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DNSR_DNS_SERVER_H
00002 #define DNSR_DNS_SERVER_H
00003
00004 #include <uv.h>
00005
00006 #include "dns.h"
00007
00012 void init_server(uv_loop_t * loop);
00013
00019 void send_to_local(const struct sockaddr * addr, const Dns_Msg * msg);
00020
00021 #endif //DNSR_DNS_SERVER_H

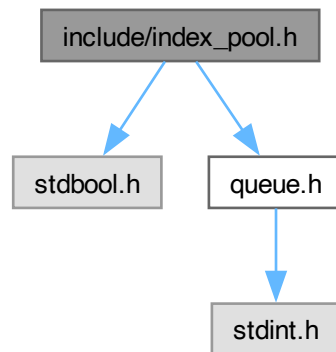
```

## 5.15 include/index\_pool.h File Reference

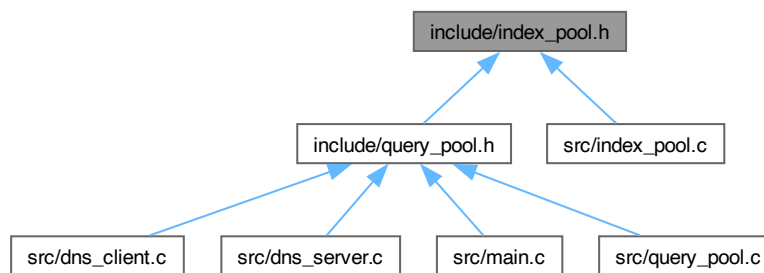
```
#include <stdbool.h>
```

```
#include "queue.h"
```

Include dependency graph for index\_pool.h:



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



### Data Structures

- struct [index\\_](#)  
Index structure.
- struct [index\\_pool](#)  
Index pool.

## Macros

- `#define INDEX_POOL_MAX_SIZE 65535`

## Typedefs

- typedef struct [index\\_\\_ Index](#)  
Index structure.
- typedef struct [index\\_pool Index\\_Pool](#)  
Index pool.

## Functions

- [Index\\_Pool \\* new\\_ipool \(\)](#)  
Create a new index pool.

### 5.15.1 Macro Definition Documentation

#### 5.15.1.1 INDEX\_POOL\_MAX\_SIZE

```
#define INDEX_POOL_MAX_SIZE 65535
```

### 5.15.2 Typedef Documentation

#### 5.15.2.1 Index

```
typedef struct index\_\_ Index
```

Index structure.

#### 5.15.2.2 Index\_Pool

```
typedef struct index\_pool Index\_Pool
```

Index pool.

### 5.15.3 Function Documentation

#### 5.15.3.1 new\_ipool()

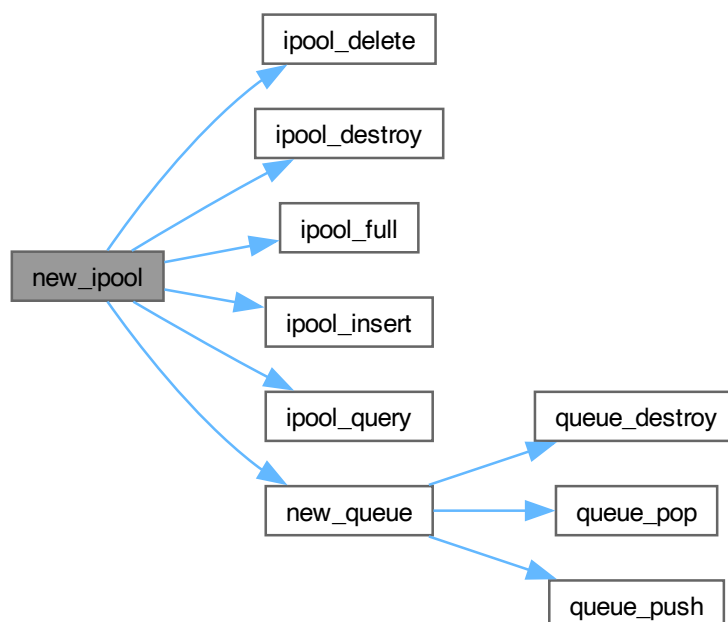
[Index\\_Pool](#) \* new\_ipool ()

Create a new index pool.

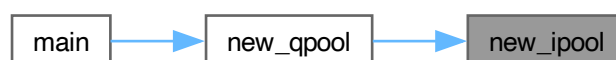
Returns

The new index pool

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.16 index\_pool.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DNSR_INDEX_POOL_H
00002 #define DNSR_INDEX_POOL_H
00003
00004 #include <stdbool.h>
00005
00006 #include "queue.h"
00007
00008 #define INDEX_POOL_MAX_SIZE 65535
00009
00011 typedef struct index_
00012 {
00013     uint16_t id;
00014     uint16_t prev_id;
00015 } Index;
00016
00018 typedef struct index_pool
00019 {
00020     Index * pool[INDEX_POOL_MAX_SIZE];
00021     unsigned short count;
00022     Queue * queue;
00023
00029     bool (* full)(struct index_pool * ipool);
00030
00037     uint16_t (* insert)(struct index_pool * ipool, Index * req);
00038
00045     bool (* query)(struct index_pool * ipool, uint16_t index);
00046
00053     Index * (* delete)(struct index_pool * ipool, uint16_t index);
00054
00059     void (* destroy)(struct index_pool * ipool);
00060 } Index_Pool;
00061
00066 Index_Pool * new_ipool();
00067
00068 #endif //DNSR_INDEX_POOL_H

```

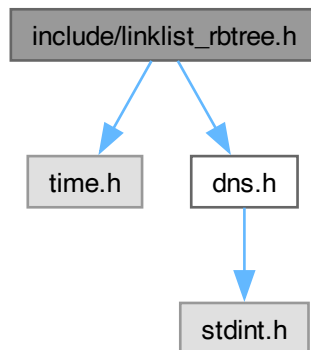
## 5.17 include/linklist\_rbtrees.h File Reference

```

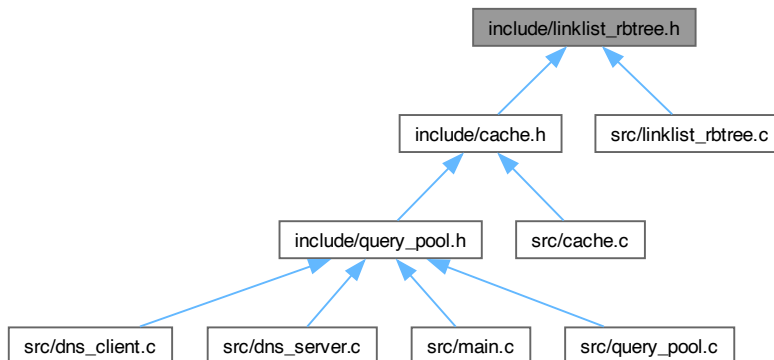
#include <time.h>
#include "dns.h"

```

Include dependency graph for linklist\_rbtrees.h:



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



## Data Structures

- struct `rbtree_value`  
Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.
- struct `dns_rr_linklist`  
Linked list of Red-Black Tree nodes.
- struct `rbtree_node`  
Node of the Red-Black Tree.
- struct `linklist_rbtrees`  
Red-Black Tree.

## Typedefs

- typedef struct [rbtree\\_value](#) [Rbtree\\_Value](#)  
Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.
- typedef struct [dns\\_rr\\_linklist](#) [Dns\\_RR\\_LinkList](#)  
Linked list of Red-Black Tree nodes.
- typedef struct [rbtree\\_node](#) [Rbtree\\_Node](#)  
Node of the Red-Black Tree.
- typedef struct [linklist\\_rbtree](#) [Rbtree](#)  
Red-Black Tree.

## Enumerations

- enum [Color](#) { [BLACK](#) , [RED](#) }  
Red-Black Tree color.

## Functions

- [Dns\\_RR\\_LinkList](#) \* [new\\_linklist](#) ()  
Create a new linked list.
- [Rbtree](#) \* [new\\_rbtree](#) ()  
Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

### 5.17.1 Typedef Documentation

#### 5.17.1.1 [Dns\\_RR\\_LinkList](#)

typedef struct [dns\\_rr\\_linklist](#) [Dns\\_RR\\_LinkList](#)

Linked list of Red-Black Tree nodes.

#### 5.17.1.2 [Rbtree](#)

typedef struct [linklist\\_rbtree](#) [Rbtree](#)

Red-Black Tree.

### 5.17.1.3 Rbtree\_\_Node

```
typedef struct rbtree\_\_node Rbtree__Node
```

Node of the Red-Black Tree.

### 5.17.1.4 Rbtree\_\_Value

```
typedef struct rbtree\_\_value Rbtree__Value
```

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

## 5.17.2 Enumeration Type Documentation

### 5.17.2.1 Color

```
enum Color
```

Red-Black Tree color.

Enumerator

BLACK	
RED	

## 5.17.3 Function Documentation

### 5.17.3.1 new\_\_linklist()

```
Dns\_\_RR\_\_LinkList * new__linklist ()
```

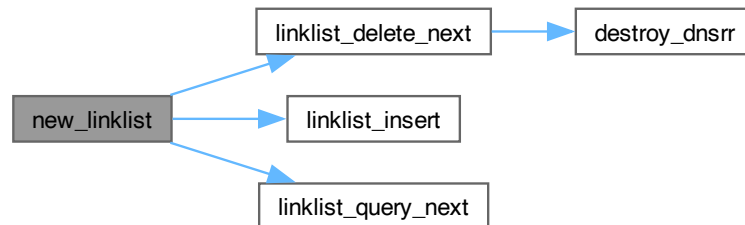
Create a new linked list.



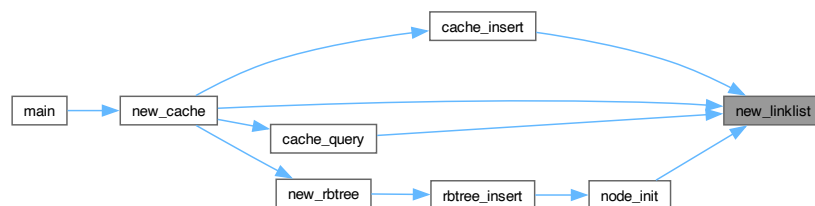
## Returns

The new linked list

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.17.3.2 new\_rbtrees()

```
Rbtrees * new_rbtrees ()
```

Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

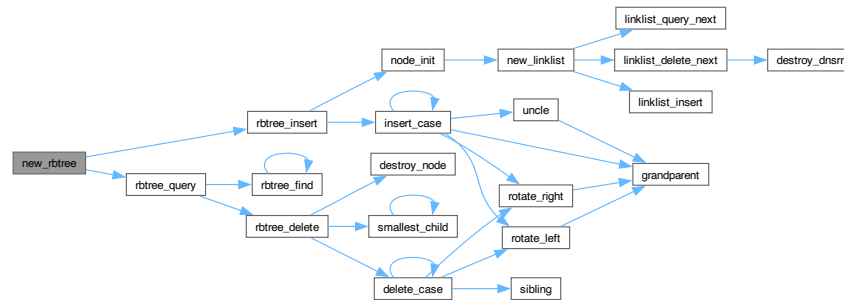
## Returns

A pointer to the newly created red-black tree

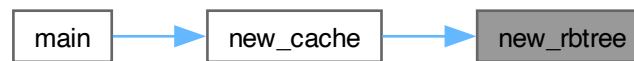
Note

If memory allocation fails, the function will log a fatal error and terminate the program.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.18 linklist\_rbtrees.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DNSR_LINKLIST_RBTREE_H
00002 #define DNSR_LINKLIST_RBTREE_H
00003
00004 #include <time.h>
00005
00006 #include "dns.h"
00007
00009 typedef enum {
00010     BLACK, RED
00011 } Color;
00012
00014 typedef struct rbtrees_value {
00015     Dns_RR *rr;
00016     uint16_t ancount;
00017     uint16_t nscount;
00018     uint16_t arcount;
00019     uint8_t type;
00020 } Rbtrees_Value;
00021
00023 typedef struct dns_rr_linklist {

```

```

00024  Rbtree_Value *value;
00025  time_t expire_time;
00026  struct dns_rr_linklist *next;
00027
00034  void (*insert)(struct dns_rr_linklist *list, struct dns_rr_linklist *new_list_node);
00035
00040  void (*delete_next)(struct dns_rr_linklist *list);
00041
00049  struct dns_rr_linklist *(*query_next)(struct dns_rr_linklist *list, const uint8_t *qname, const uint16_t qtype);
00050 } Dns_RR_LinkList;
00051
00053 typedef struct rbtree_node {
00054     unsigned int key;
00055     Dns_RR_LinkList *rr_list;
00056     Color color;
00057     struct rbtree_node *left;
00058     struct rbtree_node *right;
00059     struct rbtree_node *parent;
00060 } Rbtree_Node;
00061
00063 typedef struct linklist_rbtree {
00064     Rbtree_Node *root;
00065
00072     void (*insert)(struct linklist_rbtree *tree, unsigned int key, Dns_RR_LinkList *list);
00073
00080     Dns_RR_LinkList *(*query)(struct linklist_rbtree *tree, unsigned int data);
00081 } Rbtree;
00082
00087 Dns_RR_LinkList *new_linklist();
00088
00096 Rbtree *new_rbtree();
00097
00098 #endif //DNSR_LINKLIST_RBTREE_H

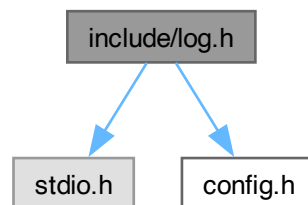
```

## 5.19 include/log.h File Reference

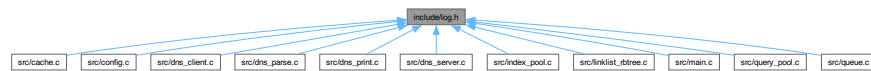
#include <stdio.h>

#include "config.h"

Include dependency graph for log.h:



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



## Macros

- `#define log_debug(args...)`
- `#define log_info(args...)`
- `#define log_error(args...)`
- `#define log_fatal(args...)`

## Variables

- `FILE * log_file`

## 5.19.1 Macro Definition Documentation

### 5.19.1.1 log\_debug

```
#define log_debug(
    args...)
```

Value:

```
if (LOG_MASK & 1) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[DEBUG] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[37m[DEBUG]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

### 5.19.1.2 log\_error

```
#define log_error(
    args...)
```

Value:

```
if (LOG_MASK & 4) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[ERROR] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[33m[ERROR]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

### 5.19.1.3 log\_fatal

```
#define log_fatal(  
    args...)
```

Value:

```
if (LOG_MASK & 8) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[FATAL] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[31m[FATAL]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
    exit(EXIT_FAILURE); \
}
```

### 5.19.1.4 log\_info

```
#define log_info(  
    args...)
```

Value:

```
if (LOG_MASK & 2) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[INFO ] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[34m[INFO ]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

## 5.19.2 Variable Documentation

### 5.19.2.1 log\_file

```
FILE* log_file [extern]
```

## 5.20 log.h

[Go to the documentation of this file.](#)

```
00001 #ifndef DNSR_LOG_H
00002 #define DNSR_LOG_H
00003
00004 #include <stdio.h>
00005
00006 #include "config.h"
00007
00008 extern FILE * log_file;
00009
```

```

00010 #define log_debug(args...) \
00011     if (LOG_MASK & 1) \
00012     { \
00013         if (log_file != stderr) \
00014             fprintf(log_file, "[DEBUG] %s:%d ", __FILE__, __LINE__); \
00015         else \
00016             fprintf(log_file, "\x1b[37m[DEBUG]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
00017         fprintf(log_file, args); \
00018         fprintf(log_file, "\n"); \
00019     }
00020
00021 #define log_info(args...) \
00022     if (LOG_MASK & 2) \
00023     { \
00024         if (log_file != stderr) \
00025             fprintf(log_file, "[INFO ] %s:%d ", __FILE__, __LINE__); \
00026         else \
00027             fprintf(log_file, "\x1b[34m[INFO ]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
00028         fprintf(log_file, args); \
00029         fprintf(log_file, "\n"); \
00030     }
00031
00032 #define log_error(args...) \
00033     if (LOG_MASK & 4) \
00034     { \
00035         if (log_file != stderr) \
00036             fprintf(log_file, "[ERROR] %s:%d ", __FILE__, __LINE__); \
00037         else \
00038             fprintf(log_file, "\x1b[33m[ERROR]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
00039         fprintf(log_file, args); \
00040         fprintf(log_file, "\n"); \
00041     }
00042
00043 #define log_fatal(args...) \
00044     if (LOG_MASK & 8) \
00045     { \
00046         if (log_file != stderr) \
00047             fprintf(log_file, "[FATAL] %s:%d ", __FILE__, __LINE__); \
00048         else \
00049             fprintf(log_file, "\x1b[31m[FATAL]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
00050         fprintf(log_file, args); \
00051         fprintf(log_file, "\n"); \
00052         exit(EXIT_FAILURE); \
00053     }
00054
00055 #endif //DNSR_LOG_H

```

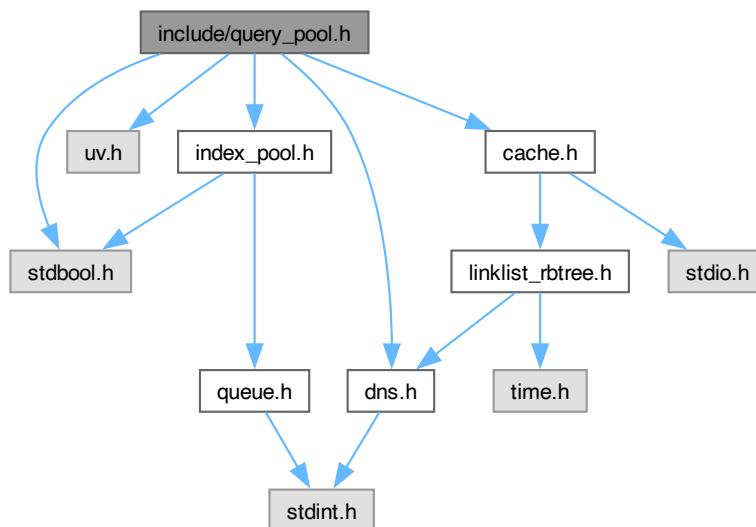
## 5.21 include/query\_pool.h File Reference

```

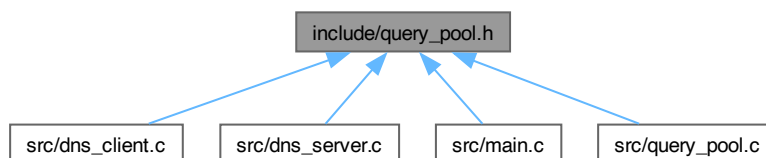
#include <stdbool.h>
#include <uv.h>
#include "dns.h"
#include "index_pool.h"
#include "cache.h"

```

Include dependency graph for query\_pool.h:



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



## Data Structures

- struct `dns_query`  
DNS query structure.
- struct `query_pool`  
DNS query pool.

