PythonDHCP

Generated by Doxygen 1.9.3

Chapter 1

Python DHCP Server

This is a purely Python DHCP server that does not require any additional libraries or installs other that Python 3.

This DHCP server program will assign IP addresses ten seconds after it received packets from clients. So it can be used in networks that already have a dhcp server running.

First argument is of program is testet for being configuration file fg. ./dhcp.py dhcp.conf if that file does not exists arguments are read from command line, strings must be so called double escaped "'string'" dhcp.py -broadcast_↩ address "['255.255.255.255']" -name_server "'192.168.0.1'"

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 dhcp Namespace Reference

Classes

- class ALL
- class CASEINSENSITIVE
- class CSVDatabase
- class DHCPServer
- class DHCPServerConfiguration
- class DHCPTransaction
- class GREATER
- class Host
- class HostDatabase
- class NETWORK
- class PriorityQueue
- class ThreadedTcpRequestHandler
- class ThreadedTcpServer
- class TransactionDelayWorker
- class WriteBootProtocolPacket

Functions

- def get_host_ip_addresses ()
- def ip_addresses (network, subnet_mask)
- def sorted_hosts (hosts)
- def debug_msg (msg, type)

Variables

- ALL = ALL()
- messages = TTLOrderedDict(default_ttl=86400)
- configuration = DHCPServerConfiguration()
- debug
- ip_address_lease_time
- server = DHCPServer(configuration)

6.1.1 Function Documentation

6.1.1.1 debug_msg()

```
def dhcp.debug_msg (
              msg,
              type )
```

Definition at line 716 of file dhcp.py.

6.1.1.2 get_host_ip_addresses()

```
def dhcp.get_host_ip_addresses ( )
```

```
Get IP address of current host.
```

Definition at line 59 of file dhcp.py.

6.1.1.3 ip_addresses()

```
def dhcp.ip_addresses (
              network,
              subnet_mask )
```

Definition at line 352 of file dhcp.py.

6.1.1.4 sorted_hosts()

```
def dhcp.sorted_hosts (
              hosts )
```

Definition at line 515 of file dhcp.py.

6.1.2 Variable Documentation

6.1.2.1 ALL

```
dhcp.ALL = ALL()
```

Definition at line 369 of file dhcp.py.

6.1.2.2 configuration

```
dhcp.configuration = DHCPServerConfiguration()
```

Definition at line 721 of file dhcp.py.

6.1.2.3 debug

```
dhcp.debug
```

Definition at line 722 of file dhcp.py.

6.1.2.4 ip_address_lease_time

```
dhcp.ip_address_lease_time
```

Definition at line 726 of file dhcp.py.

6.1.2.5 messages

```
dhcp.messages = TTLOrderedDict(default_ttl=86400)
```

Definition at line 714 of file dhcp.py.

6.1.2.6 server

```
dhcp.server = DHCPServer(configuration)
```

Definition at line 727 of file dhcp.py.

6.2 listener Namespace Reference

Classes

- class ReadBootProtocolPacket

Functions

- def inet_ntoaX (data)
- def inet_atonX (ips)
- def macunpack (data)
- def macpack (mac)
- def unpackbool (data)
- def packbool (bool)

Variables

- dictionary dhcp_message_types
- reversed_dhcp_message_types = dict()
- shortunpack = lambda data: (data[0] $<<$ 8) + data[1]
- shortpack = lambda i: bytes([i $>>$ 8, i & 255])
- list options
- data = base64.b16decode(b'02010600f7b41ad100000000c0a80064000000000000000000000000007c7a914bca6c00000000000
- p = ReadBootProtocolPacket(data)
- s1 = socket(type = SOCK_DGRAM)
- reads = select.select([s1], [ ], [ ], 1)[0]
- packet = ReadBootProtocolPacket($*$s.recvfrom(4096))

6.2.1 Function Documentation

6.2.1.1 inet_atonX()

```
def listener.inet_atonX (
              ips )
```

Definition at line 14 of file listener.py.

6.2.1.2 inet_ntoaX()

```
def listener.inet_ntoaX (
              data )
```

Definition at line 11 of file listener.py.

6.2.1.3 macpack()

```
def listener.macpack (
              mac )
```

Definition at line 39 of file listener.py.

6.2.1.4 macunpack()

```
def listener.macunpack (
              data )
```

Definition at line 35 of file listener.py.

6.2.1.5 packbool()

```
def listener.packbool (
              bool )
```

Definition at line 45 of file listener.py.

6.2.1.6 unpackbool()

```
def listener.unpackbool (
              data )
```

Definition at line 42 of file listener.py.

6.2.2 Variable Documentation

6.2.2.1 data

```
listener.data = base64.b16decode(b'02010600f7b41ad100000000c0a80064000000000000000000000000007c7a914bca6c0000000
```

Definition at line 224 of file listener.py.

6.2.2.2 dhcp_message_types

```
dictionary listener.dhcp_message_types
```

Initial value:
```
=  {
    1 : 'DHCPDISCOVER',
    2 : 'DHCPOFFER',
    3 : 'DHCPREQUEST',
    4 : 'DHCPDECLINE',
    5 : 'DHCPACK',
    6 : 'DHCPNAK',
    7 : 'DHCPRELEASE',
    8 : 'DHCPINFORM',
}
```

Definition at line 17 of file listener.py.

6.2.2.3 options

```
list listener.options
```

Definition at line 48 of file listener.py.

6.2.2.4 p

```
listener.p = ReadBootProtocolPacket(data)
```

Definition at line 226 of file listener.py.

6.2.2.5 packet

```
listener.packet = ReadBootProtocolPacket(*s.recvfrom(4096))
```

Definition at line 258 of file listener.py.

6.2.2.6 reads

```
listener.reads = select.select([s1], [], [], 1)[0]
```

Definition at line 256 of file listener.py.

6.2.2.7 reversed_dhcp_message_types

```
listener.reversed_dhcp_message_types = dict()
```

Definition at line 27 of file listener.py.

6.2.2.8 s1

```
listener.s1 = socket(type = SOCK_DGRAM)
```

Definition at line 249 of file listener.py.

6.2.2.9 shortpack

```
listener.shortpack = lambda i:  bytes([i >> 8, i & 255])
```

Definition at line 32 of file listener.py.

6.2.2.10 shortunpack

```
listener.shortunpack = lambda data:  (data[0] << 8) + data[1]
```

Definition at line 31 of file listener.py.

6.3 ttldict Namespace Reference

Classes

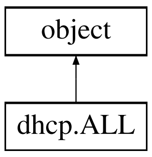
- class TTLOrderedDict

Chapter 7

Class Documentation

7.1 dhcp.ALL Class Reference

Inheritance diagram for dhcp.ALL:



Public Member Functions

- def __eq__ (self, other)
- def __repr__ (self)

7.1.1 Detailed Description

```
Comparator class
```

Definition at line 361 of file dhcp.py.

7.1.2 Member Function Documentation

7.1.2.1 __eq__()

```
def dhcp.ALL.__eq__ (
              self,
              other )
```

Definition at line 364 of file dhcp.py.

7.1.2.2 __repr__()

```
def dhcp.ALL.__repr__ (
              self )
```

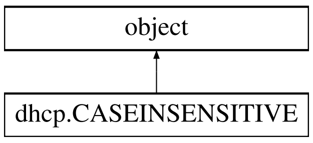
Definition at line 366 of file dhcp.py.

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

- dhcp.py

7.2 dhcp.CASEINSENSITIVE Class Reference

Inheritance diagram for dhcp.CASEINSENSITIVE:



Public Member Functions

- def __init__ (self, s)
- def __eq__ (self, other)

Public Attributes

- s

7.2.1 Detailed Description

```
Comparator class
```

Definition at line 391 of file dhcp.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def dhcp.CASEINSENSITIVE.__init__ (
              self,
              s )
```

Definition at line 394 of file dhcp.py.

7.2.3 Member Function Documentation

7.2.3.1 __eq__()

```
def dhcp.CASEINSENSITIVE.__eq__ (
              self,
              other )
```

Definition at line 396 of file dhcp.py.

7.2.4 Member Data Documentation

7.2.4.1 s

```
dhcp.CASEINSENSITIVE.s
```

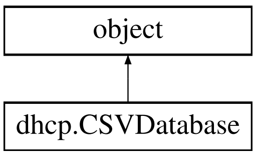
Definition at line 395 of file dhcp.py.

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

- dhcp.py

7.3 dhcp.CSVDatabase Class Reference

Inheritance diagram for dhcp.CSVDatabase:



Public Member Functions

- def __init__ (self, file_name)
- def file (self, mode='r')
- def get (self, pattern)
- def add (self, line)
- def delete (self, pattern)
- def all (self)

Public Attributes

- file_name

Static Public Attributes

- string delimiter = ';'

7.3.1 Detailed Description

```
Class handling CSV file database to keep host definitions
```

Definition at line 399 of file dhcp.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def dhcp.CSVDatabase.__init__ (
              self,
              file_name )
```

```
Construct new CSV database with storage in file_name
```

Definition at line 404 of file dhcp.py.

7.3.3 Member Function Documentation

7.3.3.1 add()

```
def dhcp.CSVDatabase.add (
              self,
              line )
```

```
Add host entry to CSV file
```

Definition at line 421 of file dhcp.py.

7.3.3.2 all()

```
def dhcp.CSVDatabase.all (
              self )
```

```
Get all entries from CSV file
```

Definition at line 437 of file dhcp.py.

7.3.3.3 delete()

```
def dhcp.CSVDatabase.delete (
              self,
              pattern )
```

```
Delete host entry from CSV file
```

Definition at line 427 of file dhcp.py.

7.3.3.4 file()

```
def dhcp.CSVDatabase.file (
              self,
              mode = 'r' )
```

```
Open CSV file with selected mode
```

Definition at line 410 of file dhcp.py.

7.3.3.5 get()

```
def dhcp.CSVDatabase.get (
              self,
              pattern )
```

```
Get CSV entry representing host(MAC) and lease(IP)
```

Definition at line 415 of file dhcp.py.

7.3.4 Member Data Documentation

7.3.4.1 delimiter

```
string dhcp.CSVDatabase.delimiter = ';'  [static]
```

Definition at line 402 of file dhcp.py.

7.3.4.2 file_name

```
dhcp.CSVDatabase.file_name
```

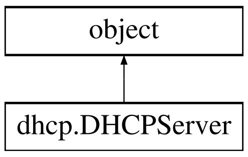
Definition at line 407 of file dhcp.py.

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

- dhcp.py

7.4 dhcp.DHCPServer Class Reference

Inheritance diagram for dhcp.DHCPServer:



Public Member Functions

- def __init__ (self, configuration=None)
- def close (self)
- def update (self, timeout=0)
- def received (self, packet)
- def client_has_chosen (self, packet)
- def is_valid_client_address (self, address)
- def get_ip_address (self, packet)
- def server_identifiers (self)
- def broadcast (self, packet)
- def run (self)
- def run_in_thread (self)
- def debug_clients (self)
- def get_all_hosts (self)
- def get_current_hosts (self)

Public Attributes

- configuration
- socket
- delay_worker
- closed
- transactions
- hosts
- time_started

7.4.1 Detailed Description

Definition at line 520 of file dhcp.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def dhcp.DHCPServer.__init__ (
              self,
              configuration = None )
```

Definition at line 522 of file dhcp.py.

7.4.3 Member Function Documentation

7.4.3.1 broadcast()

```
def dhcp.DHCPServer.broadcast (
              self,
              packet )
```

Definition at line 619 of file dhcp.py.

7.4.3.2 client_has_chosen()

```
def dhcp.DHCPServer.client_has_chosen (
              self,
              packet )
```

Definition at line 566 of file dhcp.py.

7.4.3.3 close()

```
def dhcp.DHCPServer.close (
              self )
```

Definition at line 536 of file dhcp.py.

7.4.3.4 debug_clients()

```
def dhcp.DHCPServer.debug_clients (
              self )
```

Definition at line 648 of file dhcp.py.

7.4.3.5 get_all_hosts()

```
def dhcp.DHCPServer.get_all_hosts (
              self )
```

Definition at line 654 of file dhcp.py.

7.4.3.6 get_current_hosts()

```
def dhcp.DHCPServer.get_current_hosts (
              self )
```

Definition at line 657 of file dhcp.py.

7.4.3.7 get_ip_address()

```
def dhcp.DHCPServer.get_ip_address (
              self,
              packet )
```

Definition at line 581 of file dhcp.py.

7.4.3.8 is_valid_client_address()

```
def dhcp.DHCPServer.is_valid_client_address (
              self,
              address )
```

Definition at line 573 of file dhcp.py.

7.4.3.9 received()

```
def dhcp.DHCPServer.received (
              self,
              packet )
```

Definition at line 562 of file dhcp.py.

7.4.3.10 run()

```
def dhcp.DHCPServer.run (
              self )
```

Definition at line 634 of file dhcp.py.

7.4.3.11 run_in_thread()

```
def dhcp.DHCPServer.run_in_thread (
              self )
```

Definition at line 643 of file dhcp.py.

7.4.3.12 server_identifiers()

```
def dhcp.DHCPServer.server_identifiers (
              self )
```

Definition at line 616 of file dhcp.py.

7.4.3.13 update()

```
def dhcp.DHCPServer.update (
              self,
              timeout = 0 )
```

Definition at line 543 of file dhcp.py.

7.4.4 Member Data Documentation

7.4.4.1 closed

```
dhcp.DHCPServer.closed
```

Definition at line 531 of file dhcp.py.

7.4.4.2 configuration

```
dhcp.DHCPServer.configuration
```

Definition at line 526 of file dhcp.py.

7.4.4.3 delay_worker

```
dhcp.DHCPServer.delay_worker
```

Definition at line 530 of file dhcp.py.

7.4.4.4 hosts

```
dhcp.DHCPServer.hosts
```

Definition at line 533 of file dhcp.py.

7.4.4.5 socket

```
dhcp.DHCPServer.socket
```

Definition at line 527 of file dhcp.py.

7.4.4.6 time_started

```
dhcp.DHCPServer.time_started
```

Definition at line 534 of file dhcp.py.

7.4.4.7 transactions

```
dhcp.DHCPServer.transactions
```

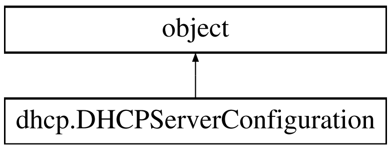
Definition at line 532 of file dhcp.py.

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

- dhcp.py

7.5 dhcp.DHCPServerConfiguration Class Reference

Inheritance diagram for dhcp.DHCPServerConfiguration:



Public Member Functions

- def load (self, file)
- def adjust_if_this_computer_is_a_router (self)
- def all_ip_addresses (self)
- def network_filter (self)

Public Attributes

- network
- broadcast_address

Static Public Attributes

- int dhcp_offer_after_seconds = 10
- int dhcp_acknowledge_after_seconds = 10
- int length_of_transaction = 40
- string network = '192.168.173.0'
- string broadcast_address = '255.255.255.255'
- string subnet_mask = '255.255.255.0'
- router = None
- int ip_address_lease_time = 300
- domain_name_server = None
- string host_file = 'hosts.csv'
- debug = lambda ∗args, ∗∗kw: None

7.5.1 Detailed Description

```
Class to load DHCP server configuration from file or command line
```

Definition at line 295 of file dhcp.py.

7.5.2 Member Function Documentation

7.5.2.1 adjust_if_this_computer_is_a_router()