## Macros

- `#define QUERY_POOL_MAX_SIZE 256`

## Typedefs

- typedef struct [dns\\_query](#) [Dns\\_Query](#)  
DNS query structure.
- typedef struct [query\\_pool](#) [Query\\_Pool](#)  
DNS query pool.

## Functions

- [Query\\_Pool](#) \* [new\\_qpool](#) ([uv\\_loop\\_t](#) \*[loop](#), [Cache](#) \*[cache](#))  
Create a new query pool This function initializes a new query pool and returns a pointer to it.

## 5.21.1 Macro Definition Documentation

### 5.21.1.1 QUERY\_POOL\_MAX\_SIZE

```
#define QUERY_POOL_MAX_SIZE 256
```

## 5.21.2 Typedef Documentation

### 5.21.2.1 Dns\_Query

```
typedef struct dns\_query Dns\_Query
```

DNS query structure.

### 5.21.2.2 Query\_Pool

```
typedef struct query\_pool Query\_Pool
```

DNS query pool.

## 5.21.3 Function Documentation

### 5.21.3.1 new\_qpool()

```
Query\_Pool * new\_qpool (  
    uv\_loop\_t * loop,  
    Cache * cache)
```

Create a new query pool This function initializes a new query pool and returns a pointer to it.



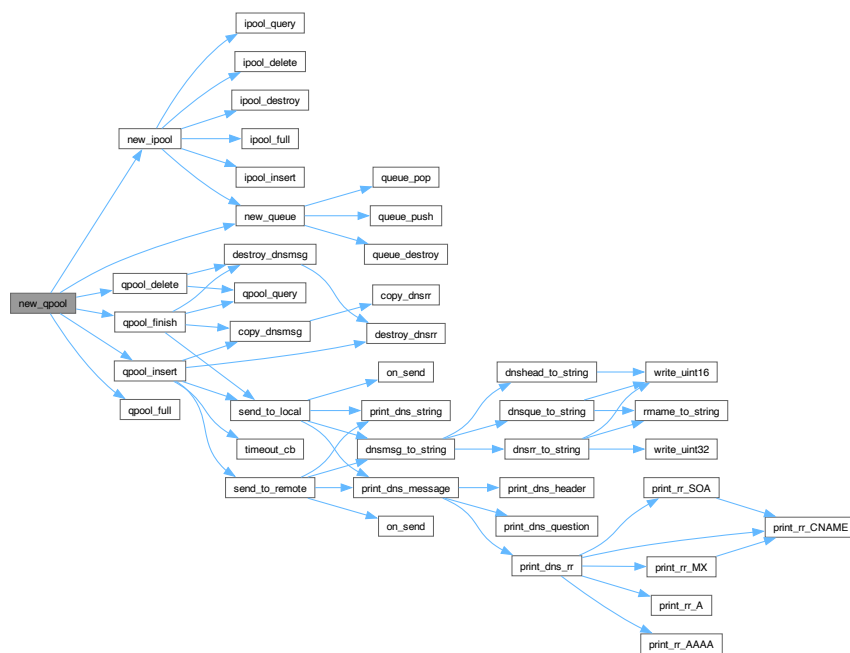
## Parameters

loop	The libuv event loop
cache	The cache used for storing DNS responses

## Returns

A pointer to the newly created query pool

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.22 query\_pool.h

[Go to the documentation of this file.](#)

```

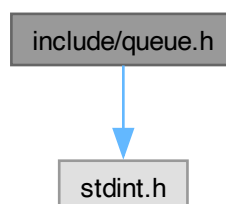
00001 #ifndef DNSR_QUERY_POOL_H
00002 #define DNSR_QUERY_POOL_H
00003
00004 #include <stdbool.h>
00005 #include <uv.h>
00006
00007 #include "dns.h"
00008 #include "index_pool.h"
00009 #include "cache.h"
00010
00011 #define QUERY_POOL_MAX_SIZE 256
00012
00014 typedef struct dns_query {
00015     uint16_t id;
00016     uint16_t prev_id;
00017     struct sockaddr addr;
00018     Dns_Msg * msg;
00019     uv_timer_t timer;
00020 } Dns_Query;
00021
00023 typedef struct query_pool {
00024     Dns_Query * pool[QUERY_POOL_MAX_SIZE];
00025     unsigned short count;
00026     Queue * queue;
00027     Index_Pool * ipool;
00028     uv_loop_t * loop;
00029     Cache * cache;
00030
00036     bool (* full)(struct query_pool * qpool);
00037
00047     void (* insert)(struct query_pool * qpool, const struct sockaddr * addr, const Dns_Msg * msg);
00048
00056     void (* finish)(struct query_pool * qpool, const Dns_Msg * msg);
00057
00064     void (* delete)(struct query_pool * qpool, uint16_t id);
00065 } Query_Pool;
00066
00074 Query_Pool *new_qpool(uv_loop_t * loop, Cache * cache);
00075
00076 #endif //DNSR_QUERY_POOL_H

```

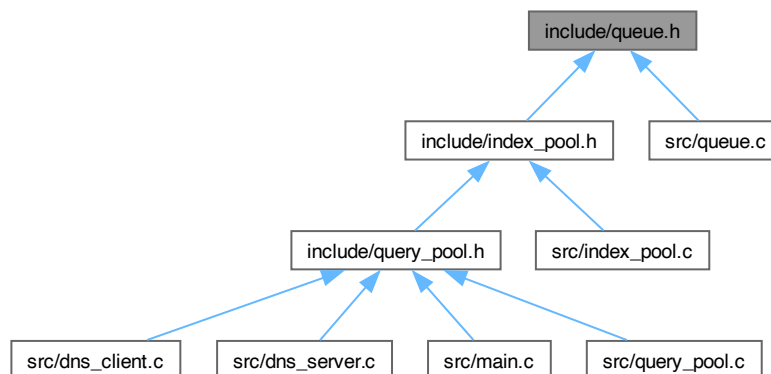
## 5.23 include/queue.h File Reference

#include <stdint.h>

Include dependency graph for queue.h:



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



## Data Structures

- struct [queue](#)  
Circular queue.

## Macros

- `#define` [QUEUE\\_MAX\\_SIZE](#) 65536

## Typedefs

- typedef struct [queue](#) [Queue](#)  
Circular queue.

## Functions

- [Queue](#) \* [new\\_queue](#) ()  
Create a new queue.

### 5.23.1 Macro Definition Documentation

#### 5.23.1.1 QUEUE\_MAX\_SIZE

```
#define QUEUE_MAX_SIZE 65536
```

## 5.23.2 Typedef Documentation

### 5.23.2.1 Queue

typedef struct [queue](#) Queue

Circular queue.

## 5.23.3 Function Documentation

### 5.23.3.1 new\_queue()

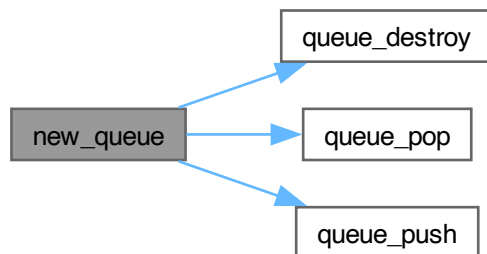
Queue \* new\_queue ()

Create a new queue.

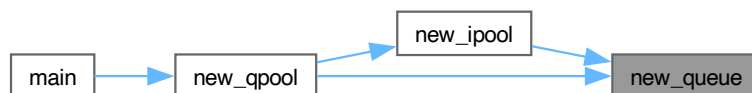
Returns

The new queue

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.24 queue.h

[Go to the documentation of this file.](#)

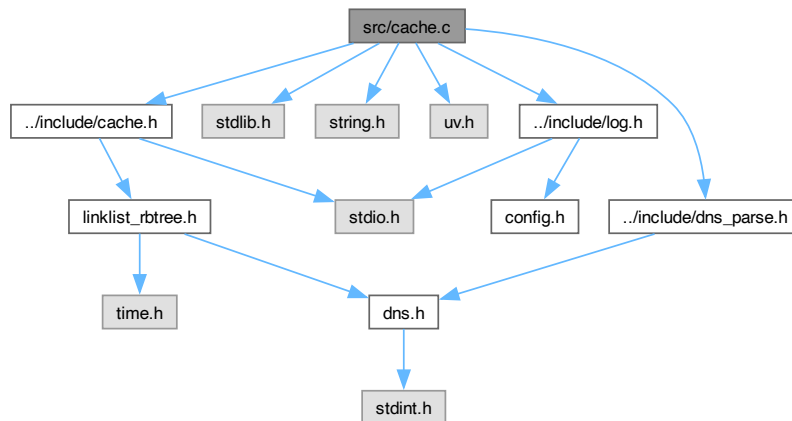
```
00001 #include <stdint.h>
00002
00003 #ifndef DNSR_QUEUE_H
00004 #define DNSR_QUEUE_H
00005
00006 #define QUEUE_MAX_SIZE 65536
00007
00009 typedef struct queue
00010 {
00011     uint16_t q[QUEUE_MAX_SIZE];
00012     unsigned short head;
00013     unsigned short tail;
00014
00020     void (* push)(struct queue * queue, uint16_t num);
00021
00027     uint16_t (* pop)(struct queue * queue);
00028
00033     void (* destroy)(struct queue * queue);
00034 } Queue;
00035
00040 Queue * new_queue();
00041
00042 #endif //DNSR_QUEUE_H
```

## 5.25 README.md File Reference

## 5.26 src/cache.c File Reference

```
#include "../include/cache.h"
#include <stdlib.h>
#include <string.h>
#include <uv.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
```

Include dependency graph for cache.c:



## Functions

- static unsigned int [BKDRHash](#) (const uint8\_t \*str)  
Compute the hash of a string using the BKDR hash algorithm.
- static uint32\_t [get\\_min\\_ttl](#) (const [Dns\\_RR](#) \*pr)   
Get the smallest TTL (Time-To-Live) value in a list of Resource Records (RRs).
- static void [cache\\_insert](#) ([Cache](#) \*cache, const [Dns\\_Msg](#) \*msg)  
Insert a DNS message into the cache.
- static [Rbtrees\\_Value](#) \* [cache\\_query](#) ([Cache](#) \*cache, const [Dns\\_Que](#) \*que)  
Query the cache for a DNS question.
- [Cache](#) \* [new\\_cache](#) (FILE \*hosts\_file)  
Create a new cache and initialize it with data from the hosts file.

## 5.26.1 Function Documentation

### 5.26.1.1 BKDRHash()

```
static unsigned int BKDRHash (
    const uint8_t * str)  [static]
```

Compute the hash of a string using the BKDR hash algorithm.

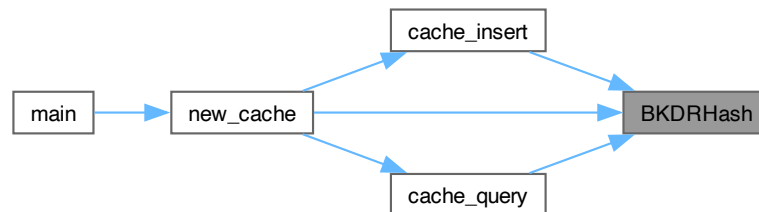
Parameters

str	The input string.
-----	-------------------

## Returns

The computed hash value.

Here is the caller graph for this function:



## 5.26.1.2 cache\_insert()

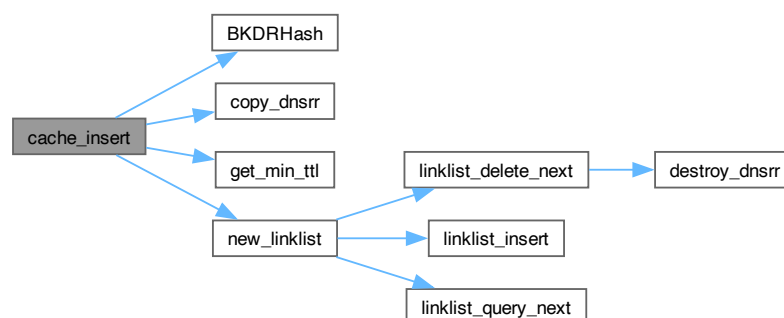
```
static void cache_insert (
    Cache * cache,
    const Dns_Msg * msg) [static]
```

Insert a DNS message into the cache.

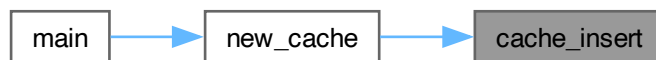
## Parameters

cache	The cache where the message will be inserted.
msg	The DNS message to be inserted.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.26.1.3 cache\_query()

```
static Rbtree_Value * cache_query (
    Cache * cache,
    const Dns_Que * que)  [static]
```

Query the cache for a DNS question.

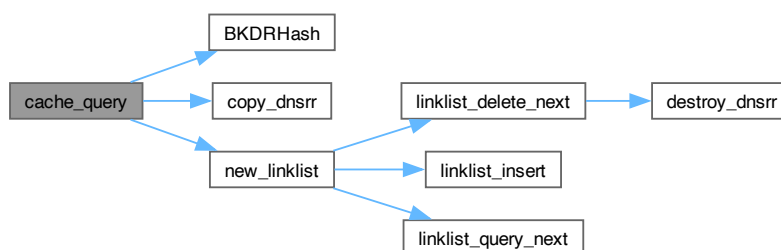
Parameters

cache	The cache to query.
que	The DNS question.

Returns

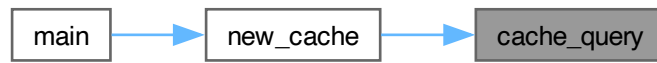
The value found in the cache or NULL if not found.

Here is the call graph for this function:





Here is the caller graph for this function:



#### 5.26.1.4 `get_min_ttl()`

```
static uint32_t get_min_ttl (  
    const Dns_RR * prr) [static]
```

Get the smallest TTL (Time-To-Live) value in a list of Resource Records (RRs).

Parameters

<code>prr</code>	The head node of the RR linked list.
------------------	--------------------------------------

Returns

The minimum TTL value.

Here is the caller graph for this function:



#### 5.26.1.5 `new_cache()`

```
Cache * new_cache (  
    FILE * hosts_file)
```

Create a new cache and initialize it with data from the hosts file.

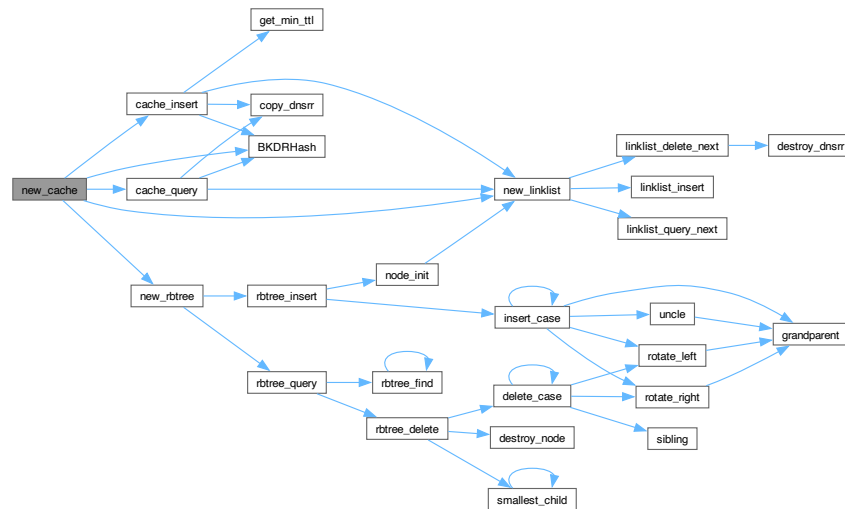
## Parameters

hosts_file	The file containing hosts data.
------------	---------------------------------

## Returns

The newly created cache.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.27 src/config.c File Reference

```

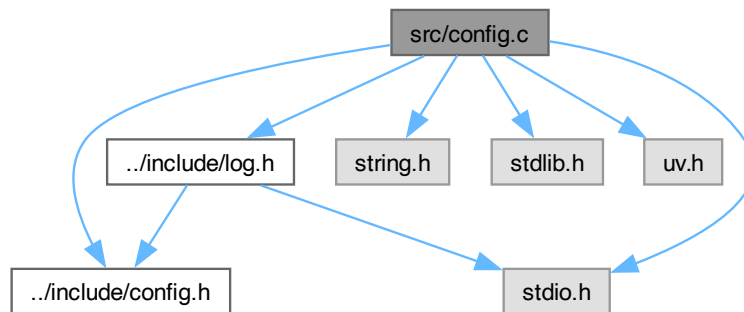
#include "../include/config.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

```

```
#include <uv.h>
```

```
#include "../include/log.h"
```

Include dependency graph for config.c:



## Functions

- void `init_config` (int argc, char \*const \*argv)  
Parse command line arguments.

## Variables

- char \* `REMOTE_HOST` = "8.8.8.8"  
Remote DNS server address.
- int `LOG_MASK` = 15  
Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.
- int `CLIENT_PORT` = 0  
Local DNS client port.
- char \* `HOSTS_PATH` = "../dnsrelay.txt"  
Hosts file path.
- char \* `LOG_PATH` = NULL  
Log file path.

### 5.27.1 Function Documentation

#### 5.27.1.1 `init_config()`

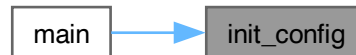
```
void init_config (
    int argc,
    char *const * argv)
```

Parse command line arguments.

## Parameters

argc	Number of arguments
argv	Array of argument strings

Here is the caller graph for this function:



## 5.27.2 Variable Documentation

### 5.27.2.1 CLIENT\_PORT

```
int CLIENT_PORT = 0
```

Local DNS client port.

### 5.27.2.2 HOSTS\_PATH

```
char* HOSTS_PATH = "../dnsrelay.txt"
```

Hosts file path.

### 5.27.2.3 LOG\_MASK

```
int LOG_MASK = 15
```

Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.

### 5.27.2.4 LOG\_PATH

```
char* LOG_PATH = NULL
```

Log file path.

## 5.27.2.5 REMOTE\_HOST

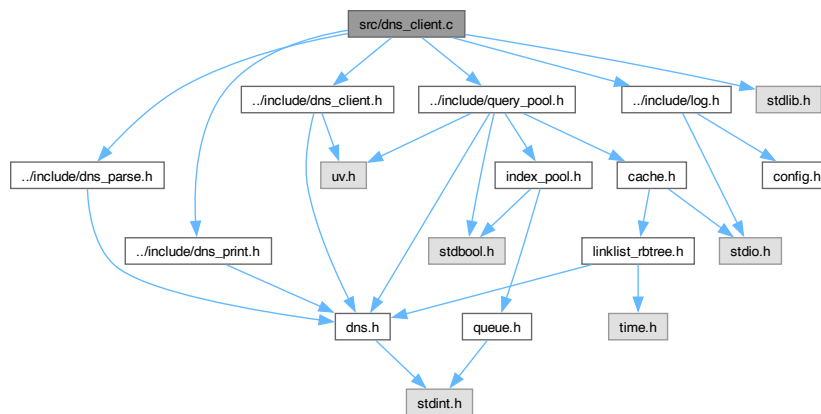
```
char* REMOTE_HOST = "8.8.8.8"
```

Remote DNS server address.

## 5.28 src/dns\_client.c File Reference

```
#include "../include/dns_client.h"
#include <stdlib.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
#include "../include/dns_print.h"
#include "../include/query_pool.h"
```

Include dependency graph for dns\_client.c:



## Functions

- static void [alloc\\_buffer](#) (uv\_handle\_t \*handle, size\_t suggested\_size, uv\_buf\_t \*buf)  
Allocate space for the buffer.
- static void [on\\_read](#) (uv\_udp\_t \*handle, ssize\_t nread, const uv\_buf\_t \*buf, const struct sockaddr \*addr, unsigned flags)  
Callback function for receiving response messages from the remote server.
- static void [on\\_send](#) (uv\_udp\_send\_t \*req, int status)  
Callback function for sending query messages to the remote server.
- void [init\\_client](#) (uv\_loop\_t \*loop)  
Initialize the DNS client.
- void [send\\_to\\_remote](#) (const [Dns\\_Msg](#) \*msg)  
Send a DNS query message to the remote server.

## Variables

- static uv\_udp\_t [client\\_socket](#)  
Socket for client communication with the remote server.
- static struct sockaddr\_in [local\\_addr](#)  
Local address.
- static struct sockaddr [send\\_addr](#)  
Remote server address.
- [Query\\_Pool](#) \* [qpool](#)  
Query pool.

## 5.28.1 Function Documentation

### 5.28.1.1 alloc\_buffer()

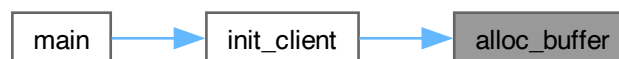
```
static void alloc_buffer (
    uv_handle_t * handle,
    size_t suggested_size,
    uv_buf_t * buf) [static]
```

Allocate space for the buffer.

#### Parameters

handle	Allocation handle
suggested_size	Suggested buffer size
buf	Buffer to be allocated

Here is the caller graph for this function:



### 5.28.1.2 init\_client()

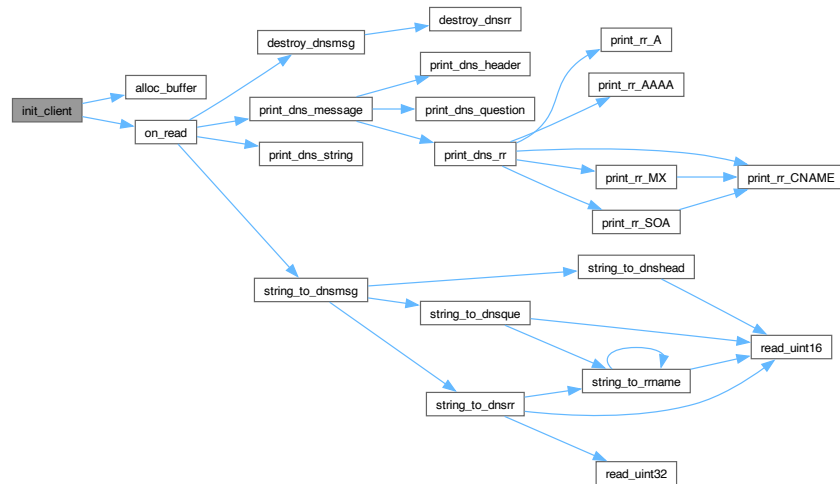
```
void init_client (
    uv_loop_t * loop)
```

Initialize the DNS client.

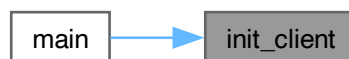
## Parameters

loop	The libuv event loop
------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.28.1.3 on\_read()

```

static void on_read (
    uv_udp_t * handle,
    ssize_t nread,
    const uv_buf_t * buf,
    const struct sockaddr * addr,
    unsigned flags) [static]

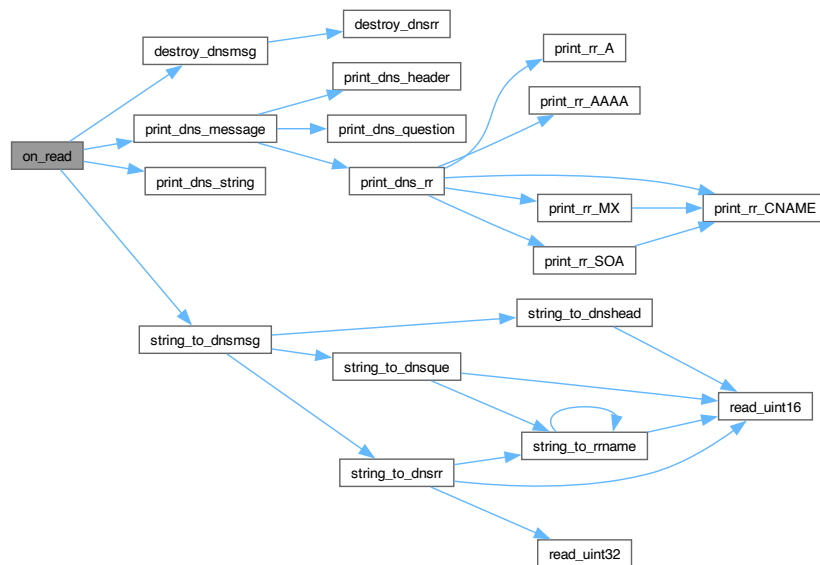
```

Callback function for receiving response messages from the remote server.

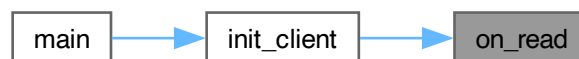
## Parameters

handle	Query handle
nread	Number of bytes received
buf	Buffer containing the received message
addr	Address of the sender
flags	Flags indicating special conditions for the received data

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.28.1.4 on\_send()

```

static void on_send (
    uv_udp_send_t * req,
    int status) [static]

```

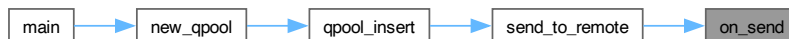


Callback function for sending query messages to the remote server.

## Parameters

req	Send handle
status	Send status, indicating whether the send was successful

Here is the caller graph for this function:

5.28.1.5 `send_to_remote()`

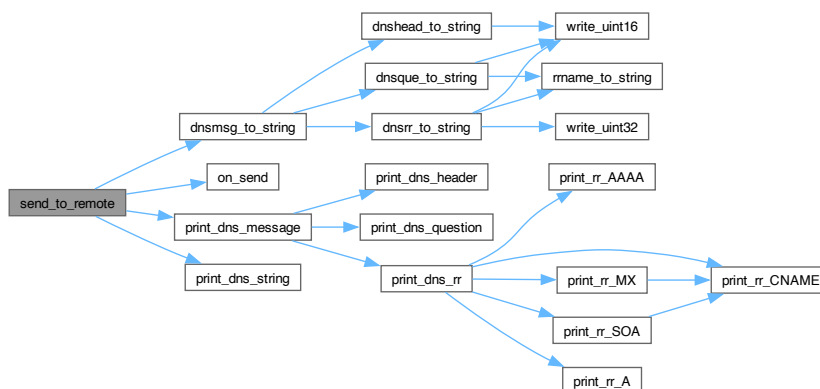
```
void send_to_remote (
    const Dns\_Msg * msg)
```

Send a DNS query message to the remote server.

## Parameters

msg	The DNS message to be sent
-----	----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.28.2 Variable Documentation

### 5.28.2.1 client\_socket

```
uv_udp_t client_socket [static]
```

Socket for client communication with the remote server.

### 5.28.2.2 local\_addr

```
struct sockaddr_in local_addr [static]
```

Local address.

### 5.28.2.3 qpool

```
Query_Pool* qpool [extern]
```

Query pool.

### 5.28.2.4 send\_addr

```
struct sockaddr send_addr [static]
```

Remote server address.

## 5.29 src/dns\_parse.c File Reference

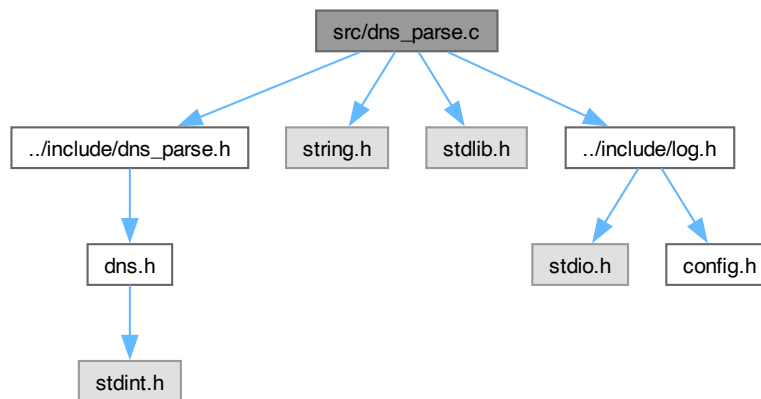
```
#include "../include/dns_parse.h"
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
#include "../include/log.h"
```

Include dependency graph for dns\_parse.c:



### Functions

- static uint16\_t [read\\_uint16](#) (const char \*pstring, unsigned \*offset)  
Read a 16-bit number in big-endian format from a byte stream.
- static uint32\_t [read\\_uint32](#) (const char \*pstring, unsigned \*offset)  
Read a 32-bit number in big-endian format from a byte stream.
- static unsigned [string\\_to\\_rrname](#) (uint8\_t \*pname, const char \*pstring, unsigned \*offset)  
Read a NAME field from a byte stream.
- static void [string\\_to\\_dnshead](#) (Dns\_Header \*phead, const char \*pstring, unsigned \*offset)  
Read a Header Section from a byte stream.
- static void [string\\_to\\_dnsque](#) (Dns\_Queue \*pque, const char \*pstring, unsigned \*offset)  
Read a Question Section from a byte stream.
- static void [string\\_to\\_dnsrr](#) (Dns\_RR \*prr, const char \*pstring, unsigned \*offset)  
Read a Resource Record from a byte stream.
- void [string\\_to\\_dnsmsg](#) (Dns\_Msg \*pmsg, const char \*pstring)  
Convert a byte stream to a DNS message structure.
- static void [write\\_uint16](#) (const char \*pstring, unsigned \*offset, uint16\_t num)  
Write a 16-bit number in little-endian format to a byte stream.

- static void [write\\_uint32](#) (const char \*pstring, unsigned \*offset, uint32\_t num)  
Write a 32-bit number in little-endian format to a byte stream.
- static void [rrname\\_to\\_string](#) (const uint8\_t \*pname, char \*pstring, unsigned \*offset)  
Write a NAME field to a byte stream.
- static void [dnshead\\_to\\_string](#) (const [Dns\\_Header](#) \*phead, char \*pstring, unsigned \*offset)  
Write a Header Section to a byte stream.
- static void [dnsque\\_to\\_string](#) (const [Dns\\_Queue](#) \*pque, char \*pstring, unsigned \*offset)  
Write a Question Section to a byte stream.
- static void [dnsrr\\_to\\_string](#) (const [Dns\\_RR](#) \*prr, char \*pstring, unsigned \*offset)  
Write a Resource Record to a byte stream.
- unsigned [dnsmsg\\_to\\_string](#) (const [Dns\\_Msg](#) \*pmsg, char \*pstring)  
Convert a DNS message structure to a byte stream.
- void [destroy\\_dnsrr](#) ([Dns\\_RR](#) \*prr)  
Release memory allocated for a Resource Record.
- void [destroy\\_dnsmsg](#) ([Dns\\_Msg](#) \*pmsg)  
Release memory allocated for a DNS message.
- [Dns\\_RR](#) \* [copy\\_dnsrr](#) (const [Dns\\_RR](#) \*src)  
Copy a Resource Record.
- [Dns\\_Msg](#) \* [copy\\_dnsmsg](#) (const [Dns\\_Msg](#) \*src)  
Copy a DNS message.

## 5.29.1 Function Documentation

### 5.29.1.1 [copy\\_dnsmsg\(\)](#)

```
Dns\_Msg * copy\_dnsmsg (  
    const Dns\_Msg * src)
```

Copy a DNS message.

Parameters

src	The DNS message to copy
-----	-------------------------

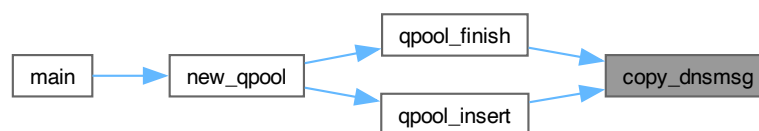
## Returns

A copy of the DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.29.1.2 copy\_dnsrr()

```
Dns_RR * copy_dnsrr (  
    const Dns_RR * src)
```

Copy a Resource Record.

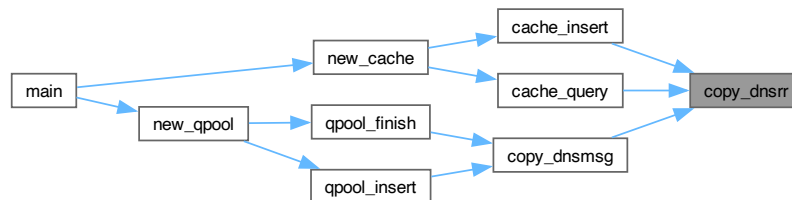
## Parameters

src	The Resource Record to copy
-----	-----------------------------

Returns

A copy of the Resource Record

Here is the caller graph for this function:



### 5.29.1.3 `destroy_dnsmsg()`

```
void destroy_dnsmsg (
    Dns_Msg * pmsg)
```

Release memory allocated for a DNS message.

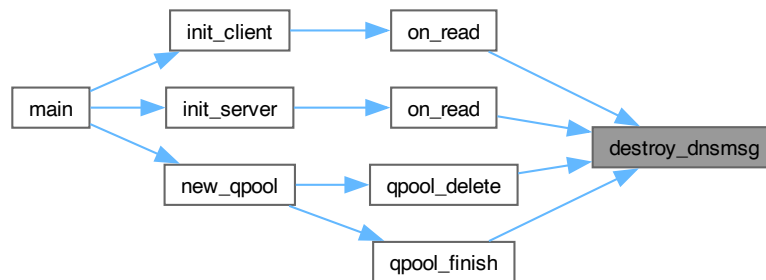
Parameters

pmsg	The DNS message to release
------	----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.29.1.4 destroy\_dnsrr()

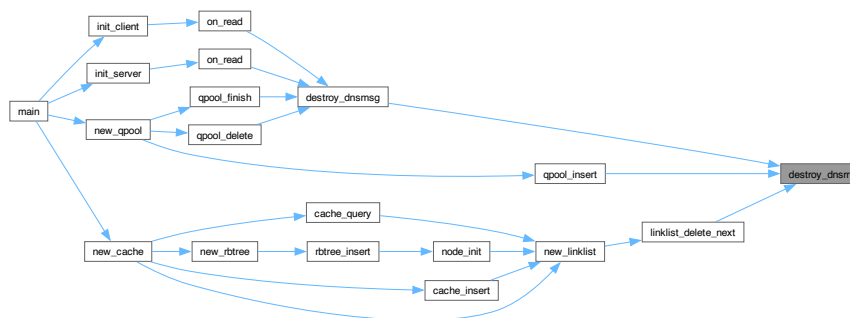
```
void destroy_dnsrr (
    Dns_RR * prr)
```

Release memory allocated for a Resource Record.

Parameters

prr	The Resource Record to release
-----	--------------------------------

Here is the caller graph for this function:





## 5.29.1.5 dnshead\_to\_string()

```
static void dnshead_to_string (
    const Dns_Header * phead,
    char * pstring,
    unsigned * offset)  [static]
```

Write a Header Section to a byte stream.

Parameters

phead	The Header Section
pstring	The start of the byte stream
offset	The offset in the byte stream

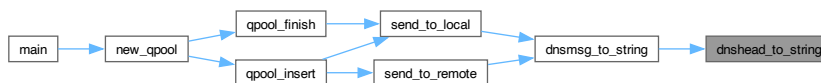
Note

After writing, the offset increases to the position after the Header Section

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.6 dnsmmsg\_to\_string()

```
unsigned dnsmmsg_to_string (
    const Dns_Msg * pmsg,
    char * pstring)
```

Convert a DNS message structure to a byte stream.

Write a NAME field to a byte stream.

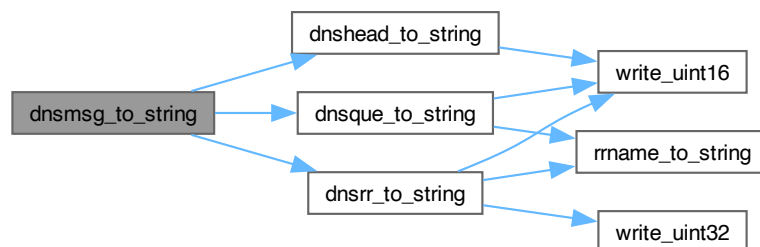
## Parameters

pmsg	The DNS message structure to convert
pstring	The byte stream to write to

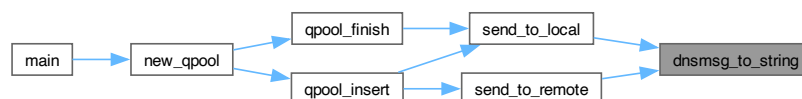
## Returns

The total length of the byte stream

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.7 dnsque\_to\_string()

```

static void dnsque_to_string (
    const Dns_Queue * pque,
    char * pstring,
    unsigned * offset) [static]

```

Write a Question Section to a byte stream.

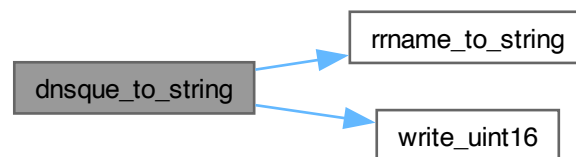
## Parameters

pque	The Question Section
pstring	The start of the byte stream
offset	The offset in the byte stream

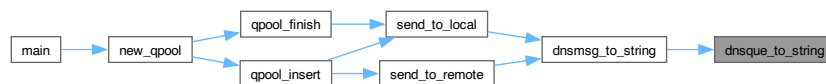
## Note

After writing, the offset increases to the position after the Question Section

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.8 dnsrr\_to\_string()

```

static void dnsrr_to_string (
    const Dns_RR * prr,
    char * pstring,
    unsigned * offset) [static]
  
```

Write a Resource Record to a byte stream.