```
def dhcp.DHCPServerConfiguration.adjust_if_this_computer_is_a_router (
              self )
```

```
Automatically adjust some DHCP configuration parameters if this computer is router
```

Definition at line 329 of file dhcp.py.

7.5.2.2 all_ip_addresses()

```
def dhcp.DHCPServerConfiguration.all_ip_addresses (
              self )
```

Definition at line 343 of file dhcp.py.

7.5.2.3 load()

```
def dhcp.DHCPServerConfiguration.load (
              self,
              file )
```

```
Load configuration from file using exec to parse file as object dictionary
or get ALL command line arguments and change them using regexp to file layout
and treat as file
```

Definition at line 314 of file dhcp.py.

7.5.2.4 network_filter()

```
def dhcp.DHCPServerConfiguration.network_filter (
              self )
```

Definition at line 349 of file dhcp.py.

7.5.3 Member Data Documentation

7.5.3.1 broadcast_address [1/2]

```
string dhcp.DHCPServerConfiguration.broadcast_address = '255.255.255.255'  [static]
```

Definition at line 303 of file dhcp.py.

7.5.3.2 broadcast_address [2/2]

```
dhcp.DHCPServerConfiguration.broadcast_address
```

Definition at line 338 of file dhcp.py.

7.5.3.3 debug

```
dhcp.DHCPServerConfiguration.debug = lambda *args, **kw:  None  [static]
```

Definition at line 312 of file dhcp.py.

7.5.3.4 dhcp_acknowledge_after_seconds

```
int dhcp.DHCPServerConfiguration.dhcp_acknowledge_after_seconds = 10  [static]
```

Definition at line 299 of file dhcp.py.

7.5.3.5 dhcp_offer_after_seconds

```
int dhcp.DHCPServerConfiguration.dhcp_offer_after_seconds = 10  [static]
```

Definition at line 298 of file dhcp.py.

7.5.3.6 domain_name_server

```
dhcp.DHCPServerConfiguration.domain_name_server = None  [static]
```

Definition at line 308 of file dhcp.py.

7.5.3.7 host_file

```
string dhcp.DHCPServerConfiguration.host_file = 'hosts.csv'  [static]
```

Definition at line 310 of file dhcp.py.

7.5.3.8 ip_address_lease_time

```
int dhcp.DHCPServerConfiguration.ip_address_lease_time = 300  [static]
```

Definition at line 307 of file dhcp.py.

7.5.3.9 length_of_transaction

```
int dhcp.DHCPServerConfiguration.length_of_transaction = 40  [static]
```

Definition at line 300 of file dhcp.py.

7.5.3.10 network `[1/2]`

```
string dhcp.DHCPServerConfiguration.network = '192.168.173.0'  [static]
```

Definition at line 302 of file dhcp.py.

7.5.3.11 network `[2/2]`

```
dhcp.DHCPServerConfiguration.network
```

Definition at line 337 of file dhcp.py.

7.5.3.12 router

```
dhcp.DHCPServerConfiguration.router = None  [static]
```

Definition at line 305 of file dhcp.py.

7.5.3.13 subnet_mask

```
string dhcp.DHCPServerConfiguration.subnet_mask = '255.255.255.0'  [static]
```

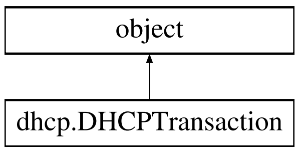
Definition at line 304 of file dhcp.py.

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

- dhcp.py

7.6 dhcp.DHCPTransaction Class Reference

Inheritance diagram for dhcp.DHCPTransaction:



Public Member Functions

- def __init__ (self, server)
- def is_done (self)
- def close (self)
- def receive (self, packet)
- def received_dhcp_discover (self, discovery)
- def send_offer (self, discovery)
- def received_dhcp_request (self, request)
- def acknowledge (self, request)
- def received_dhcp_inform (self, inform)

Public Attributes

- server
- configuration
- packets
- done_time
- done
- do_after
- debug

7.6.1 Detailed Description

```
Class representing DHCP Transaction
```

Definition at line 194 of file dhcp.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def dhcp.DHCPTransaction.__init__ (
              self,
              server )
```

```
Contructor of new transaction
```

Definition at line 197 of file dhcp.py.

7.6.3 Member Function Documentation

7.6.3.1 acknowledge()

```
def dhcp.DHCPTransaction.acknowledge (
              self,
              request )
```

```
Method used to handle DHCP Acknowledge packet
```

Definition at line 270 of file dhcp.py.

7.6.3.2 close()

```
def dhcp.DHCPTransaction.close (
              self )
```

```
Close transaction
```

Definition at line 213 of file dhcp.py.

7.6.3.3 is_done()

```
def dhcp.DHCPTransaction.is_done (
              self )
```

```
Check if transaction is done
```

Definition at line 208 of file dhcp.py.

7.6.3.4 receive()

```
def dhcp.DHCPTransaction.receive (
              self,
              packet )
```

```
Receive DHCP UDP packet check it's type and call a proper callback
```

Definition at line 218 of file dhcp.py.

7.6.3.5 received_dhcp_discover()

```
def dhcp.DHCPTransaction.received_dhcp_discover (
              self,
              discovery )
```

```
Method used to handle DHCP Discover packet
```

Definition at line 234 of file dhcp.py.

7.6.3.6 received_dhcp_inform()

```
def dhcp.DHCPTransaction.received_dhcp_inform (
              self,
              inform )
```

```
Method used to handle DHCP Inform packet
```

Definition at line 288 of file dhcp.py.

7.6.3.7 received_dhcp_request()

```
def dhcp.DHCPTransaction.received_dhcp_request (
              self,
              request )
```

```
Method used to handle DHCP Request packet
```

Definition at line 261 of file dhcp.py.

7.6.3.8 send_offer()

```
def dhcp.DHCPTransaction.send_offer (
              self,
              discovery )
```

```
Method used to send DHCP offer packet
```

Definition at line 241 of file dhcp.py.

7.6.4 Member Data Documentation

7.6.4.1 configuration

`dhcp.DHCPTransaction.configuration`

Definition at line 201 of file dhcp.py.

7.6.4.2 debug

`dhcp.DHCPTransaction.debug`

Definition at line 206 of file dhcp.py.

7.6.4.3 do_after

`dhcp.DHCPTransaction.do_after`

Definition at line 205 of file dhcp.py.

7.6.4.4 done

`dhcp.DHCPTransaction.done`

Definition at line 204 of file dhcp.py.

7.6.4.5 done_time

`dhcp.DHCPTransaction.done_time`

Definition at line 203 of file dhcp.py.

7.6.4.6 packets

```
dhcp.DHCPTransaction.packets
```

Definition at line 202 of file dhcp.py.

7.6.4.7 server

```
dhcp.DHCPTransaction.server
```

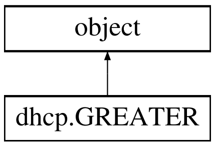
Definition at line 200 of file dhcp.py.