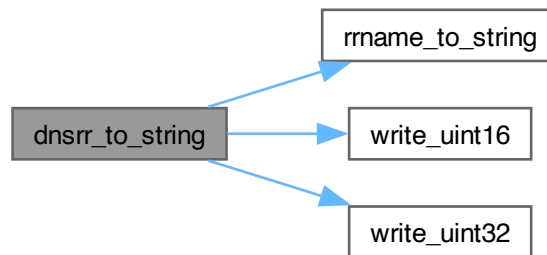
## Parameters

prr	The Resource Record
pstring	The start of the byte stream
offset	The offset in the byte stream

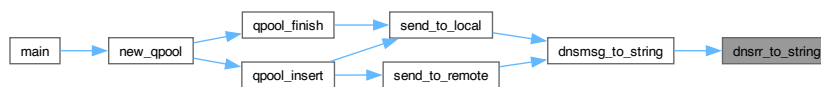
Note

After writing, the offset increases to the position after the Resource Record

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.29.1.9 read\_uint16()

```
static uint16_t read_uint16 (
    const char * pstring,
    unsigned * offset)  [static]
```

Read a 16-bit number in big-endian format from a byte stream.

Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream

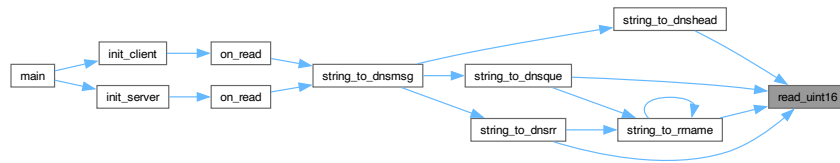
Returns

The 16-bit number starting from (pstring + \*offset)

Note

After reading, the offset increases by 2

Here is the caller graph for this function:



#### 5.29.1.10 read\_uint32()

```
static uint32_t read_uint32 (
    const char * pstring,
    unsigned * offset)  [static]
```

Read a 32-bit number in big-endian format from a byte stream.

Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream

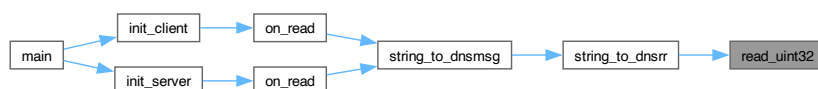
Returns

The 32-bit number starting from (pstring + \*offset)

Note

After reading, the offset increases by 4

Here is the caller graph for this function:



### 5.29.1.11 rname\_to\_string()

```
static void rname_to_string (
    const uint8_t * pname,
    char * pstring,
    unsigned * offset)  [static]
```

Write a NAME field to a byte stream.

Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

Note

After writing, the offset increases to the position after the NAME field

Here is the caller graph for this function:



### 5.29.1.12 string\_to\_dnshead()

```
static void string_to_dnshead (
    Dns\_Header * phead,
    const char * pstring,
    unsigned * offset)  [static]
```

Read a Header Section from a byte stream.

Parameters

phead	The Header Section
pstring	The start of the byte stream
offset	The offset in the byte stream

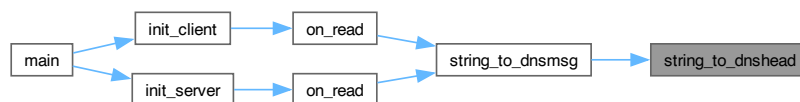
## Note

After reading, the offset increases to the position after the Header Section

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.13 string\_to\_dnsmmsg()

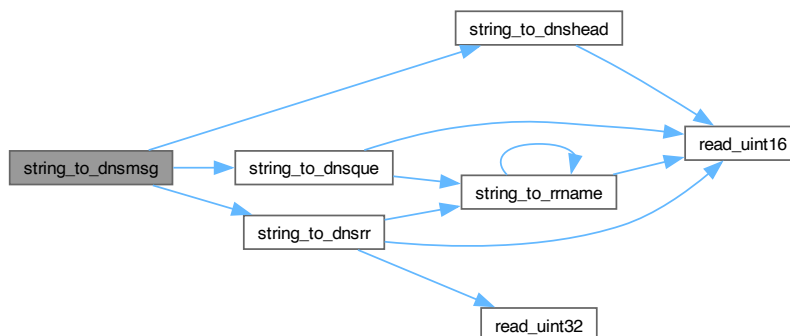
```
void string_to_dnsmmsg (
    Dns\_Msg * pmsg,
    const char * pstring)
```

Convert a byte stream to a DNS message structure.

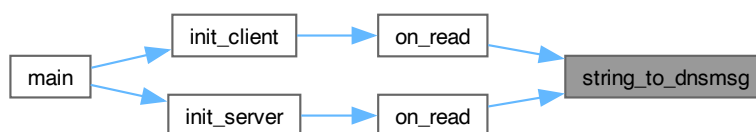
## Parameters

pmsg	The DNS message structure to populate
pstring	The byte stream to read from

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.29.1.14 string\_to\_\_dnsque()

```
static void string_to__dnsque (
    Dns_Queue * pque,
    const char * pstring,
    unsigned * offset) [static]
```

Read a Question Section from a byte stream.

Parameters

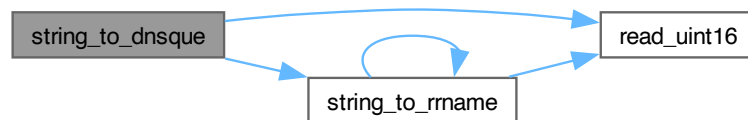
pque	The Question Section
pstring	The start of the byte stream
offset	The offset in the byte stream



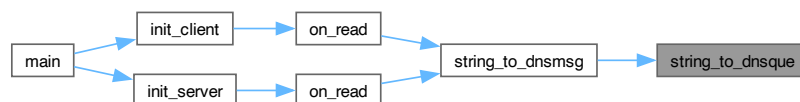
## Note

After reading, the offset increases to the position after the Question Section; space is allocated for the NAME field

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.15 string\_to\_dnsrr()

```

static void string_to_dnsrr (
    Dns_RR * prr,
    const char * pstring,
    unsigned * offset) [static]
  
```

Read a Resource Record from a byte stream.

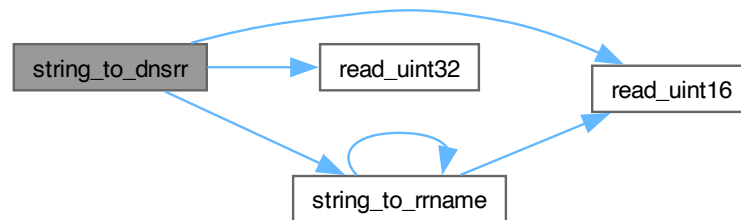
## Parameters

prr	The Resource Record
pstring	The start of the byte stream
offset	The offset in the byte stream

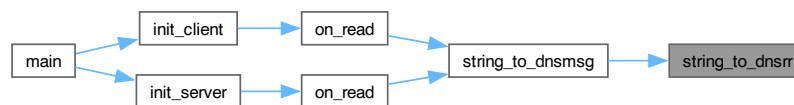
## Note

After reading, the offset increases to the position after the Resource Record; space is allocated for the NAME and RDATA fields

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.29.1.16 string\_to\_rrname()

```
static unsigned string_to_rrname (
    uint8_t * pname,
    const char * pstring,
    unsigned * offset) [static]
```

Read a NAME field from a byte stream.

#### Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

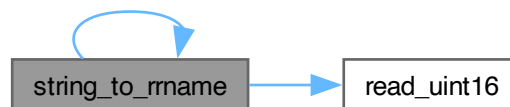
## Returns

The total length of the NAME field

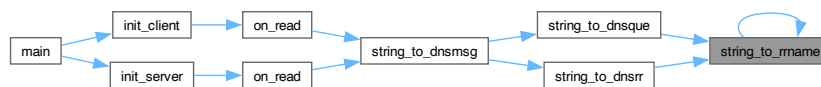
## Note

After reading, the offset increases to the position after the NAME field

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.29.1.17 write\_uint16()

```

static void write_uint16 (
    const char * pstring,
    unsigned * offset,
    uint16_t num) [static]
  
```

Write a 16-bit number in little-endian format to a byte stream.

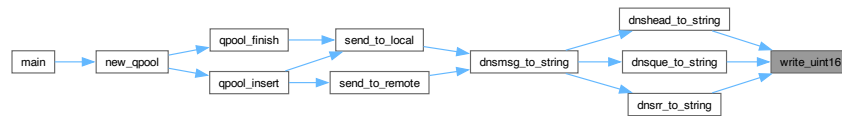
## Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream
num	The number to write

## Note

After writing, the offset increases by 2

Here is the caller graph for this function:



## 5.29.1.18 write\_uint32()

```

static void write_uint32 (
    const char * pstring,
    unsigned * offset,
    uint32_t num) [static]
  
```

Write a 32-bit number in little-endian format to a byte stream.

## Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream
num	The number to write

## Note

After writing, the offset increases by 4

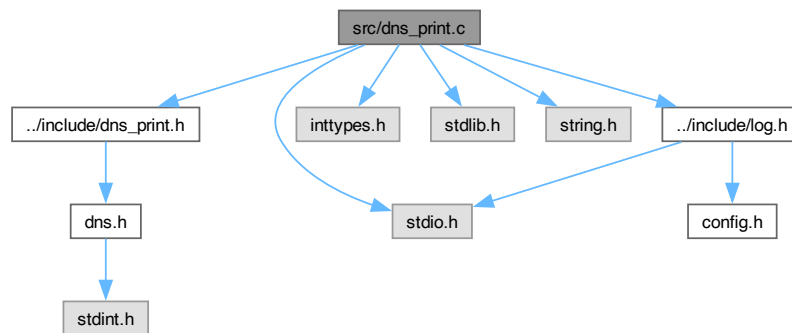
Here is the caller graph for this function:



## 5.30 src/dns\_print.c File Reference

```
#include "../include/dns_print.h"
#include <stdio.h>
#include <inttypes.h>
#include <stdlib.h>
#include <string.h>
#include "../include/log.h"
```

Include dependency graph for dns\_print.c:



### Functions

- void [print\\_dns\\_string](#) (const char \*pstring, unsigned int len)  
Print DNS message byte stream.
- static void [print\\_rr\\_A](#) (const uint8\_t \*rdata)  
Print the rdata field of an A type RR.
- static void [print\\_rr\\_AAAA](#) (const uint8\_t \*rdata)  
Print the rdata field of an AAAA type RR.
- static void [print\\_rr\\_CNAME](#) (const uint8\_t \*rdata)  
Print the rdata field of a CNAME type RR.
- static void [print\\_rr\\_SOA](#) (uint16\_t rdlength, const uint8\_t \*rdata)  
Print the rdata field of an SOA type RR.
- static void [print\\_rr\\_MX](#) (const uint8\_t \*rdata)  
Print the rdata field of an MX type RR.
- static void [print\\_dns\\_header](#) (const [Dns\\_Header](#) \*phead)  
Print the Header Section.
- static void [print\\_dns\\_question](#) (const [Dns\\_Que](#) \*pque)  
Print the Question Section.
- static void [print\\_dns\\_rr](#) (const [Dns\\_RR](#) \*prr)

Print the Resource Record.

- void `print_dns_message` (const `Dns_Msg` \*pmsg)

Print the entire DNS message.

### 5.30.1 Function Documentation

#### 5.30.1.1 `print_dns_header()`

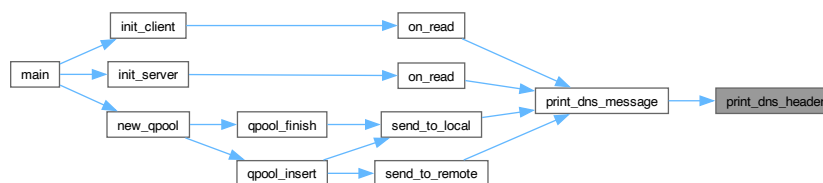
```
static void print_dns_header (
    const Dns_Header * phead)  [static]
```

Print the Header Section.

Parameters

phead	The Header Section
-------	--------------------

Here is the caller graph for this function:



#### 5.30.1.2 `print_dns_message()`

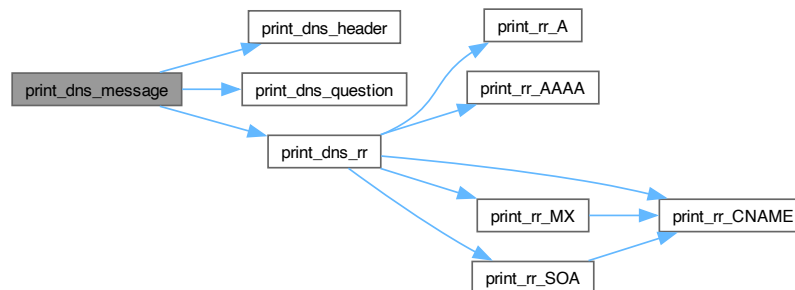
```
void print_dns_message (
    const Dns_Msg * pmsg)
```

Print the entire DNS message.

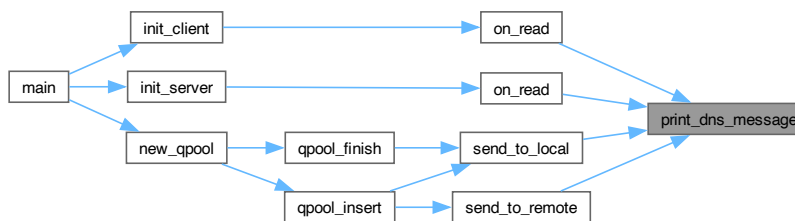
Parameters

pmsg	The DNS message
------	-----------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.30.1.3 print\_dns\_question()

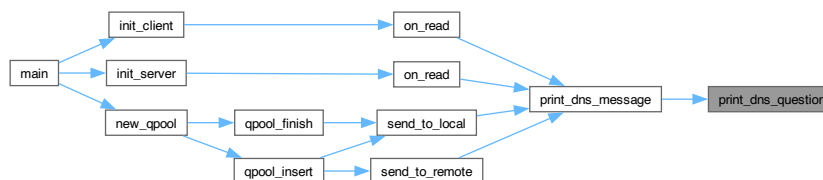
```
static void print_dns_question (
    const Dns_Que * pque) [static]
```

Print the Question Section.

Parameters

pque	The Question Section
------	----------------------

Here is the caller graph for this function:



#### 5.30.1.4 print\_dns\_rr()

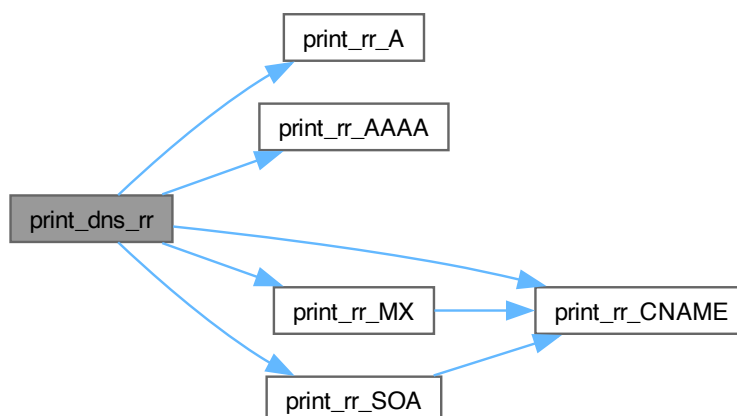
```
static void print_dns_rr (
    const Dns_RR * prr) [static]
```

Print the Resource Record.

Parameters

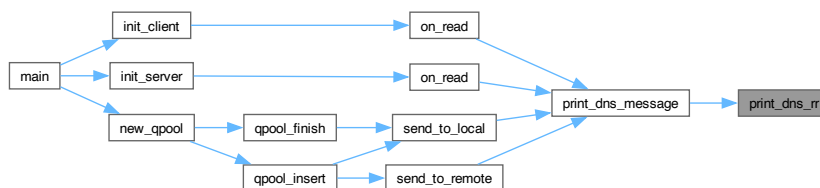
prr	The Resource Record
-----	---------------------

Here is the call graph for this function:





Here is the caller graph for this function:



### 5.30.1.5 print\_dns\_string()

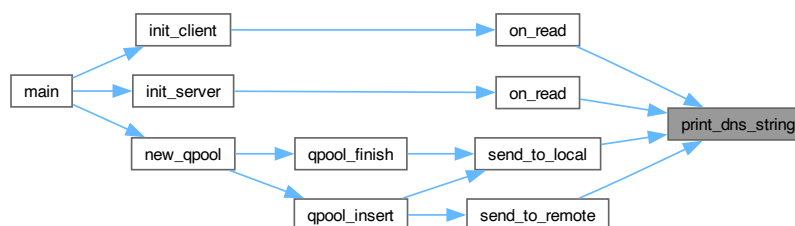
```
void print_dns_string (
    const char * pstring,
    unsigned int len)
```

Print DNS message byte stream.

Parameters

pstring	The byte stream
len	The length of the byte stream

Here is the caller graph for this function:



### 5.30.1.6 print\_rr\_A()

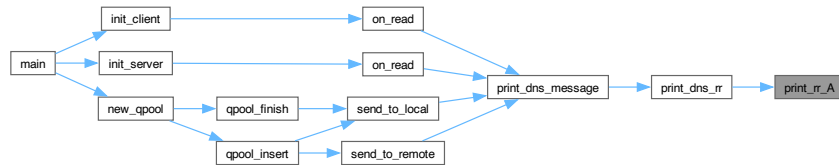
```
static void print_rr_A (
    const uint8_t * rdata) [static]
```

Print the rdata field of an A type RR.

## Parameters

rdata	The rdata field
-------	-----------------

Here is the caller graph for this function:



## 5.30.1.7 print\_rr\_AAAA()

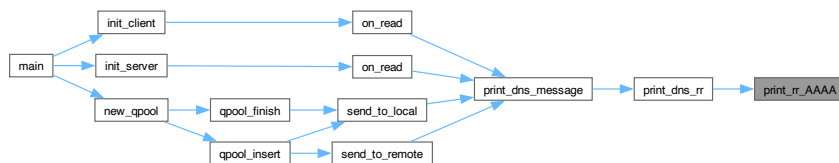
```
static void print_rr_AAAA (
    const uint8_t * rdata) [static]
```

Print the rdata field of an AAAA type RR.

## Parameters

rdata	The rdata field
-------	-----------------

Here is the caller graph for this function:



## 5.30.1.8 print\_rr\_CNAME()

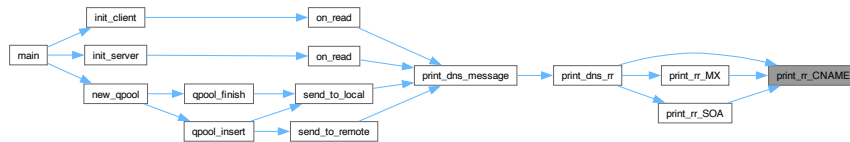
```
static void print_rr_CNAME (
    const uint8_t * rdata) [static]
```

Print the rdata field of a CNAME type RR.

## Parameters

rdata	The rdata field
-------	-----------------

Here is the caller graph for this function:



## 5.30.1.9 print\_rr\_MX()

```
static void print_rr_MX (
    const uint8_t * rdata)  [static]
```

Print the rdata field of an MX type RR.

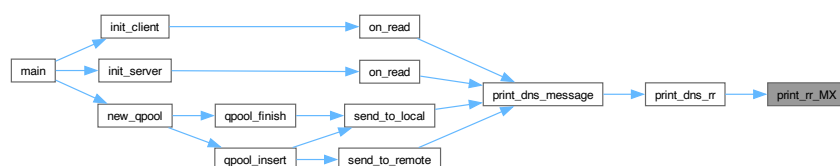
## Parameters

rdata	The rdata field
-------	-----------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.30.1.10 print\_rr\_SOA()

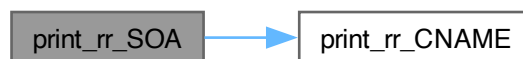
```
static void print_rr_SOA (
    uint16_t rdlength,
    const uint8_t * rdata)  [static]
```

Print the rdata field of an SOA type RR.

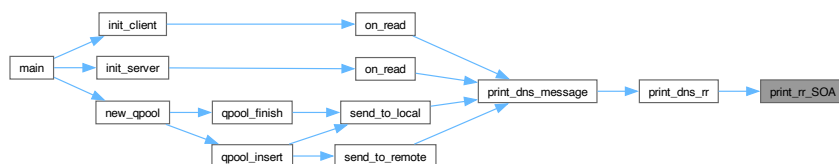
Parameters

rdlength	The rdlength field
rdata	The rdata field

Here is the call graph for this function:



Here is the caller graph for this function:

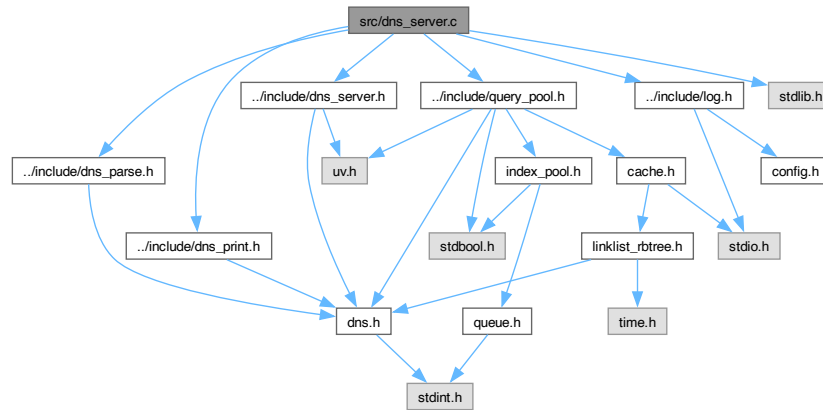


## 5.31 src/dns\_server.c File Reference

```
#include "../include/dns_server.h"
#include <stdlib.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
#include "../include/dns_print.h"
```

```
#include "../include/query_pool.h"
```

Include dependency graph for dns\_server.c:



## Functions

- static void [alloc\\_buffer](#) (uv\_handle\_t \*handle, size\_t suggested\_size, uv\_buf\_t \*buf)  
Allocate space for the buffer.
- static void [on\\_send](#) (uv\_udp\_send\_t \*req, int status)  
Callback function for sending response messages to local clients.
- static void [on\\_read](#) (uv\_udp\_t \*handle, ssize\_t nread, const uv\_buf\_t \*buf, const struct sockaddr \*addr, unsigned flags)  
Callback function for receiving query messages from local clients.
- void [init\\_server](#) (uv\_loop\_t \*loop)  
Initialize the DNS server.
- void [send\\_to\\_local](#) (const struct sockaddr \*addr, const [Dns\\_Msg](#) \*msg)  
Send a DNS response message to local clients.

## Variables

- static uv\_udp\_t [server\\_socket](#)  
Socket for server communication with local clients.
- static struct sockaddr\_in [recv\\_addr](#)  
Address for receiving DNS query messages.
- [Query\\_Pool](#) \* [qpool](#)  
Query pool.

### 5.31.1 Function Documentation

#### 5.31.1.1 alloc\_buffer()

```
static void alloc_buffer (  
    uv_handle_t * handle,  
    size_t suggested_size,  
    uv_buf_t * buf) [static]
```

Allocate space for the buffer.

Parameters

handle	Allocation handle
suggested_size	Suggested buffer size
buf	Buffer to be allocated

Allocates a buffer of fixed size `DNS_STRING_MAX_SIZE` for receiving DNS query messages from local clients. Here is the caller graph for this function:



#### 5.31.1.2 init\_server()

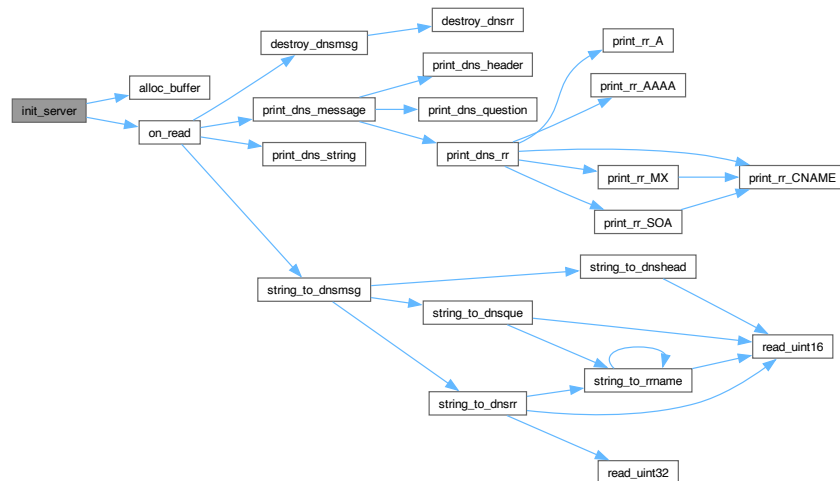
```
void init_server (  
    uv_loop_t * loop)
```

Initialize the DNS server.

Parameters

loop	The libuv event loop
------	----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.31.1.3 on\_read()

```

static void on_read (
    uv_udp_t * handle,
    ssize_t nread,
    const uv_buf_t * buf,
    const struct sockaddr * addr,
    unsigned flags) [static]

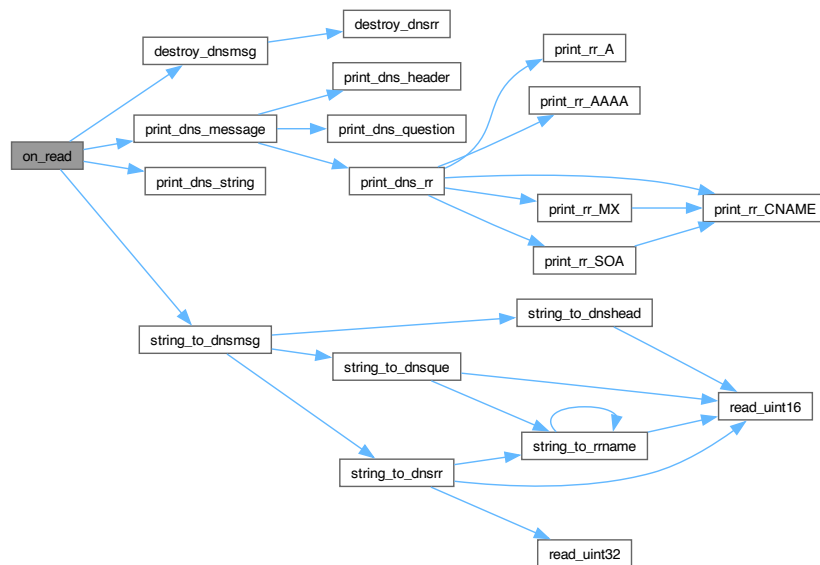
```

Callback function for receiving query messages from local clients.

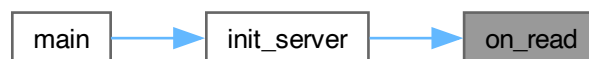
## Parameters

handle	Query handle
nread	Number of bytes received
buf	Buffer containing the received message
addr	Address of the local sender
flags	Flags indicating special conditions for the received data

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.31.1.4 on\_send()

```

static void on_send (
    uv_udp_send_t * req,
    int status) [static]

```

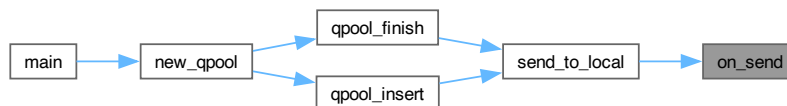


Callback function for sending response messages to local clients.

## Parameters

req	Send handle
status	Send status, indicating whether the send was successful

Here is the caller graph for this function:



## 5.31.1.5 send\_to\_local()

```

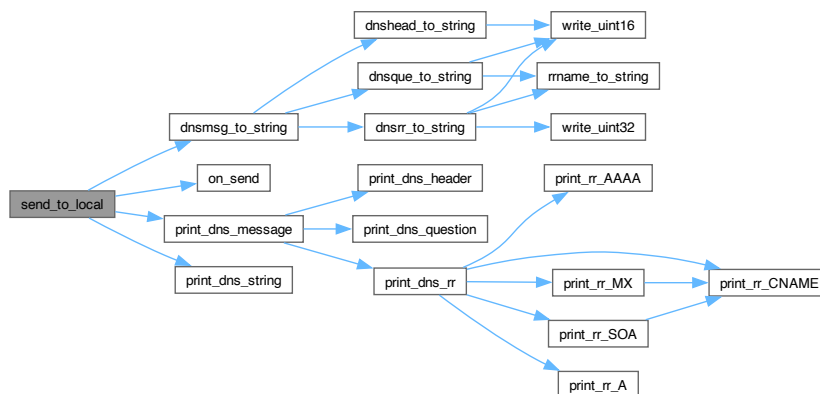
void send_to_local (
    const struct sockaddr * addr,
    const Dns\_Msg * msg)
  
```

Send a DNS response message to local clients.

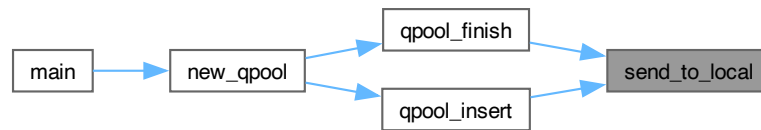
## Parameters

addr	The address of the local client
msg	The DNS message to be sent

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.31.2 Variable Documentation

### 5.31.2.1 qpool

`Query_Pool*` `qpool` [extern]

Query pool.

### 5.31.2.2 recv\_addr

`struct sockaddr_in` `recv_addr` [static]

Address for receiving DNS query messages.

### 5.31.2.3 server\_socket

`uv_udp_t` `server_socket` [static]

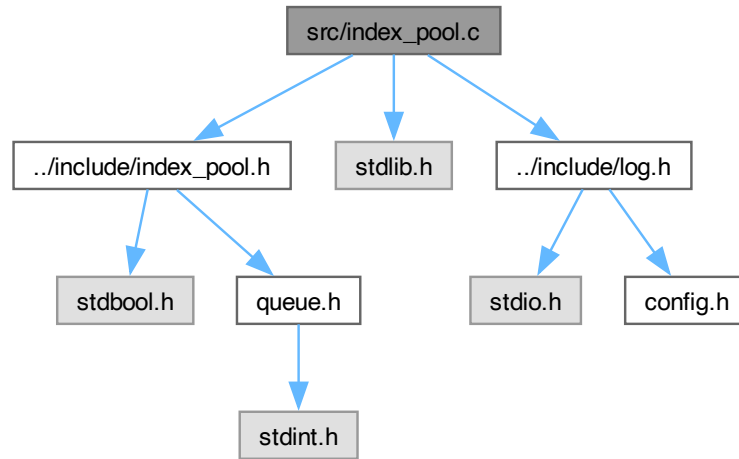
Socket for server communication with local clients.

## 5.32 src/index\_pool.c File Reference

```
#include "../include/index_pool.h"  
#include <stdlib.h>
```

```
#include "../include/log.h"
```

Include dependency graph for `index_pool.c`:



## Functions

- static bool `ipool_full` (`Index_Pool` \*`ipool`)  
Check if the index pool is full.
- static uint16\_t `ipool_insert` (`Index_Pool` \*`ipool`, `Index` \*`req`)  
Insert an index into the pool.
- static bool `ipool_query` (`Index_Pool` \*`ipool`, uint16\_t `index`)  
Query if an index exists in the pool.
- static `Index` \* `ipool_delete` (`Index_Pool` \*`ipool`, uint16\_t `index`)  
Delete an index from the pool.
- static void `ipool_destroy` (`Index_Pool` \*`ipool`)  
Destroy the index pool.
- `Index_Pool` \* `new_ipool` ()  
Create a new index pool.

## 5.32.1 Function Documentation

### 5.32.1.1 `ipool_delete()`

```
static Index * ipool_delete (
    Index_Pool * ipool,
    uint16_t index) [static]
```

Delete an index from the pool.

## Parameters

ipool	The index pool
index	The index to delete

## Returns

The deleted index

Here is the caller graph for this function:



## 5.32.1.2 ipool\_destroy()

```
static void ipool_destroy (  
    Index_Pool * ipool) [static]
```

Destroy the index pool.

## Parameters

ipool	The index pool to destroy
-------	---------------------------

Here is the caller graph for this function:



## 5.32.1.3 ipool\_full()

```
static bool ipool_full (  
    Index_Pool * ipool) [static]
```

Check if the index pool is full.

## Parameters

ipool	The index pool
-------	----------------

## Returns

True if the index pool is full, false otherwise

Here is the caller graph for this function:



## 5.32.1.4 ipool\_insert()

```
static uint16_t ipool_insert (
    Index_Pool * ipool,
    Index * req) [static]
```

Insert an index into the pool.

## Parameters

ipool	The index pool
req	The index to insert

## Returns

The ID of the inserted index

Here is the caller graph for this function:



### 5.32.1.5 ipool\_query()

```
static bool ipool_query (  
    Index_Pool * ipool,  
    uint16_t index) [static]
```

Query if an index exists in the pool.

Parameters

ipool	The index pool
index	The index to query

Returns

True if the index exists, false otherwise

Here is the caller graph for this function:



### 5.32.1.6 new\_ipool()

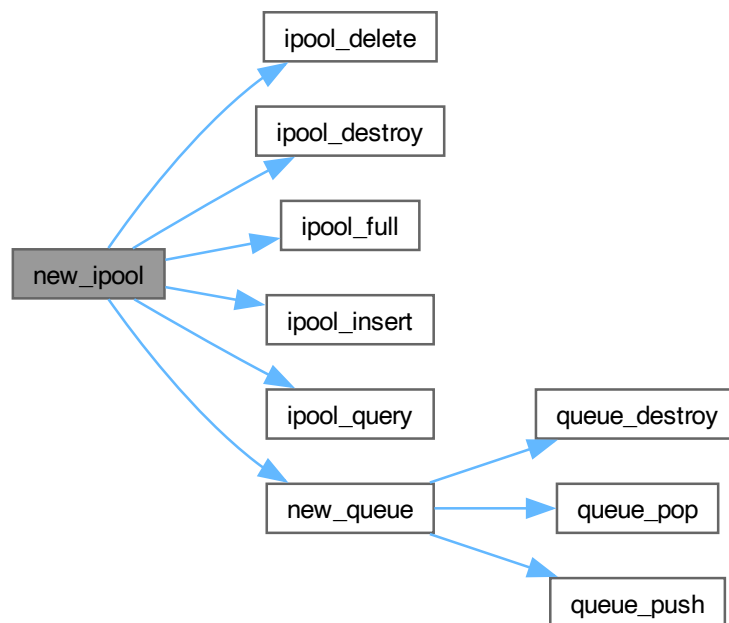
```
Index_Pool * new_ipool ()
```

Create a new index pool.

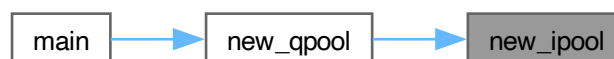
Returns

The new index pool

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.33 src/linklist\_rbtrees.c File Reference

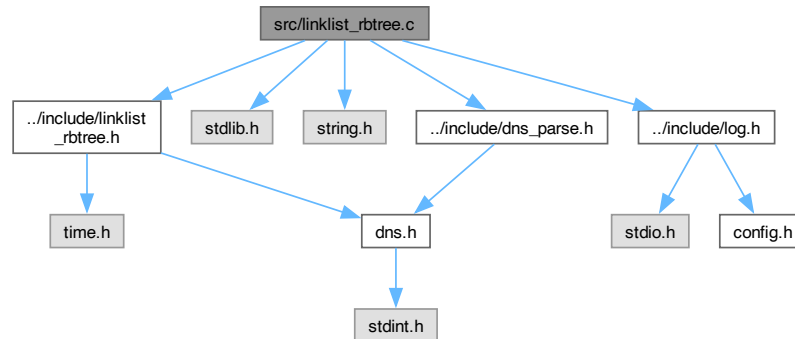
```
#include "../include/linklist_rbtrees.h"  
#include <stdlib.h>  
#include <string.h>
```



```
#include "../include/dns_parse.h"
```

```
#include "../include/log.h"
```

Include dependency graph for linklist\_rbtrees.c:



## Functions

- static void [linklist\\_insert](#) ([Dns\\_RR\\_LinkList](#) \*list, [Dns\\_RR\\_LinkList](#) \*new\_list\_node)  
Insert a new element into the linked list.
- static void [linklist\\_delete\\_next](#) ([Dns\\_RR\\_LinkList](#) \*list)  
Delete the next element in the linked list.
- static [Dns\\_RR\\_LinkList](#) \* [linklist\\_query\\_next](#) ([Dns\\_RR\\_LinkList](#) \*list, const uint8\_t \*qname, uint16\_t qtype)  
Query the next element in the linked list.
- [Dns\\_RR\\_LinkList](#) \* [new\\_linklist](#) ()  
Create a new linked list.
- static [Rbtree\\_Node](#) \* [grandparent](#) ([Rbtree\\_Node](#) \*node)  
Get the grandparent of a node.
- static [Rbtree\\_Node](#) \* [uncle](#) ([Rbtree\\_Node](#) \*node)  
Get the uncle of a node.
- static [Rbtree\\_Node](#) \* [sibling](#) ([Rbtree\\_Node](#) \*node)  
Get the sibling of a node.
- static [Rbtree\\_Node](#) \* [smallest\\_child](#) ([Rbtree\\_Node](#) \*node)  
Get the smallest child node in a subtree.
- static void [rotate\\_right](#) ([Rbtree](#) \*tree, [Rbtree\\_Node](#) \*node)  
Rotate a node to the right.
- static void [rotate\\_left](#) ([Rbtree](#) \*tree, [Rbtree\\_Node](#) \*node)  
Rotate a node to the left.
- static void [insert\\_case](#) ([Rbtree](#) \*tree, [Rbtree\\_Node](#) \*node)  
Adjust the shape of the red-black tree to keep it balanced.