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

- dhcp.py

7.7 dhcp.GREATER Class Reference

Inheritance diagram for dhcp.GREATER:



Public Member Functions

- def __init__ (self, value)
- def __eq__ (self, other)

Public Attributes

- value

7.7.1 Detailed Description

```
Comparator class
```

Definition at line 371 of file dhcp.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
def dhcp.GREATER.__init__ (
              self,
              value )
```

Definition at line 374 of file dhcp.py.

7.7.3 Member Function Documentation

7.7.3.1 __eq__()

```
def dhcp.GREATER.__eq__ (
              self,
              other )
```

Definition at line 376 of file dhcp.py.

7.7.4 Member Data Documentation

7.7.4.1 value

```
dhcp.GREATER.value
```

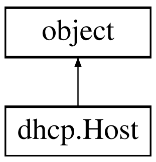
Definition at line 375 of file dhcp.py.

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

- dhcp.py

7.8 dhcp.Host Class Reference

Inheritance diagram for dhcp.Host:



Public Member Functions

- def __init__ (self, mac, ip, hostname, last_used)
- def from_tuple (cls, line)
- def from_packet (cls, packet)
- def to_tuple (self)
- def to_pattern (self)
- def __hash__ (self)
- def __eq__ (self, other)
- def has_valid_ip (self)

Static Public Member Functions

- def get_pattern (mac=ALL, ip=ALL, hostname=ALL, last_used=ALL)

Public Attributes

- mac
- ip
- hostname
- last_used

7.8.1 Detailed Description

```
Class representing host with MAC address, IP, hostname if available and last used timestamp
```

Definition at line 443 of file dhcp.py.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def dhcp.Host.__init__ (
              self,
              mac,
              ip,
              hostname,
              last_used )
```

Definition at line 446 of file dhcp.py.

7.8.3 Member Function Documentation

7.8.3.1 __eq__()

```
def dhcp.Host.__eq__ (
              self,
              other )
```

Definition at line 482 of file dhcp.py.

7.8.3.2 __hash__()

```
def dhcp.Host.__hash__ (
              self )
```

Definition at line 479 of file dhcp.py.

7.8.3.3 from_packet()

```
def dhcp.Host.from_packet (
              cls,
              packet )
```

Definition at line 459 of file dhcp.py.

7.8.3.4 from_tuple()

```
def dhcp.Host.from_tuple (
              cls,
              line )
```

Definition at line 453 of file dhcp.py.

7.8.3.5 get_pattern()

```
def dhcp.Host.get_pattern (
              mac = ALL,
              ip = ALL,
              hostname = ALL,
              last_used = ALL )   [static]
```

Definition at line 466 of file dhcp.py.

7.8.3.6 has_valid_ip()

```
def dhcp.Host.has_valid_ip (
              self )
```

```
Check if host has valid IP address
```

Definition at line 485 of file dhcp.py.

7.8.3.7 to_pattern()

```
def dhcp.Host.to_pattern (
              self )
```

```
Convert host to pattern
```

Definition at line 474 of file dhcp.py.

7.8.3.8 to_tuple()

```
def dhcp.Host.to_tuple (
              self )
```

```
Convert host to tuple
```

Definition at line 469 of file dhcp.py.

7.8.4 Member Data Documentation

7.8.4.1 hostname

```
dhcp.Host.hostname
```

Definition at line 449 of file dhcp.py.

7.8.4.2 ip

```
dhcp.Host.ip
```

Definition at line 448 of file dhcp.py.

7.8.4.3 last_used

```
dhcp.Host.last_used
```

Definition at line 450 of file dhcp.py.

7.8.4.4 mac

```
dhcp.Host.mac
```

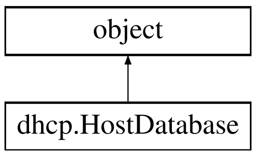
Definition at line 447 of file dhcp.py.

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

- dhcp.py

7.9 dhcp.HostDatabase Class Reference

Inheritance diagram for dhcp.HostDatabase:



Public Member Functions

- def __init__ (self, file_name)
- def get (self, **kw)
- def add (self, host)
- def delete (self, host=None, **kw)
- def all (self)
- def replace (self, host)

Public Attributes

- db

7.9.1 Detailed Description

Definition at line 490 of file dhcp.py.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 __init__()

```
def dhcp.HostDatabase.__init__ (
              self,
              file_name )
```

Definition at line 491 of file dhcp.py.

7.9.3 Member Function Documentation

7.9.3.1 add()

```
def dhcp.HostDatabase.add (
              self,
              host )
```

Definition at line 498 of file dhcp.py.

7.9.3.2 all()

```
def dhcp.HostDatabase.all (
              self )
```

Definition at line 508 of file dhcp.py.

7.9.3.3 delete()

```
def dhcp.HostDatabase.delete (
              self,
              host = None,
             ** kw )
```

Definition at line 501 of file dhcp.py.

7.9.3.4 get()

```
def dhcp.HostDatabase.get (
              self,
             ** kw )
```

Definition at line 494 of file dhcp.py.

7.9.3.5 replace()

```
def dhcp.HostDatabase.replace (
              self,
              host )
```

Definition at line 511 of file dhcp.py.

7.9.4 Member Data Documentation

7.9.4.1 db

```
dhcp.HostDatabase.db
```

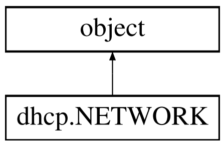
Definition at line 492 of file dhcp.py.

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

- dhcp.py

7.10 dhcp.NETWORK Class Reference

Inheritance diagram for dhcp.NETWORK:



Public Member Functions

- def __init__ (self, network, subnet_mask)
- def __eq__ (self, other)

Public Attributes

- subnet_mask
- network

7.10.1 Detailed Description

```
Comparator class to check if address within same network
```

Definition at line 379 of file dhcp.py.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

```
def dhcp.NETWORK.__init__ (
              self,
              network,
              subnet_mask )
```

Definition at line 382 of file dhcp.py.

7.10.3 Member Function Documentation

7.10.3.1 __eq__()

```
def dhcp.NETWORK.__eq__ (
              self,
              other )
```

Definition at line 385 of file dhcp.py.

7.10.4 Member Data Documentation

7.10.4.1 network

```
dhcp.NETWORK.network
```

Definition at line 384 of file dhcp.py.

7.10.4.2 subnet_mask

```
dhcp.NETWORK.subnet_mask
```

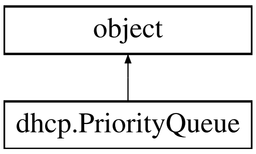
Definition at line 383 of file dhcp.py.

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

- dhcp.py

7.11 dhcp.PriorityQueue Class Reference

Inheritance diagram for dhcp.PriorityQueue:



Public Member Functions

- def __init__ (self)
- def put (self, item)
- def get (self)
- def qsize (self)

7.11.1 Detailed Description

```
This class contains Heapq for more information:
https://docs.python.org/3/library/heapq.html
```

Definition at line 64 of file dhcp.py.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 __init__()

```
def dhcp.PriorityQueue.__init__ (
              self )
```

Definition at line 68 of file dhcp.py.

7.11.3 Member Function Documentation

7.11.3.1 get()

```
def dhcp.PriorityQueue.get (
              self )
```

Definition at line 76 of file dhcp.py.

7.11.3.2 put()

```
def dhcp.PriorityQueue.put (
              self,
              item )
```

Definition at line 72 of file dhcp.py.

7.11.3.3 qsize()

```
def dhcp.PriorityQueue.qsize (
              self )
```

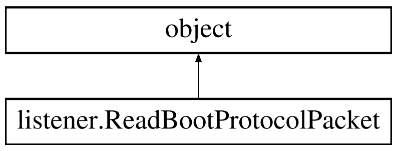
Definition at line 79 of file dhcp.py.

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

- dhcp.py

7.12 listener.ReadBootProtocolPacket Class Reference

Inheritance diagram for listener.ReadBootProtocolPacket:



Public Member Functions

- def __init__ (self, data, address=('0.0.0.0', 0))
- def __getitem__ (self, key)
- def __contains__ (self, key)
- def formatted_named_options (self)
- def __str__ (self)
- def __gt__ (self, other)

Public Attributes

- data
- address
- host
- port
- message_type
- OP
- hardware_type
- HTYPE
- hardware_address_length
- HLEN
- hops
- HOPS
- XID
- transaction_id
- seconds_elapsed
- SECS
- bootp_flags
- FLAGS
- client_ip_address
- CIADDR
- your_ip_address
- YIADDR
- next_server_ip_address
- SIADDR
- relay_agent_ip_address
- GIADDR
- client_mac_address
- CHADDR
- magic_cookie
- options
- named_options

7.12.1 Detailed Description

Definition at line 144 of file listener.py.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 __init__()

```
def listener.ReadBootProtocolPacket.__init__ (
              self,
              data,
              address = ('0.0.0.0', 0) )
```

Definition at line 152 of file listener.py.