- static `Rbtree_Node * node_init` (unsigned int key, `Dns_RR_LinkList *list`, `Rbtree_Node *fa`)  
Initialize a node and allocate memory.
- void `rbtree_insert` (`Rbtree *tree`, unsigned int key, `Dns_RR_LinkList *list`)  
Insert a key-value pair into the red-black tree.
- static `Rbtree_Node * rbtree_find` (`Rbtree_Node *node`, unsigned int key)  
Recursively search for a node with a given key starting from a given node.
- static void `destroy_node` (`Rbtree_Node *node`)  
Destroy a node in the red-black tree.
- static void `delete_case` (`Rbtree *tree`, `Rbtree_Node *node`)  
Adjust the shape of the red-black tree to keep it balanced.
- static void `rbtree_delete` (`Rbtree *tree`, `Rbtree_Node *node`)  
Delete a node from the red-black tree.
- `Dns_RR_LinkList * rbtree_query` (`Rbtree *tree`, unsigned int key)  
Query the red-black tree for a key.
- `Rbtree * new_rbtree` ()  
Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

## Variables

- static `Rbtree_Node * NIL`  
Leaf node.

## 5.33.1 Function Documentation

### 5.33.1.1 `delete_case()`

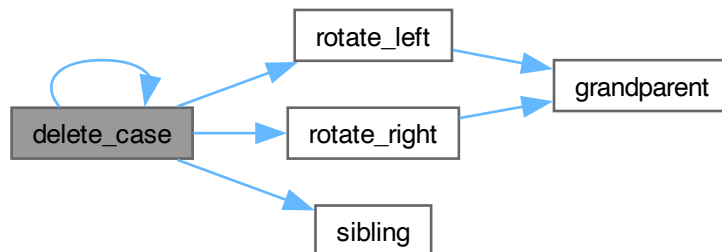
```
static void delete_case (
    Rbtree * tree,
    Rbtree_Node * node) [static]
```

Adjust the shape of the red-black tree to keep it balanced.

#### Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.2 `destroy_node()`

```
static void destroy_node (
    Rbtree_Node * node) [static]
```

Destroy a node in the red-black tree.

Parameters

node	The node to destroy
------	---------------------

Note

The linked list of the node is assumed to be empty (i.e., only the head node)

Here is the caller graph for this function:



### 5.33.1.3 grandparent()

```
static Rbtree_Node * grandparent (  
    Rbtree_Node * node)  [inline], [static]
```

Get the grandparent of a node.

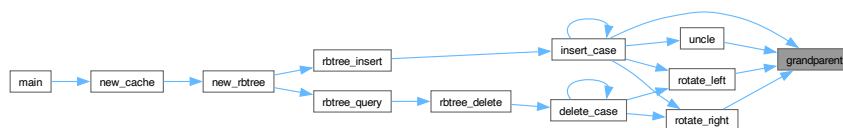
Parameters

node	The current node
------	------------------

Returns

The grandparent node if exists, otherwise NULL

Here is the caller graph for this function:



### 5.33.1.4 insert\_\_case()

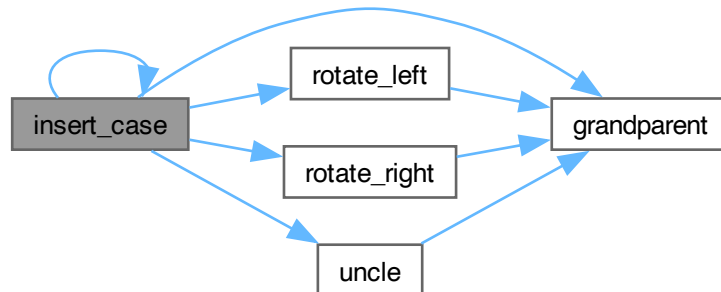
```
static void insert__case (  
    Rbtree * tree,  
    Rbtree_Node * node)  [static]
```

Adjust the shape of the red-black tree to keep it balanced.

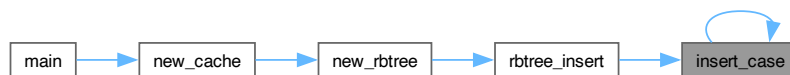
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.5 linklist\_delete\_next()

```
static void linklist_delete_next (
    Dns_RR_LinkList * list) [static]
```

Delete the next element in the linked list.

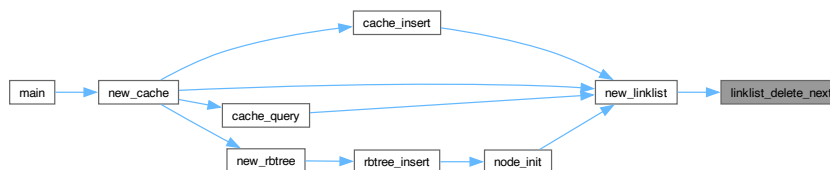
Parameters

list	The linked list
------	-----------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.33.1.6 linklist\_insert()

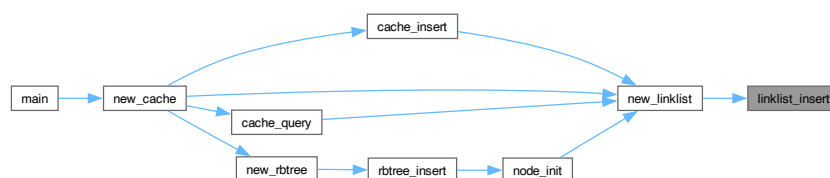
```
static void linklist_insert (
    Dns_RR_LinkList * list,
    Dns_RR_LinkList * new_list_node) [static]
```

Insert a new element into the linked list.

Parameters

list	The linked list
new_list_node	The new element to insert

Here is the caller graph for this function:



### 5.33.1.7 linklist\_query\_next()

```
static Dns_RR_LinkList * linklist_query_next (
    Dns_RR_LinkList * list,
    const uint8_t * qname,
    uint16_t qtype) [static]
```

Query the next element in the linked list.

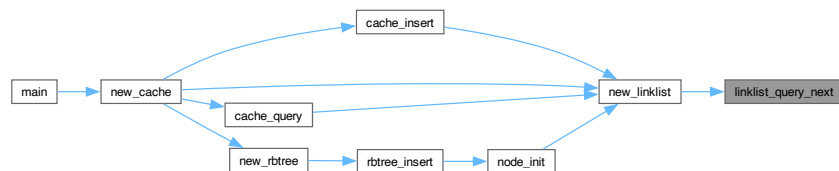
## Parameters

list	The linked list
qname	The query name
qtype	The query type

## Returns

The queried element if found, otherwise NULL

Here is the caller graph for this function:



## 5.33.1.8 new\_linklist()

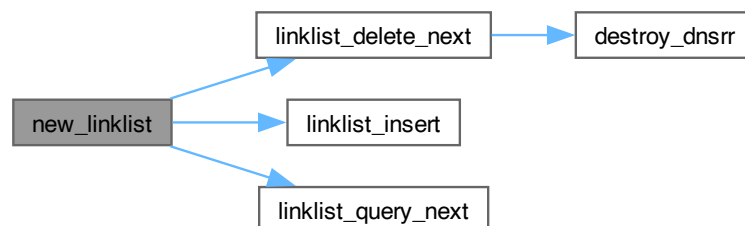
```
Dns_RR_LinkList * new_linklist ()
```

Create a new linked list.

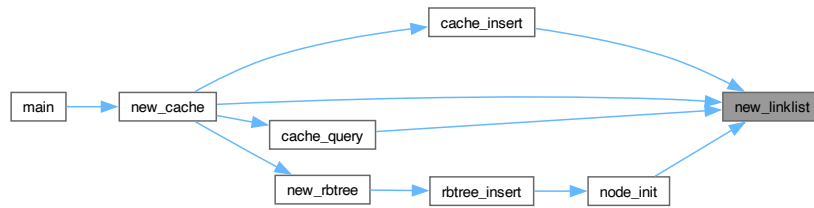
## Returns

The new linked list

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.33.1.9 new\_rbtree()

**Rbtree** \* new\_rbtree ()

Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

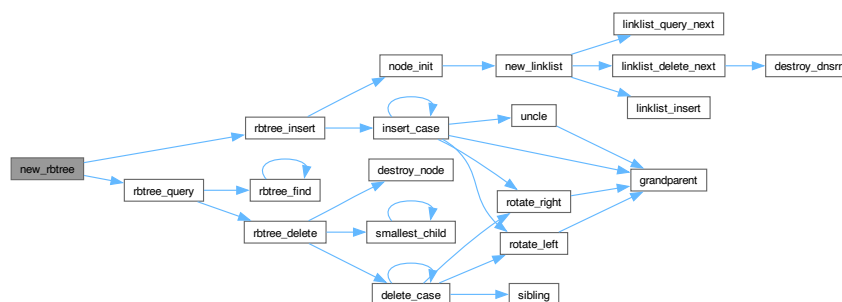
Returns

A pointer to the newly created red-black tree

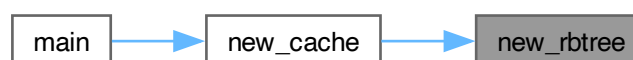
Note

If memory allocation fails, the function will log a fatal error and terminate the program.

Here is the call graph for this function:



Here is the caller graph for this function:





## 5.33.1.10 node\_init()

```
static Rbtree_Node * node_init (
    unsigned int key,
    Dns_RR_LinkList * list,
    Rbtree_Node * fa)  [static]
```

Initialize a node and allocate memory.

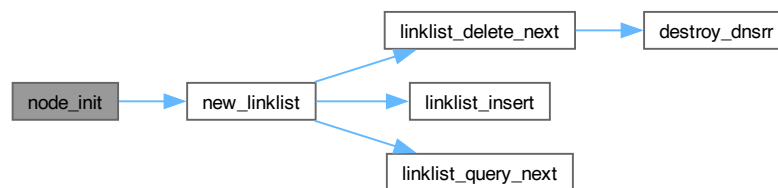
Parameters

key	The key of the node
list	The value of the node
fa	The parent node

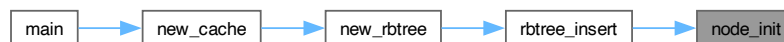
Returns

A pointer to the new node

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.33.1.11 rbtree\_delete()

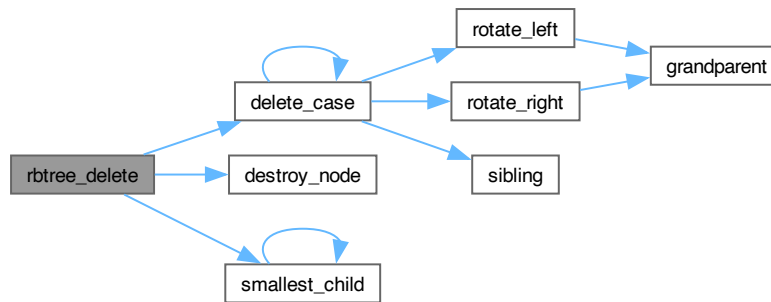
```
static void rbtree_delete (
    Rbtree * tree,
    Rbtree_Node * node)  [static]
```

Delete a node from the red-black tree.

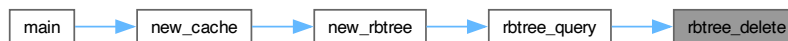
## Parameters

tree	The tree that the node belongs to
node	The node to delete

Here is the call graph for this function:



Here is the caller graph for this function:

5.33.1.12 `rbtree_find()`

```
static Rbtree_Node * rbtrees_find (
    Rbtree_Node * node,
    unsigned int key) [static]
```

Recursively search for a node with a given key starting from a given node.

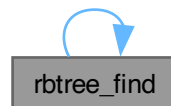
## Parameters

node	The current node
key	The key to search for

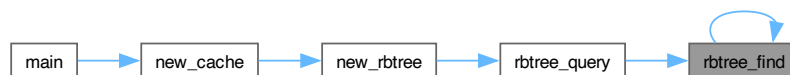
Returns

A pointer to the node if found, otherwise NULL

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.13 rbtrees\_insert()

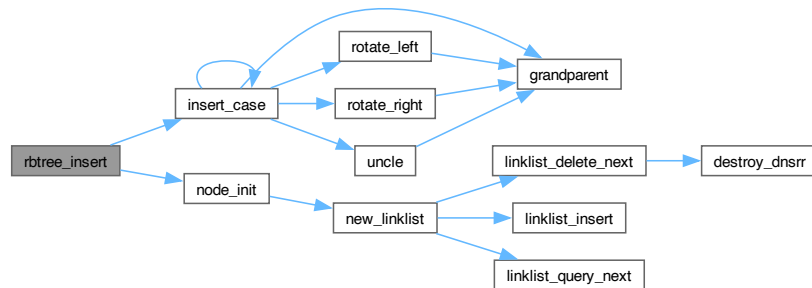
```
void rbtrees_insert (  
    Rbtrees * tree,  
    unsigned int key,  
    Dns_RR_LinkList * list)
```

Insert a key-value pair into the red-black tree.

Parameters

tree	The red-black tree
key	The key
list	The value

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.14 rbtree\_query()

```

Dns_RR_LinkList * rbtree_query (
    Rbtree * tree,
    unsigned int key)

```

Query the red-black tree for a key.

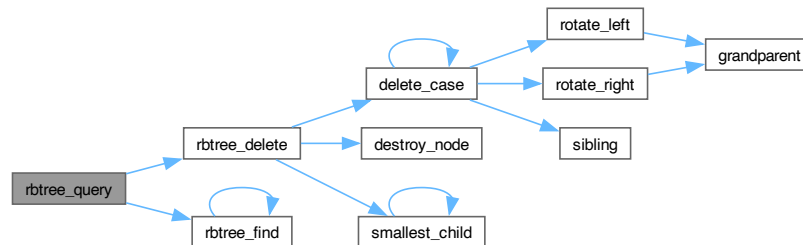
Parameters

tree	The red-black tree
key	The key to query

Returns

The linked list of the value if found, otherwise NULL

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.15 rotate\_left()

```

static void rotate_left (
    Rbtree * tree,
    Rbtree_Node * node) [static]
  
```

Rotate a node to the left.

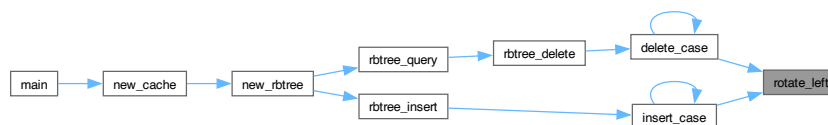
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.33.1.16 rotate\_right()

```
static void rotate_right (
    Rbtree * tree,
    Rbtree_Node * node) [static]
```

Rotate a node to the right.

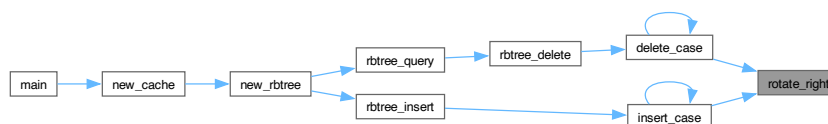
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.33.1.17 sibling()

```
static Rbtree__Node * sibling (
    Rbtree__Node * node)  [inline], [static]
```

Get the sibling of a node.

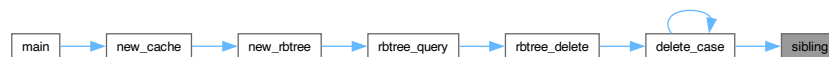
Parameters

node	The current node
------	------------------

Returns

The sibling node if exists, otherwise NULL

Here is the caller graph for this function:



## 5.33.1.18 smallest\_child()

```
static Rbtree__Node * smallest_child (
    Rbtree__Node * node)  [static]
```

Get the smallest child node in a subtree.

Parameters

node	The root of the subtree
------	-------------------------

## Returns

The smallest child node

Here is the call graph for this function:



Here is the caller graph for this function:

5.33.1.19 `uncle()`

```
static Rbtree__Node * uncle (
    Rbtree__Node * node)  [inline], [static]
```

Get the uncle of a node.

## Parameters

node	The current node
------	------------------

## Returns

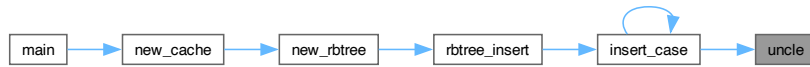
The uncle node if exists, otherwise NULL

Here is the call graph for this function:





Here is the caller graph for this function:



## 5.33.2 Variable Documentation

### 5.33.2.1 NIL

`Rbtree_Node* NIL` [static]

Leaf node.

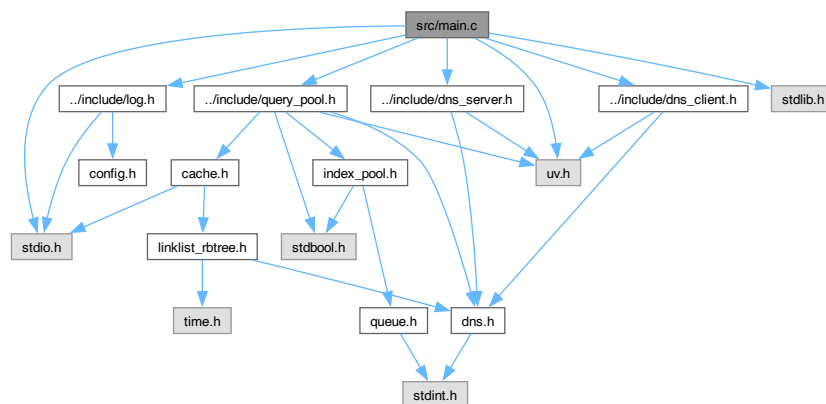
## 5.34 src/main.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <uv.h>
#include "../include/log.h"
#include "../include/dns_client.h"
#include "../include/dns_server.h"
#include "../include/query_pool.h"

```

Include dependency graph for main.c:



## Functions

- int [main](#) (int argc, char \*argv[])

## Variables

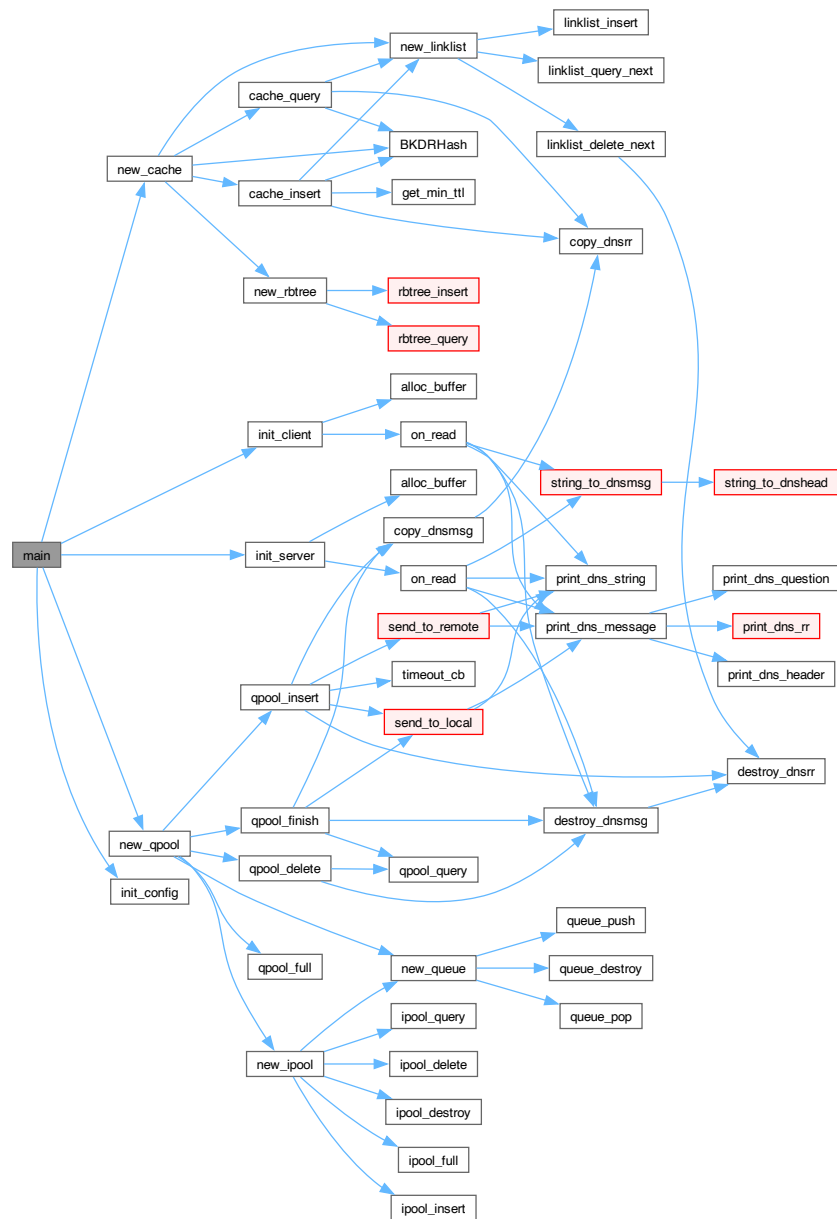
- uv\_loop\_t \* [loop](#)
- [Cache](#) \* [cache](#)
- [Query\\_Pool](#) \* [qpool](#)  
Query pool.
- FILE \* [log\\_file](#)

## 5.34.1 Function Documentation

### 5.34.1.1 main()

```
int main (  
    int argc,  
    char * argv[])
```

Here is the call graph for this function:



## 5.34.2 Variable Documentation

### 5.34.2.1 cache

Cache\* cache

### 5.34.2.2 log\_file

FILE\* log\_file

### 5.34.2.3 loop

`uv_loop_t* loop`

### 5.34.2.4 qpool

`Query_Pool* qpool`

Query pool.

## 5.35 src/query\_pool.c File Reference

```
#include "../include/query_pool.h"
```

```
#include <stdlib.h>
```

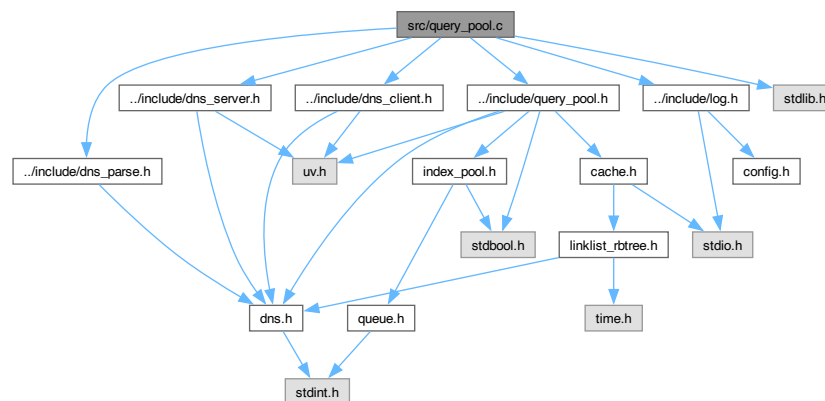
```
#include "../include/log.h"
```

```
#include "../include/dns_parse.h"
```

```
#include "../include/dns_client.h"
```

```
#include "../include/dns_server.h"
```

Include dependency graph for query\_pool.c:



## Functions

- static void [timeout\\_cb](#) (uv\_timer\_t \*timer)

Timeout callback function This function is called when a query times out. It stops the timer and deletes the query from the query pool.

- static bool [qpool\\_full](#) ([Query\\_Pool](#) \*this)

Check if the query pool is full.

- static void `qpool_insert` (`Query_Pool *qpool`, const struct sockaddr \*addr, const `Dns_Msg *msg`)  
Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.
- static bool `qpool_query` (`Query_Pool *qpool`, uint16\_t id)  
Check if a query exists in the query pool.
- static void `qpool_finish` (`Query_Pool *qpool`, const `Dns_Msg *msg`)  
Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.
- static void `qpool_delete` (`Query_Pool *qpool`, uint16\_t id)  
Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.
- `Query_Pool * new_qpool` (`uv_loop_t *loop`, `Cache *cache`)  
Create a new query pool This function initializes a new query pool and returns a pointer to it.

### 5.35.1 Function Documentation

#### 5.35.1.1 `new_qpool()`

```
Query_Pool * new_qpool (
    uv_loop_t * loop,
    Cache * cache)
```

Create a new query pool This function initializes a new query pool and returns a pointer to it.

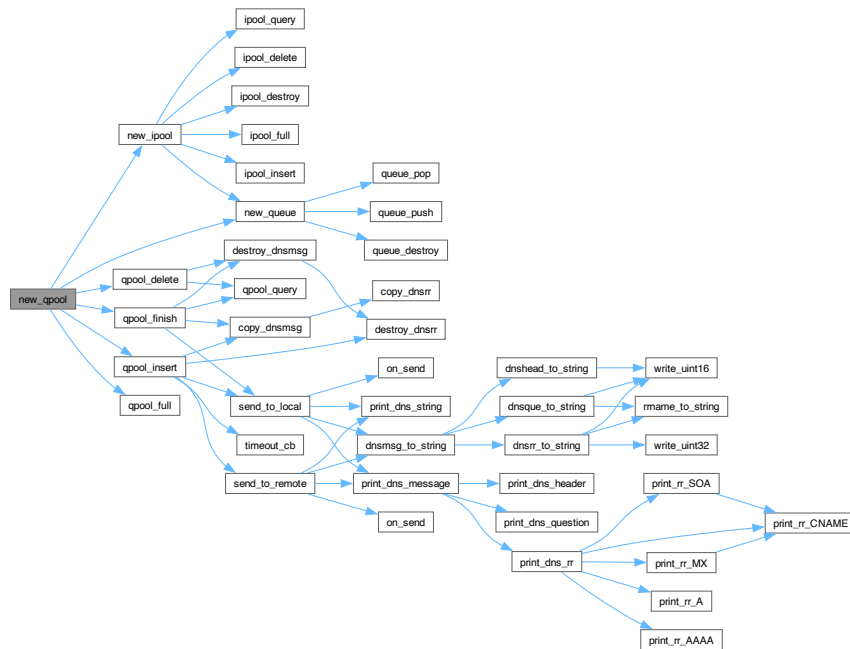
Parameters

loop	The libuv event loop
cache	The cache used for storing DNS responses

Returns

A pointer to the newly created query pool

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.35.1.2 qpool\_delete()

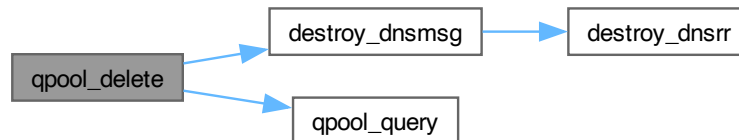
```
static void qpool_delete (
    Query_Pool * qpool,
    uint16_t id) [static]
```

Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

## Parameters

qpool	The query pool
id	The ID of the query to be deleted

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.35.1.3 qpool\_finish()

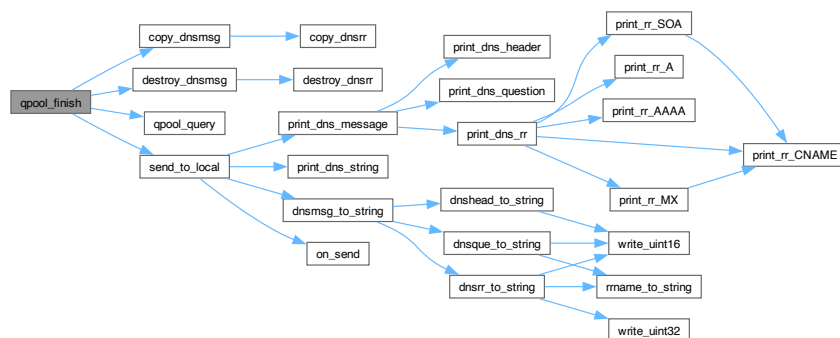
```
static void qpool_finish (  
    Query_Pool * qpool,  
    const Dns_Msg * msg) [static]
```

Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.

## Parameters

qpool	The query pool
msg	The DNS message containing the response

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.35.1.4 qpool\_full()

```
static bool qpool_full (
    Query_Pool * this) [static]
```

Check if the query pool is full.

Parameters

this	The query pool
------	----------------



## Returns

true if the query pool is full, false otherwise

Here is the caller graph for this function:



## 5.35.1.5 qpool\_insert()

```

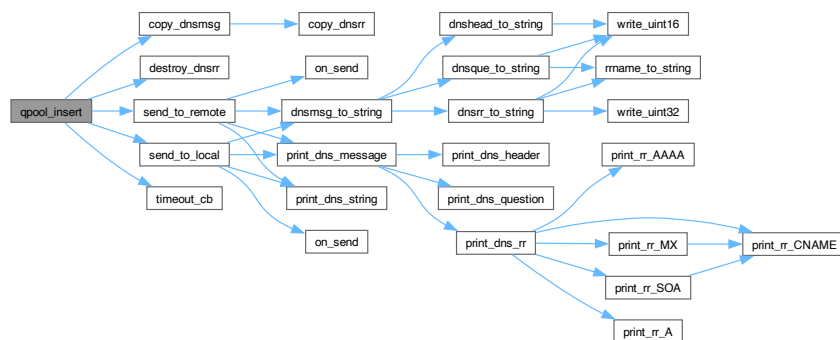
static void qpool_insert (
    Query_Pool * qpool,
    const struct sockaddr * addr,
    const Dns_Msg * msg) [static]
  
```

Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.

## Parameters

qpool	The query pool
addr	The address of the client
msg	The DNS message containing the query

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.35.1.6 qpool\_query()

```
static bool qpool_query (
    Query_Pool * qpool,
    uint16_t id) [static]
```

Check if a query exists in the query pool.

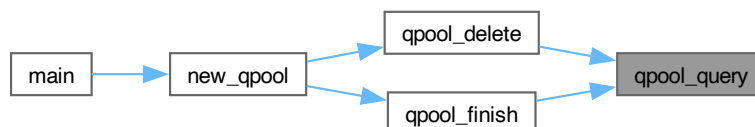
Parameters

qpool	The query pool
id	The ID of the query

Returns

true if the query exists in the query pool, false otherwise

Here is the caller graph for this function:



#### 5.35.1.7 timeout\_cb()

```
static void timeout_cb (
    uv_timer_t * timer) [static]
```

Timeout callback function This function is called when a query times out. It stops the timer and deletes the query from the query pool.

## Parameters

timer	The timer that timed out
-------	--------------------------

Here is the caller graph for this function:



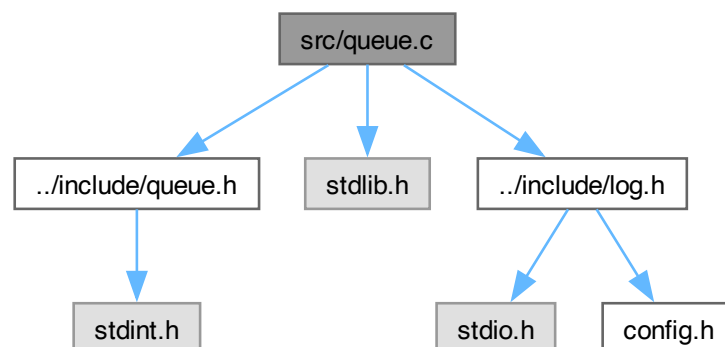
## 5.36 src/queue.c File Reference

```
#include "../include/queue.h"
```

```
#include <stdlib.h>
```

```
#include "../include/log.h"
```

Include dependency graph for queue.c:



## Functions

- static void `queue_push` (`Queue *queue`, `uint16_t num`)  
Push a number onto the queue.
- static `uint16_t` `queue_pop` (`Queue *queue`)  
Pop a number from the queue.
- static void `queue_destroy` (`Queue *queue`)

Destroy the queue.

- `Queue * new_queue ()`

Create a new queue.

### 5.36.1 Function Documentation

#### 5.36.1.1 `new_queue()`

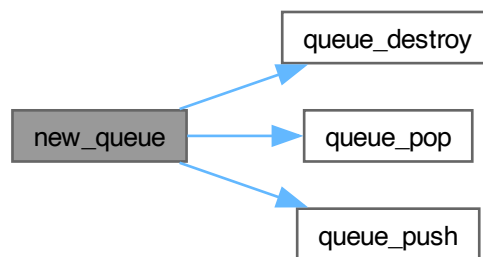
`Queue * new_queue ()`

Create a new queue.

Returns

The new queue

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.36.1.2 `queue_destroy()`

`static void queue_destroy (`

`Queue * queue) [static]`

Destroy the queue.

## Parameters

queue	The queue to destroy
-------	----------------------

Here is the caller graph for this function:



## 5.36.1.3 queue\_pop()

```
static uint16_t queue_pop (
    Queue * queue) [static]
```

Pop a number from the queue.

## Parameters

queue	The queue
-------	-----------

## Returns

The number popped from the queue

Here is the caller graph for this function:



## 5.36.1.4 queue\_push()