7.12.3 Member Function Documentation

7.12.3.1 __contains__()

```
def listener.ReadBootProtocolPacket.__contains__ (
              self,
              key )
```

Definition at line 205 of file listener.py.

7.12.3.2 __getitem__()

```
def listener.ReadBootProtocolPacket.__getitem__ (
              self,
              key )
```

Definition at line 201 of file listener.py.

7.12.3.3 __gt__()

```
def listener.ReadBootProtocolPacket.__gt__ (
              self,
              other )
```

Definition at line 221 of file listener.py.

7.12.3.4 __str__()

```
def listener.ReadBootProtocolPacket.__str__ (
              self )
```

Definition at line 212 of file listener.py.

7.12.3.5 formatted_named_options()

```
def listener.ReadBootProtocolPacket.formatted_named_options (
              self )
```

Definition at line 209 of file listener.py.

7.12.4 Member Data Documentation

7.12.4.1 address

```
listener.ReadBootProtocolPacket.address
```

Definition at line 154 of file listener.py.

7.12.4.2 bootp_flags

```
listener.ReadBootProtocolPacket.bootp_flags
```

Definition at line 168 of file listener.py.

7.12.4.3 CHADDR

```
listener.ReadBootProtocolPacket.CHADDR
```

Definition at line 175 of file listener.py.

7.12.4.4 CIADDR

```
listener.ReadBootProtocolPacket.CIADDR
```

Definition at line 170 of file listener.py.

7.12.4.5 client_ip_address

```
listener.ReadBootProtocolPacket.client_ip_address
```

Definition at line 170 of file listener.py.

7.12.4.6 client_mac_address

```
listener.ReadBootProtocolPacket.client_mac_address
```

Definition at line 175 of file listener.py.

7.12.4.7 data

```
listener.ReadBootProtocolPacket.data
```

Definition at line 153 of file listener.py.

7.12.4.8 FLAGS

```
listener.ReadBootProtocolPacket.FLAGS
```

Definition at line 168 of file listener.py.

7.12.4.9 GIADDR

```
listener.ReadBootProtocolPacket.GIADDR
```

Definition at line 173 of file listener.py.

7.12.4.10 hardware_address_length

```
listener.ReadBootProtocolPacket.hardware_address_length
```

Definition at line 162 of file listener.py.

7.12.4.11 hardware_type

```
listener.ReadBootProtocolPacket.hardware_type
```

Definition at line 161 of file listener.py.

7.12.4.12 HLEN

```
listener.ReadBootProtocolPacket.HLEN
```

Definition at line 162 of file listener.py.

7.12.4.13 hops

```
listener.ReadBootProtocolPacket.hops
```

Definition at line 163 of file listener.py.

7.12.4.14 HOPS

```
listener.ReadBootProtocolPacket.HOPS
```

Definition at line 163 of file listener.py.

7.12.4.15 host

```
listener.ReadBootProtocolPacket.host
```

Definition at line 155 of file listener.py.

7.12.4.16 HTYPE

```
listener.ReadBootProtocolPacket.HTYPE
```

Definition at line 161 of file listener.py.

7.12.4.17 magic_cookie

```
listener.ReadBootProtocolPacket.magic_cookie
```

Definition at line 177 of file listener.py.

7.12.4.18 message_type

```
listener.ReadBootProtocolPacket.message_type
```

Definition at line 160 of file listener.py.

7.12.4.19 named_options

```
listener.ReadBootProtocolPacket.named_options
```

Definition at line 179 of file listener.py.

7.12.4.20 next_server_ip_address

```
listener.ReadBootProtocolPacket.next_server_ip_address
```

Definition at line 172 of file listener.py.

7.12.4.21 OP

```
listener.ReadBootProtocolPacket.OP
```

Definition at line 160 of file listener.py.

7.12.4.22 options

```
listener.ReadBootProtocolPacket.options
```

Definition at line 178 of file listener.py.

7.12.4.23 port

```
listener.ReadBootProtocolPacket.port
```

Definition at line 156 of file listener.py.

7.12.4.24 relay_agent_ip_address

```
listener.ReadBootProtocolPacket.relay_agent_ip_address
```

Definition at line 173 of file listener.py.

7.12.4.25 seconds_elapsed

```
listener.ReadBootProtocolPacket.seconds_elapsed
```

Definition at line 167 of file listener.py.

7.12.4.26 SECS

```
listener.ReadBootProtocolPacket.SECS
```

Definition at line 167 of file listener.py.

7.12.4.27 SIADDR

```
listener.ReadBootProtocolPacket.SIADDR
```

Definition at line 172 of file listener.py.

7.12.4.28 transaction_id

```
listener.ReadBootProtocolPacket.transaction_id
```

Definition at line 165 of file listener.py.

7.12.4.29 XID

```
listener.ReadBootProtocolPacket.XID
```

Definition at line 165 of file listener.py.

7.12.4.30 YIADDR

```
listener.ReadBootProtocolPacket.YIADDR
```

Definition at line 171 of file listener.py.

7.12.4.31 your_ip_address

```
listener.ReadBootProtocolPacket.your_ip_address
```

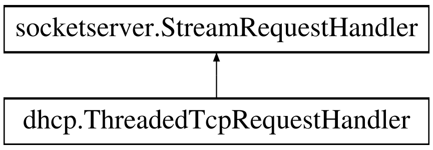
Definition at line 171 of file listener.py.

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

- listener.py

7.13 dhcp.ThreadedTcpRequestHandler Class Reference

Inheritance diagram for dhcp.ThreadedTcpRequestHandler:


socketserver.StreamRequestHandler
dhcp.ThreadedTcpRequestHandler

Public Member Functions

- def handle (self)

7.13.1 Detailed Description

```
Control socket client connection handler
```

Definition at line 660 of file dhcp.py.

7.13.2 Member Function Documentation

7.13.2.1 handle()

```
def dhcp.ThreadedTcpRequestHandler.handle (
              self )
```

```
Method used to handle client connection parsing commands and giving response to them
```

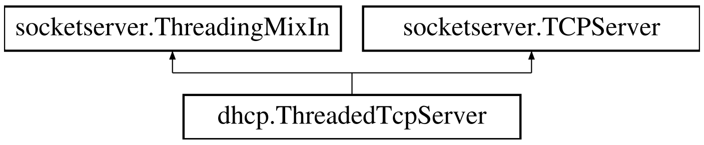
Definition at line 663 of file dhcp.py.

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

- dhcp.py

7.14 dhcp.ThreadedTcpServer Class Reference

Inheritance diagram for dhcp.ThreadedTcpServer:



Public Member Functions

- def setEvents (self, data)
- def setHosts (self, data)
- def setConfiguration (self, data)

Public Attributes

- events
- hosts
- configuration

7.14.1 Detailed Description

```
DHCP server control interface TCP server
```

Definition at line 694 of file dhcp.py.

7.14.2 Member Function Documentation

7.14.2.1 setConfiguration()

```
def dhcp.ThreadedTcpServer.setConfiguration (
              self,
              data )
```

```
Set DHCP UDP global configuration reference
```

Definition at line 707 of file dhcp.py.

7.14.2.2 setEvents()

```
def dhcp.ThreadedTcpServer.setEvents (
              self,
              data )
```

```
Set DHCP events dictionary reference
```

Definition at line 697 of file dhcp.py.