```
static void queue_push (
    Queue * queue,
    uint16_t num) [static]
```

Push a number onto the queue.

## Parameters

queue	The queue
num	The number to push

Here is the caller graph for this function:



# 索引

- aa
  - dns\_header, [13](#)
- addr
  - dns\_query, [17](#)
- alloc\_buffer
  - dns\_client.c, [96](#)
  - dns\_server.c, [128](#)
- ancount
  - dns\_header, [13](#)
  - rbtree\_value, [39](#)
- arcount
  - dns\_header, [13](#)
  - rbtree\_value, [39](#)
- BKDRHash
  - cache.c, [88](#)
- BLACK
  - linklist\_rbtrees.h, [74](#)
- Cache
  - cache.h, [43](#)
- cache
  - main.c, [157](#)
  - query\_pool, [31](#)
- cache.c
  - BKDRHash, [88](#)
  - cache\_insert, [89](#)
  - cache\_query, [90](#)
  - get\_min\_ttl, [91](#)
  - new\_cache, [91](#)
- cache.h
  - Cache, [43](#)
  - CACHE\_SIZE, [42](#)
  - new\_cache, [43](#)
- cache\_, [9](#)
  - head, [11](#)
  - insert, [11](#)
  - query, [11](#)
  - size, [11](#)
  - tail, [12](#)
  - tree, [12](#)
- cache\_insert
  - cache.c, [89](#)
- cache\_query
  - cache.c, [90](#)
- CACHE\_SIZE
  - cache.h, [42](#)
- class
  - dns\_rr, [20](#)
- CLIENT\_PORT
  - config.c, [94](#)
  - config.h, [45](#)
- client\_socket
  - dns\_client.c, [101](#)
- Color
  - linklist\_rbtrees.h, [74](#)
- color
  - rbtree\_node, [37](#)
- config.c
  - CLIENT\_PORT, [94](#)
  - HOSTS\_PATH, [94](#)
  - init\_config, [93](#)
  - LOG\_MASK, [94](#)
  - LOG\_PATH, [94](#)
  - REMOTE\_HOST, [94](#)
- config.h
  - CLIENT\_PORT, [45](#)
  - HOSTS\_PATH, [45](#)
  - init\_config, [45](#)
  - LOG\_MASK, [46](#)
  - LOG\_PATH, [46](#)
  - REMOTE\_HOST, [46](#)
- copy\_dnsmmsg

- dns\_parse.c, [103](#)
- dns\_parse.h, [57](#)
- copy\_dnsrr
  - dns\_parse.c, [104](#)
  - dns\_parse.h, [57](#)
- count
  - index\_pool, [25](#)
  - query\_pool, [31](#)
- delete
  - index\_pool, [25](#)
  - query\_pool, [31](#)
- delete\_case
  - linklist\_rbtrees.c, [140](#)
- delete\_next
  - dns\_rr\_linklist, [22](#)
- destroy
  - index\_pool, [25](#)
  - queue, [34](#)
- destroy\_dnsmsg
  - dns\_parse.c, [105](#)
  - dns\_parse.h, [58](#)
- destroy\_dnsrr
  - dns\_parse.c, [106](#)
  - dns\_parse.h, [59](#)
- destroy\_node
  - linklist\_rbtrees.c, [141](#)
- dns.h
  - DNS\_CLASS\_IN, [48](#)
  - Dns\_Header, [51](#)
  - Dns\_Msg, [51](#)
  - DNS\_OPCODE\_IQUERY, [48](#)
  - DNS\_OPCODE\_QUERY, [49](#)
  - DNS\_OPCODE\_STATUS, [49](#)
  - DNS\_QR\_ANSWER, [49](#)
  - DNS\_QR\_QUERY, [49](#)
  - Dns\_Queue, [51](#)
  - DNS\_RCODE\_NXDOMAIN, [49](#)
  - DNS\_RCODE\_OK, [49](#)
  - DNS\_RCODE\_SERVFAIL, [49](#)
  - Dns\_RR, [51](#)
  - DNS\_RR\_NAME\_MAX\_SIZE, [49](#)
  - DNS\_STRING\_MAX\_SIZE, [49](#)
  - DNS\_TYPE\_A, [50](#)
  - DNS\_TYPE\_AAAA, [50](#)
  - DNS\_TYPE\_CNAME, [50](#)
  - DNS\_TYPE\_HINFO, [50](#)
  - DNS\_TYPE\_MINFO, [50](#)
  - DNS\_TYPE\_MX, [50](#)
  - DNS\_TYPE\_NS, [50](#)
  - DNS\_TYPE\_PTR, [50](#)
  - DNS\_TYPE\_SOA, [50](#)
  - DNS\_TYPE\_TXT, [51](#)
- DNS\_CLASS\_IN
  - dns.h, [48](#)
- dns\_client.c
  - alloc\_buffer, [96](#)
  - client\_socket, [101](#)
  - init\_client, [96](#)
  - local\_addr, [101](#)
  - on\_read, [97](#)
  - on\_send, [98](#)
  - qpool, [101](#)
  - send\_addr, [101](#)
  - send\_to\_remote, [100](#)
- dns\_client.h
  - init\_client, [54](#)
  - send\_to\_remote, [54](#)
- Dns\_Header
  - dns.h, [51](#)
- dns\_header, [12](#)
  - aa, [13](#)
  - ancount, [13](#)
  - arcount, [13](#)
  - id, [13](#)
  - nscount, [13](#)
  - opcode, [13](#)
  - qdcnt, [13](#)
  - qr, [13](#)
  - ra, [13](#)
  - rcode, [14](#)
  - rd, [14](#)
  - tc, [14](#)
  - z, [14](#)
- Dns\_Msg
  - dns.h, [51](#)



- dns\_msg, [14](#)
  - header, [15](#)
  - que, [15](#)
  - rr, [15](#)
- DNS\_OPCODE\_IQUERY
  - dns.h, [48](#)
- DNS\_OPCODE\_QUERY
  - dns.h, [49](#)
- DNS\_OPCODE\_STATUS
  - dns.h, [49](#)
- dns\_parse.c
  - copy\_dnsmmsg, [103](#)
  - copy\_dnsrr, [104](#)
  - destroy\_dnsmmsg, [105](#)
  - destroy\_dnsrr, [106](#)
  - dnshead\_to\_string, [106](#)
  - dnsmmsg\_to\_string, [107](#)
  - dnsque\_to\_string, [108](#)
  - dnsrr\_to\_string, [109](#)
  - read\_uint16, [110](#)
  - read\_uint32, [111](#)
  - rrname\_to\_string, [111](#)
  - string\_to\_dnshead, [112](#)
  - string\_to\_dnsmmsg, [113](#)
  - string\_to\_dnsque, [114](#)
  - string\_to\_dnsrr, [115](#)
  - string\_to\_rrname, [116](#)
  - write\_uint16, [117](#)
  - write\_uint32, [118](#)
- dns\_parse.h
  - copy\_dnsmmsg, [57](#)
  - copy\_dnsrr, [57](#)
  - destroy\_dnsmmsg, [58](#)
  - destroy\_dnsrr, [59](#)
  - dnsmmsg\_to\_string, [59](#)
  - string\_to\_dnsmmsg, [60](#)
- dns\_print.c
  - print\_dns\_header, [120](#)
  - print\_dns\_message, [120](#)
  - print\_dns\_question, [121](#)
  - print\_dns\_rr, [122](#)
  - print\_dns\_string, [123](#)
  - print\_rr\_A, [123](#)
  - print\_rr\_AAAA, [124](#)
  - print\_rr\_CNAME, [124](#)
  - print\_rr\_MX, [125](#)
  - print\_rr\_SOA, [125](#)
- dns\_print.h
  - print\_dns\_message, [63](#)
  - print\_dns\_string, [63](#)
- DNS\_QR\_ANSWER
  - dns.h, [49](#)
- DNS\_QR\_QUERY
  - dns.h, [49](#)
- Dns\_Queue
  - dns.h, [51](#)
- Dns\_Query
  - query\_pool.h, [82](#)
- dns\_query, [16](#)
  - addr, [17](#)
  - id, [17](#)
  - msg, [17](#)
  - prev\_id, [17](#)
  - timer, [17](#)
- dns\_question, [18](#)
  - next, [18](#)
  - qclass, [18](#)
  - qname, [18](#)
  - qtype, [18](#)
- DNS\_RCODE\_NXDOMAIN
  - dns.h, [49](#)
- DNS\_RCODE\_OK
  - dns.h, [49](#)
- DNS\_RCODE\_SERVFAIL
  - dns.h, [49](#)
- Dns\_RR
  - dns.h, [51](#)
- dns\_rr, [19](#)
  - class, [20](#)
  - name, [20](#)
  - next, [20](#)
  - rdata, [20](#)
  - rdlength, [20](#)
  - ttl, [20](#)
  - type, [20](#)
- Dns\_RR\_LinkList

- linklist\_\_rbtree.h, 73
- dns\_rr\_linklist, 21
  - delete\_next, 22
  - expire\_time, 22
  - insert, 22
  - next, 22
  - query\_next, 22
  - value, 23
- DNS\_RR\_NAME\_MAX\_SIZE
  - dns.h, 49
- dns\_server.c
  - alloc\_buffer, 128
  - init\_server, 128
  - on\_read, 129
  - on\_send, 130
  - qpool, 133
  - recv\_addr, 133
  - send\_to\_local, 132
  - server\_socket, 133
- dns\_server.h
  - init\_server, 65
  - send\_to\_local, 66
- DNS\_STRING\_MAX\_SIZE
  - dns.h, 49
- DNS\_TYPE\_A
  - dns.h, 50
- DNS\_TYPE\_AAAA
  - dns.h, 50
- DNS\_TYPE\_CNAME
  - dns.h, 50
- DNS\_TYPE\_HINFO
  - dns.h, 50
- DNS\_TYPE\_MINFO
  - dns.h, 50
- DNS\_TYPE\_MX
  - dns.h, 50
- DNS\_TYPE\_NS
  - dns.h, 50
- DNS\_TYPE\_PTR
  - dns.h, 50
- DNS\_TYPE\_SOA
  - dns.h, 50
- DNS\_TYPE\_TXT
  - dns.h, 51
- dnshead\_to\_string
  - dns\_parse.c, 106
- dnsmsg\_to\_string
  - dns\_parse.c, 107
  - dns\_parse.h, 59
- dnsque\_to\_string
  - dns\_parse.c, 108
- dnsrr\_to\_string
  - dns\_parse.c, 109
- expire\_time
  - dns\_rr\_linklist, 22
- finish
  - query\_pool, 32
- full
  - index\_pool, 26
  - query\_pool, 32
- get\_min\_ttl
  - cache.c, 91
- grandparent
  - linklist\_\_rbtree.c, 141
- head
  - cache\_, 11
  - queue, 34
- header
  - dns\_msg, 15
- HOSTS\_PATH
  - config.c, 94
  - config.h, 45
- id
  - dns\_header, 13
  - dns\_query, 17
  - index\_, 24
- include/cache.h, 41, 44
- include/config.h, 44, 46
- include/dns.h, 47, 52
- include/dns\_client.h, 53, 55
- include/dns\_parse.h, 56, 61
- include/dns\_print.h, 62, 64
- include/dns\_server.h, 64, 67

- include/index\_pool.h, [68](#), [71](#)
- include/linklist\_rbtrees.h, [71](#), [76](#)
- include/log.h, [77](#), [79](#)
- include/query\_pool.h, [80](#), [83](#)
- include/queue.h, [84](#), [87](#)
- Index
  - index\_pool.h, [69](#)
- index\_, [23](#)
  - id, [24](#)
  - prev\_id, [24](#)
- Index\_Pool
  - index\_pool.h, [69](#)
- index\_pool, [24](#)
  - count, [25](#)
  - delete, [25](#)
  - destroy, [25](#)
  - full, [26](#)
  - insert, [26](#)
  - pool, [26](#)
  - query, [26](#)
  - queue, [27](#)
- index\_pool.c
  - ipool\_delete, [134](#)
  - ipool\_destroy, [135](#)
  - ipool\_full, [135](#)
  - ipool\_insert, [136](#)
  - ipool\_query, [136](#)
  - new\_ipool, [137](#)
- index\_pool.h
  - Index, [69](#)
  - Index\_Pool, [69](#)
  - INDEX\_POOL\_MAX\_SIZE, [69](#)
  - new\_ipool, [70](#)
- INDEX\_POOL\_MAX\_SIZE
  - index\_pool.h, [69](#)
- init\_client
  - dns\_client.c, [96](#)
  - dns\_client.h, [54](#)
- init\_config
  - config.c, [93](#)
  - config.h, [45](#)
- init\_server
  - dns\_server.c, [128](#)
  - dns\_server.h, [65](#)
- insert
  - cache\_, [11](#)
  - dns\_rr\_linklist, [22](#)
  - index\_pool, [26](#)
  - linklist\_rbtrees, [29](#)
  - query\_pool, [32](#)
- insert\_case
  - linklist\_rbtrees.c, [142](#)
- ipool
  - query\_pool, [33](#)
- ipool\_delete
  - index\_pool.c, [134](#)
- ipool\_destroy
  - index\_pool.c, [135](#)
- ipool\_full
  - index\_pool.c, [135](#)
- ipool\_insert
  - index\_pool.c, [136](#)
- ipool\_query
  - index\_pool.c, [136](#)
- key
  - rbtree\_node, [37](#)
- left
  - rbtree\_node, [37](#)
- linklist\_delete\_next
  - linklist\_rbtrees.c, [143](#)
- linklist\_insert
  - linklist\_rbtrees.c, [144](#)
- linklist\_query\_next
  - linklist\_rbtrees.c, [144](#)
- linklist\_rbtrees, [27](#)
  - insert, [29](#)
  - query, [29](#)
  - root, [29](#)
- linklist\_rbtrees.c
  - delete\_case, [140](#)
  - destroy\_node, [141](#)
  - grandparent, [141](#)
  - insert\_case, [142](#)
  - linklist\_delete\_next, [143](#)
  - linklist\_insert, [144](#)

- linklist\_query\_next, [144](#)
- new\_linklist, [145](#)
- new\_rbtrees, [146](#)
- NIL, [155](#)
- node\_init, [147](#)
- rbtree\_delete, [147](#)
- rbtree\_find, [148](#)
- rbtree\_insert, [149](#)
- rbtree\_query, [150](#)
- rotate\_left, [151](#)
- rotate\_right, [152](#)
- sibling, [152](#)
- smallest\_child, [153](#)
- uncle, [154](#)
- linklist\_rbtrees.h
  - BLACK, [74](#)
  - Color, [74](#)
  - Dns\_RR\_LinkList, [73](#)
  - new\_linklist, [74](#)
  - new\_rbtrees, [75](#)
  - Rbtrees, [73](#)
  - Rbtrees\_Node, [73](#)
  - Rbtrees\_Value, [74](#)
  - RED, [74](#)
- local\_addr
  - dns\_client.c, [101](#)
- log.h
  - log\_debug, [78](#)
  - log\_error, [78](#)
  - log\_fatal, [78](#)
  - log\_file, [79](#)
  - log\_info, [79](#)
- log\_debug
  - log.h, [78](#)
- log\_error
  - log.h, [78](#)
- log\_fatal
  - log.h, [78](#)
- log\_file
  - log.h, [79](#)
  - main.c, [157](#)
- log\_info
  - log.h, [79](#)
- LOG\_MASK
  - config.c, [94](#)
  - config.h, [46](#)
- LOG\_PATH
  - config.c, [94](#)
  - config.h, [46](#)
- loop
  - main.c, [157](#)
  - query\_pool, [33](#)
- main
  - main.c, [156](#)
- main.c
  - cache, [157](#)
  - log\_file, [157](#)
  - loop, [157](#)
  - main, [156](#)
  - qpool, [158](#)
- msg
  - dns\_query, [17](#)
- name
  - dns\_rr, [20](#)
- new\_cache
  - cache.c, [91](#)
  - cache.h, [43](#)
- new\_ipool
  - index\_pool.c, [137](#)
  - index\_pool.h, [70](#)
- new\_linklist
  - linklist\_rbtrees.c, [145](#)
  - linklist\_rbtrees.h, [74](#)
- new\_qpool
  - query\_pool.c, [159](#)
  - query\_pool.h, [82](#)
- new\_queue
  - queue.c, [166](#)
  - queue.h, [86](#)
- new\_rbtrees
  - linklist\_rbtrees.c, [146](#)
  - linklist\_rbtrees.h, [75](#)
- next
  - dns\_question, [18](#)
  - dns\_rr, [20](#)

- dns\_rr\_linklist, [22](#)
- NIL
  - linklist\_rbtrees.c, [155](#)
- node\_init
  - linklist\_rbtrees.c, [147](#)
- nscount
  - dns\_header, [13](#)
  - rbtree\_value, [39](#)
- on\_read
  - dns\_client.c, [97](#)
  - dns\_server.c, [129](#)
- on\_send
  - dns\_client.c, [98](#)
  - dns\_server.c, [130](#)
- opcode
  - dns\_header, [13](#)
- parent
  - rbtree\_node, [37](#)
- pool
  - index\_pool, [26](#)
  - query\_pool, [33](#)
- pop
  - queue, [34](#)
- prev\_id
  - dns\_query, [17](#)
  - index\_, [24](#)
- print\_dns\_header
  - dns\_print.c, [120](#)
- print\_dns\_message
  - dns\_print.c, [120](#)
  - dns\_print.h, [63](#)
- print\_dns\_question
  - dns\_print.c, [121](#)
- print\_dns\_rr
  - dns\_print.c, [122](#)
- print\_dns\_string
  - dns\_print.c, [123](#)
  - dns\_print.h, [63](#)
- print\_rr\_A
  - dns\_print.c, [123](#)
- print\_rr\_AAAA
  - dns\_print.c, [124](#)
- print\_rr\_CNAME
  - dns\_print.c, [124](#)
- print\_rr\_MX
  - dns\_print.c, [125](#)
- print\_rr\_SOA
  - dns\_print.c, [125](#)
- push
  - queue, [35](#)
- q
  - queue, [35](#)
- qclass
  - dns\_question, [18](#)
- qdcnt
  - dns\_header, [13](#)
- qname
  - dns\_question, [18](#)
- qpool
  - dns\_client.c, [101](#)
  - dns\_server.c, [133](#)
  - main.c, [158](#)
- qpool\_delete
  - query\_pool.c, [160](#)
- qpool\_finish
  - query\_pool.c, [161](#)
- qpool\_full
  - query\_pool.c, [162](#)
- qpool\_insert
  - query\_pool.c, [163](#)
- qpool\_query
  - query\_pool.c, [164](#)
- qr
  - dns\_header, [13](#)
- qtype
  - dns\_question, [18](#)
- que
  - dns\_msg, [15](#)
- query
  - cache\_, [11](#)
  - index\_pool, [26](#)
  - linklist\_rbtrees, [29](#)
- query\_next
  - dns\_rr\_linklist, [22](#)

- Query\_Pool
  - query\_pool.h, 82
- query\_pool, 30
  - cache, 31
  - count, 31
  - delete, 31
  - finish, 32
  - full, 32
  - insert, 32
  - ipool, 33
  - loop, 33
  - pool, 33
  - queue, 33
- query\_pool.c
  - new\_qpool, 159
  - qpool\_delete, 160
  - qpool\_finish, 161
  - qpool\_full, 162
  - qpool\_insert, 163
  - qpool\_query, 164
  - timeout\_cb, 164
- query\_pool.h
  - Dns\_Query, 82
  - new\_qpool, 82
  - Query\_Pool, 82
  - QUERY\_POOL\_MAX\_SIZE, 82
- QUERY\_POOL\_MAX\_SIZE
  - query\_pool.h, 82
- Queue
  - queue.h, 86
- queue, 33
  - destroy, 34
  - head, 34
  - index\_pool, 27
  - pop, 34
  - push, 35
  - q, 35
  - query\_pool, 33
  - tail, 35
- queue.c
  - new\_queue, 166
  - queue\_destroy, 166
  - queue\_pop, 167
  - queue\_push, 167
- queue.h
  - new\_queue, 86
  - Queue, 86
  - QUEUE\_MAX\_SIZE, 85
- queue\_destroy
  - queue.c, 166
- QUEUE\_MAX\_SIZE
  - queue.h, 85
- queue\_pop
  - queue.c, 167
- queue\_push
  - queue.c, 167
- ra
  - dns\_header, 13
- Rbtree
  - linklist\_rbtrees.h, 73
- rbtree\_delete
  - linklist\_rbtrees.c, 147
- rbtree\_find
  - linklist\_rbtrees.c, 148
- rbtree\_insert
  - linklist\_rbtrees.c, 149
- Rbtree\_Node
  - linklist\_rbtrees.h, 73
- rbtree\_node, 36
  - color, 37
  - key, 37
  - left, 37
  - parent, 37
  - right, 37
  - rr\_list, 37
- rbtree\_query
  - linklist\_rbtrees.c, 150
- Rbtree\_Value
  - linklist\_rbtrees.h, 74
- rbtree\_value, 38
  - ancount, 39
  - arcount, 39
  - nscount, 39
  - rr, 39
  - type, 39

- rcode
  - dns\_header, [14](#)
- rd
  - dns\_header, [14](#)
- rdata
  - dns\_rr, [20](#)
- rdlength
  - dns\_rr, [20](#)
- read\_uint16
  - dns\_parse.c, [110](#)
- read\_uint32
  - dns\_parse.c, [111](#)
- README.md, [87](#)
- recv\_addr
  - dns\_server.c, [133](#)
- RED
  - linklist\_rbtrees.h, [74](#)
- REMOTE\_HOST
  - config.c, [94](#)
  - config.h, [46](#)
- right
  - rbtree\_node, [37](#)
- root
  - linklist\_rbtrees, [29](#)
- rotate\_left
  - linklist\_rbtrees.c, [151](#)
- rotate\_right
  - linklist\_rbtrees.c, [152](#)
- rr
  - dns\_msg, [15](#)
  - rbtree\_value, [39](#)
- rr\_list
  - rbtree\_node, [37](#)
- rrname\_to\_string
  - dns\_parse.c, [111](#)
- send\_addr
  - dns\_client.c, [101](#)
- send\_to\_local
  - dns\_server.c, [132](#)
  - dns\_server.h, [66](#)
- send\_to\_remote
  - dns\_client.c, [100](#)
  - dns\_client.h, [54](#)
- server\_socket
  - dns\_server.c, [133](#)
- sibling
  - linklist\_rbtrees.c, [152](#)
- size
  - cache\_, [11](#)
- smallest\_child
  - linklist\_rbtrees.c, [153](#)
- src/cache.c, [87](#)
- src/config.c, [92](#)
- src/dns\_client.c, [95](#)
- src/dns\_parse.c, [102](#)
- src/dns\_print.c, [119](#)
- src/dns\_server.c, [126](#)
- src/index\_pool.c, [133](#)
- src/linklist\_rbtrees.c, [138](#)
- src/main.c, [155](#)
- src/query\_pool.c, [158](#)
- src/queue.c, [165](#)
- string\_to\_dnshead
  - dns\_parse.c, [112](#)
- string\_to\_dnsmsg
  - dns\_parse.c, [113](#)
  - dns\_parse.h, [60](#)
- string\_to\_dnsque
  - dns\_parse.c, [114](#)
- string\_to\_dnsrr
  - dns\_parse.c, [115](#)
- string\_to\_rrname
  - dns\_parse.c, [116](#)
- tail
  - cache\_, [12](#)
  - queue, [35](#)
- tc
  - dns\_header, [14](#)
- timeout\_cb
  - query\_pool.c, [164](#)
- timer
  - dns\_query, [17](#)
- TLDNS Relay, [1](#)
- tree

- cache\_, [12](#)
- ttl
  - dns\_rr, [20](#)
- type
  - dns\_rr, [20](#)
  - rbtree\_value, [39](#)
- uncle
  - linklist\_rbtrees.c, [154](#)
- value
  - dns\_rr\_linklist, [23](#)
- write\_uint16
  - dns\_parse.c, [117](#)
- write\_uint32
  - dns\_parse.c, [118](#)
- z
  - dns\_header, [14](#)