7.14.2.3 setHosts()

```
def dhcp.ThreadedTcpServer.setHosts (
              self,
              data )
```

```
Set DHCP host database with active leases reference
```

Definition at line 702 of file dhcp.py.

7.14.3 Member Data Documentation

7.14.3.1 configuration

```
dhcp.ThreadedTcpServer.configuration
```

Definition at line 710 of file dhcp.py.

7.14.3.2 events

```
dhcp.ThreadedTcpServer.events
```

Definition at line 700 of file dhcp.py.

7.14.3.3 hosts

```
dhcp.ThreadedTcpServer.hosts
```

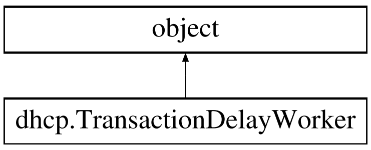
Definition at line 705 of file dhcp.py.

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

- dhcp.py

7.15 dhcp.TransactionDelayWorker Class Reference

Inheritance diagram for dhcp.TransactionDelayWorker:



Public Member Functions

- def __init__ (self)
- def do_after (self, seconds, func, args=(), kw={})
- def close (self)

Public Attributes

- closed
- queue
- thread

7.15.1 Detailed Description

```
Class used to delay response to DHCP client
```

Definition at line 20 of file dhcp.py.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 __init__()

```
def dhcp.TransactionDelayWorker.__init__ (
              self )
```

```
class constructor internally using priority queue where priority is time
```

Definition at line 23 of file dhcp.py.

7.15.3 Member Function Documentation

7.15.3.1 close()

```
def dhcp.TransactionDelayWorker.close (
              self )
```

```
Method used to stop worker
```

Definition at line 54 of file dhcp.py.

7.15.3.2 do_after()

```
def dhcp.TransactionDelayWorker.do_after (
              self,
              seconds,
              func,
              args = (),
              kw = {} )
```

```
Add to queue function which should be called after certain time
specified by seconds, args, kw are arguments
```

Definition at line 48 of file dhcp.py.

7.15.4 Member Data Documentation

7.15.4.1 closed

```
dhcp.TransactionDelayWorker.closed
```

Definition at line 26 of file dhcp.py.

7.15.4.2 queue

```
dhcp.TransactionDelayWorker.queue
```

Definition at line 27 of file dhcp.py.

7.15.4.3 thread

```
dhcp.TransactionDelayWorker.thread
```

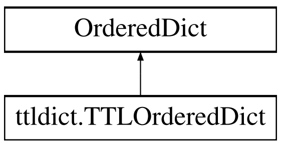
Definition at line 28 of file dhcp.py.

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

- dhcp.py

7.16 ttldict.TTLOrderedDict Class Reference

Inheritance diagram for ttldict.TTLOrderedDict:



Public Member Functions

- def __init__ (self, default_ttl, ∗args, ∗∗kwargs)
- def __repr__ (self)
- def __len__ (self)
- def set_ttl (self, key, ttl, now=None)
- def get_ttl (self, key, now=None)
- def expire_at (self, key, timestamp)
- def is_expired (self, key, now=None)
- def __iter__ (self)
- def __setitem__ (self, key, value)
- def __delitem__ (self, key)
- def __getitem__ (self, key)
- def keys (self)
- def items (self)
- def values (self)
- def get (self, key, default=None)

7.16.1 Detailed Description

```
OrderedDict with TTL
Extra args and kwargs are passed to initial .update() call
```

Definition at line 8 of file ttldict.py.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 __init__()

```
def ttldict.TTLOrderedDict.__init__ (
              self,
              default_ttl,
             * args,
             ** kwargs )
```

```
Be warned, if you use this with Python versions earlier than 3.6
when passing **kwargs order is not preseverd.
```

Definition at line 13 of file ttldict.py.

7.16.3 Member Function Documentation

7.16.3.1 __delitem__()

```
def ttldict.TTLOrderedDict.__delitem__ (
              self,
              key )
```

Definition at line 89 of file ttldict.py.

7.16.3.2 __getitem__()

```
def ttldict.TTLOrderedDict.__getitem__ (
              self,
              key )
```

Definition at line 93 of file ttldict.py.

7.16.3.3 __iter__()

```
def ttldict.TTLOrderedDict.__iter__ (
              self )
```

```
Yield only non expired keys, without purging the expired ones
```

Definition at line 72 of file ttldict.py.

7.16.3.4 __len__()

```
def ttldict.TTLOrderedDict.__len__ (
              self )
```

Definition at line 28 of file ttldict.py.

7.16.3.5 __repr__()

```
def ttldict.TTLOrderedDict.__repr__ (
              self )
```

Definition at line 24 of file ttldict.py.

7.16.3.6 __setitem__()

```
def ttldict.TTLOrderedDict.__setitem__ (
              self,
              key,
              value )
```

Definition at line 81 of file ttldict.py.

7.16.3.7 expire_at()

```
def ttldict.TTLOrderedDict.expire_at (
              self,
              key,
              timestamp )
```

```
Set the key expire timestamp
```

Definition at line 49 of file ttldict.py.

7.16.3.8 get()

```
def ttldict.TTLOrderedDict.get (
              self,
              key,
              default = None )
```

Definition at line 118 of file ttldict.py.

7.16.3.9 get_ttl()

```
def ttldict.TTLOrderedDict.get_ttl (
              self,
              key,
              now = None )
```

```
Return remaining TTL for a key
```

Definition at line 41 of file ttldict.py.

7.16.3.10 is_expired()

```
def ttldict.TTLOrderedDict.is_expired (
              self,
              key,
              now = None )
```

```
 Check if key has expired, and return it if so
```

Definition at line 55 of file ttldict.py.

7.16.3.11 items()

```
def ttldict.TTLOrderedDict.items (
              self )
```

Definition at line 106 of file ttldict.py.

7.16.3.12 keys()

```
def ttldict.TTLOrderedDict.keys (
              self )
```

Definition at line 101 of file ttldict.py.

7.16.3.13 set_ttl()

```
def ttldict.TTLOrderedDict.set_ttl (
              self,
              key,
              ttl,
              now = None )
```

```
Set TTL for the given key
```

Definition at line 33 of file ttldict.py.

7.16.3.14 values()

```
def ttldict.TTLOrderedDict.values (
              self )
```

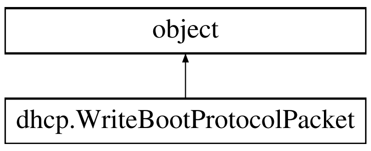
Definition at line 112 of file ttldict.py.

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

- ttldict.py

7.17 dhcp.WriteBootProtocolPacket Class Reference

Inheritance diagram for dhcp.WriteBootProtocolPacket:



Public Member Functions

- def __init__ (self, configuration)
- def to_bytes (self)
- def get_option (self, option)
- def options (self)
- def __str__ (self)

Static Public Attributes

- int message_type = 2
- int hardware_type = 1
- int hardware_address_length = 6
- int hops = 0
- transaction_id = None
- int seconds_elapsed = 0
- int bootp_flags = 0
- string client_ip_address = '0.0.0.0'
- string your_ip_address = '0.0.0.0'
- string next_server_ip_address = '0.0.0.0'
- string relay_agent_ip_address = '0.0.0.0'
- client_mac_address = None
- string magic_cookie = '99.130.83.99'
- list parameter_order = [ ]

7.17.1 Detailed Description

```
DHCP protocol datagram serializer
This class serializes UDP DHCP packet, instance is constructed using global
configuration from which dhcp options are copied
```

Definition at line 82 of file dhcp.py.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 __init__()

```
def dhcp.WriteBootProtocolPacket.__init__ (
              self,
              configuration )
```

```
Create new packet instance and search for options set in configuration
and copy them tgo packet
```

Definition at line 107 of file dhcp.py.

7.17.3 Member Function Documentation

7.17.3.1 __str__()

```
def dhcp.WriteBootProtocolPacket.__str__ (
              self )
```

```
Serialize UDP DHCP response packet to bytes
```

Definition at line 188 of file dhcp.py.

7.17.3.2 get_option()

```
def dhcp.WriteBootProtocolPacket.get_option (
              self,
              option )
```

```
Get DHCP UDP response packet option value
```

Definition at line 152 of file dhcp.py.

7.17.3.3 options()

```
def dhcp.WriteBootProtocolPacket.options (
              self )
```

```
Get DHCP UDP response packet option value
```

Definition at line 167 of file dhcp.py.

7.17.3.4 to_bytes()

```
def dhcp.WriteBootProtocolPacket.to_bytes (
              self )
```

```
Serialize UDP DHCP response packet to bytes
```

Definition at line 119 of file dhcp.py.

7.17.4 Member Data Documentation

7.17.4.1 bootp_flags

```
int dhcp.WriteBootProtocolPacket.bootp_flags = 0  [static]
```

Definition at line 95 of file dhcp.py.

7.17.4.2 client_ip_address

```
string dhcp.WriteBootProtocolPacket.client_ip_address = '0.0.0.0'  [static]
```

Definition at line 97 of file dhcp.py.

7.17.4.3 client_mac_address

```
dhcp.WriteBootProtocolPacket.client_mac_address = None  [static]
```

Definition at line 102 of file dhcp.py.

7.17.4.4 hardware_address_length

```
int dhcp.WriteBootProtocolPacket.hardware_address_length = 6  [static]
```

Definition at line 89 of file dhcp.py.

7.17.4.5 hardware_type

```
int dhcp.WriteBootProtocolPacket.hardware_type = 1  [static]
```

Definition at line 88 of file dhcp.py.

7.17.4.6 hops

```
int dhcp.WriteBootProtocolPacket.hops = 0  [static]
```

Definition at line 90 of file dhcp.py.

7.17.4.7 magic_cookie

```
string dhcp.WriteBootProtocolPacket.magic_cookie = '99.130.83.99'  [static]
```

Definition at line 103 of file dhcp.py.

7.17.4.8 message_type

```
int dhcp.WriteBootProtocolPacket.message_type = 2  [static]
```

Definition at line 87 of file dhcp.py.

7.17.4.9 next_server_ip_address

```
string dhcp.WriteBootProtocolPacket.next_server_ip_address = '0.0.0.0'  [static]
```

Definition at line 99 of file dhcp.py.

7.17.4.10 parameter_order

```
list dhcp.WriteBootProtocolPacket.parameter_order = []  [static]
```

Definition at line 105 of file dhcp.py.

7.17.4.11 relay_agent_ip_address

```
string dhcp.WriteBootProtocolPacket.relay_agent_ip_address = '0.0.0.0'  [static]
```

Definition at line 100 of file dhcp.py.

7.17.4.12 seconds_elapsed

```
int dhcp.WriteBootProtocolPacket.seconds_elapsed = 0  [static]
```

Definition at line 94 of file dhcp.py.

7.17.4.13 transaction_id

```
dhcp.WriteBootProtocolPacket.transaction_id = None  [static]
```

Definition at line 92 of file dhcp.py.

7.17.4.14 your_ip_address

```
string dhcp.WriteBootProtocolPacket.your_ip_address = '0.0.0.0'  [static]
```

Definition at line 98 of file dhcp.py.

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

- dhcp.py

Chapter 8

File Documentation

8.1 dhcp.py File Reference

Classes

- class dhcp.TransactionDelayWorker
- class dhcp.PriorityQueue
- class dhcp.WriteBootProtocolPacket
- class dhcp.DHCPTransaction
- class dhcp.DHCPServerConfiguration
- class dhcp.ALL
- class dhcp.GREATER
- class dhcp.NETWORK
- class dhcp.CASEINSENSITIVE
- class dhcp.CSVDatabase
- class dhcp.Host
- class dhcp.HostDatabase
- class dhcp.DHCPServer
- class dhcp.ThreadedTcpRequestHandler
- class dhcp.ThreadedTcpServer

Namespaces

- namespace dhcp

Functions

- def dhcp.get_host_ip_addresses ()
- def dhcp.ip_addresses (network, subnet_mask)
- def dhcp.sorted_hosts (hosts)
- def dhcp.debug_msg (msg, type)

Variables

- dhcp.ALL = ALL()
- dhcp.messages = TTLOrderedDict(default_ttl=86400)
- dhcp.configuration = DHCPServerConfiguration()
- dhcp.debug
- dhcp.ip_address_lease_time
- dhcp.server = DHCPServer(configuration)

8.2 dhcp.py

Go to the documentation of this file.

```python
#!/usr/bin/env python3

import time
import threading
import struct
import queue
import collections
import traceback
import random
import socket
import heapq
import sys
from os.path import exists
import re
from ttldict import  TTLOrderedDict
import socketserver
from listener import *


class TransactionDelayWorker(object):
    """Class used to delay response to DHCP client
    """
    def __init__(self):
        """class constructor internally using priority queue where priority is time
        """
        self.closed = False
        self.queue = PriorityQueue()
        self.thread = threading.Thread(target = self._delay_response_thread)
        self.thread.start()

    def _delay_response_thread(self):
        """thread worker
        """
        while not self.closed:
            if self.closed:
                break
            if self.queue.qsize() > 0:
                p = self.queue.get()
                t, func, args, kw = p
                now = time.time()
                if now < t:
                    time.sleep(0.01)
                    self.queue.put(p)
                else:
                    func(*args, **kw)


    def do_after(self, seconds, func, args = (), kw = {}):
        """Add to queue function which should be called after certain time
        specified by seconds, args, kw are arguments
        """
        self.queue.put((time.time() + seconds, func, args, kw))

    def close(self):
        """Method used to stop worker
        """
        self.closed = True

def get_host_ip_addresses():
    """Get IP address of current host.
    """
    return gethostbyname_ex(gethostname())[2]

class PriorityQueue(object):
    """This class contains Heapq for more information:
```

```
    https://docs.python.org/3/library/heapq.html
    """
    def __init__(self):
        self._queue = []
        self._index = 0

    def put(self, item):
        heapq.heappush(self._queue, (self._index, item))
        self._index += 1

    def get(self):
        return heapq.heappop(self._queue)[-1]

    def qsize(self):
        return len(self._queue)

class WriteBootProtocolPacket(object):
    """DHCP protocol datagram serializer
    This class serializes UDP DHCP packet, instance is constructed using global
    configuration from which dhcp options are copied
    """
    message_type = 2 # 1 for client -> server 2 for server -> client
    hardware_type = 1
    hardware_address_length = 6
    hops = 0

    transaction_id = None

    seconds_elapsed = 0
    bootp_flags = 0 # unicast

    client_ip_address = '0.0.0.0'
    your_ip_address = '0.0.0.0'
    next_server_ip_address = '0.0.0.0'
    relay_agent_ip_address = '0.0.0.0'

    client_mac_address = None
    magic_cookie = '99.130.83.99'

    parameter_order = []

    def __init__(self, configuration):
        """Create new packet instance and search for options set in configuration
        and copy them tgo packet
        """
        for i in range(256):
            names = ['option_{}'.format(i)]
            if i < len(options) and hasattr(configuration, options[i][0]):
                names.append(options[i][0])
            for name in names:
                if hasattr(configuration, name):
                    setattr(self, name, getattr(configuration, name))

    def to_bytes(self):
        """Serialize UDP DHCP response packet to bytes
        """
        result = bytearray(236)

        result[0] = self.message_type
        result[1] = self.hardware_type
        result[2] = self.hardware_address_length
        result[3] = self.hops

        result[4:8] = struct.pack('>I', self.transaction_id)

        result[ 8:10] = shortpack(self.seconds_elapsed)
        result[10:12] = shortpack(self.bootp_flags)

        result[12:16] = inet_aton(self.client_ip_address)
        result[16:20] = inet_aton(self.your_ip_address)
        result[20:24] = inet_aton(self.next_server_ip_address)
        result[24:28] = inet_aton(self.relay_agent_ip_address)

        result[28:28 + self.hardware_address_length] = macpack(self.client_mac_address)

        result += inet_aton(self.magic_cookie)

        for option in self.options:
            value = self.get_option(option)
            #print(option, value)
            if value is None:
                continue
            result += bytes([option, len(value)]) + value
        result += bytes([255])
        return bytes(result)

    def get_option(self, option):
        """Get DHCP UDP response packet option value
        """
        if option < len(options) and hasattr(self, options[option][0]):
            value = getattr(self, options[option][0])
        elif hasattr(self, 'option_{}'.format(option)):
            value = getattr(self, 'option_{}'.format(option))
        else:
            return None
        function = options[option][2]
        if function and value is not None:
            value = function(value)
        return value

    @property
    def options(self):
        """Get DHCP UDP response packet option value
        """
        done = list()
        # fulfill wishes
        for option in self.parameter_order:
            if option < len(options) and hasattr(self, options[option][0]) or hasattr(self,
      'option_{}'.format(option)):
                # this may break with the specification because we must try to fulfill the wishes
                if option not in done:
                    done.append(option)
        # add my stuff
        for option, o in enumerate(options):
            if o[0] and hasattr(self, o[0]):
                if option not in done:
                    done.append(option)
        for option in range(256):
            if hasattr(self, 'option_{}'.format(option)):
                if option not in done:
                    done.append(option)
        return done

    def __str__(self):
        """Serialize UDP DHCP response packet to bytes
        """
        return str(ReadBootProtocolPacket(self.to_bytes()))


class DHCPTransaction(object):
    """Class representing DHCP Transaction
    """
    def __init__(self, server):
        """Contructor of new transaction
        """
        self.server = server
        self.configuration = server.configuration
        self.packets = []
        self.done_time = time.time() + self.configuration.length_of_transaction
        self.done = False
        self.do_after = self.server.delay_worker.do_after
        self.debug = debug

    def is_done(self):
        """Check if transaction is done
        """
        return self.done or self.done_time < time.time()

    def close(self):
        """Close transaction
        """
        self.done = True

    def receive(self, packet):
        """Receive DHCP UDP packet check it's type and call a proper callback
        """
        # packet from client <-> packet.message_type == 1
        if packet.message_type == 1 and packet.dhcp_message_type == 'DHCPDISCOVER':
            self.do_after(self.configuration.dhcp_offer_after_seconds,
                          self.received_dhcp_discover, (packet,), )
        elif packet.message_type == 1 and packet.dhcp_message_type == 'DHCPREQUEST':
            self.do_after(self.configuration.dhcp_acknowledge_after_seconds,
                          self.received_dhcp_request, (packet,), )
        elif packet.message_type == 1 and packet.dhcp_message_type == 'DHCPINFORM':
            self.received_dhcp_inform(packet)
        else:
            return False
        return True

    def received_dhcp_discover(self, discovery):
        """Method used to handle DHCP Discover packet
        """
        if self.is_done(): return
        self.configuration.debug('discover:\n {}'.format(str(discovery).replace('\n', '\n\t')))
        self.send_offer(discovery)

    def send_offer(self, discovery):
        """Method used to send DHCP offer packet
        """
        # https://tools.ietf.org/html/rfc2131
        offer = WriteBootProtocolPacket(self.configuration)
        offer.parameter_order = discovery.parameter_request_list
        mac = discovery.client_mac_address
        ip = offer.your_ip_address = self.server.get_ip_address(discovery)
        # offer.client_ip_address =
        offer.transaction_id = discovery.transaction_id
        # offer.next_server_ip_address =
        offer.relay_agent_ip_address = discovery.relay_agent_ip_address
        offer.client_mac_address = mac
        offer.client_ip_address = discovery.client_ip_address or '0.0.0.0'
        offer.bootp_flags = discovery.bootp_flags
        offer.dhcp_message_type = 'DHCPOFFER'
        offer.client_identifier = mac
        self.configuration.debug('offer:\n {}'.format(str(offer).replace('\n', '\n\t')))
        self.server.broadcast(offer)

    def received_dhcp_request(self, request):
        """Method used to handle DHCP Request packet
        """
        if self.is_done(): return
        self.configuration.debug('request:\n {}'.format(str(request).replace('\n', '\n\t')))
        self.server.client_has_chosen(request)
        self.acknowledge(request)
        self.close()

    def acknowledge(self, request):
        """Method used to handle DHCP Acknowledge packet
        """
        ack = WriteBootProtocolPacket(self.configuration)
        ack.parameter_order = request.parameter_request_list
        ack.transaction_id = request.transaction_id
        # ack.next_server_ip_address =
        ack.bootp_flags = request.bootp_flags
        ack.relay_agent_ip_address = request.relay_agent_ip_address
        mac = request.client_mac_address
        ack.client_mac_address = mac
        requested_ip_address = request.requested_ip_address
        ack.client_ip_address = request.client_ip_address or '0.0.0.0'
        ack.your_ip_address = self.server.get_ip_address(request)
        ack.dhcp_message_type = 'DHCPACK'
        self.configuration.debug('acknowledge:\n {}'.format(str(ack).replace('\n', '\n\t')))
        self.server.broadcast(ack)

    def received_dhcp_inform(self, inform):
        """Method used to handle DHCP Inform packet
        """
        self.configuration.debug('inform:\n {}'.format(str(inform).replace('\n', '\n\t')))
        self.close()
        self.server.client_has_chosen(inform)

class DHCPServerConfiguration(object):
    """Class to load DHCP server configuration from file or command line
    """
    dhcp_offer_after_seconds = 10
    dhcp_acknowledge_after_seconds = 10
    length_of_transaction = 40

    network = '192.168.173.0'
    broadcast_address = '255.255.255.255'
    subnet_mask = '255.255.255.0'
    router = None # list of ips
    # 1 day is 86400
    ip_address_lease_time = 300 # seconds
    domain_name_server = None # list of ips

    host_file = 'hosts.csv'

    debug = lambda *args, **kw: None

    def load(self, file):
        """Load configuration from file using exec to parse file as object dictionary
        or get ALL command line arguments and change them using regexp to file layout
        and treat as file
        """
        if(len(file) > 0 and exists(file)):
            with open(file) as f:
                exec(f.read(), self.__dict__)
        else:
            args = ' '.join(sys.argv[1:])
            args = re.sub(' -', "\r\n", args)
            args = re.sub('^-', ", args)
            args = re.sub('^([a-z_]+)([ ]+)(.+)$', r"\1=\3", args, flags=re.MULTILINE)
            exec(args, self.__dict__)

    def adjust_if_this_computer_is_a_router(self):
        """Automatically adjust some DHCP configuration parameters if this computer is router
        """
        ip_addresses = get_host_ip_addresses()
        for ip in reversed(ip_addresses):
            if ip.split('.')[-1] == '1':
                self.router = [ip]
                self.domain_name_server = [ip]
                self.networknetwork = '.'.join(ip.split('.')[:-1] + ['0'])
                self.broadcast_addressbroadcast_address = '.'.join(ip.split('.')[:-1] + ['255'])
                #self.ip_forwarding_enabled = True
                #self.non_local_source_routing_enabled = True
                #self.perform_mask_discovery = True

    def all_ip_addresses(self):
        ips = ip_addresses(self.networknetwork, self.subnet_mask)
        for i in range(5):
            next(ips)
        return ips

    def network_filter(self):
        return NETWORK(self.networknetwork, self.subnet_mask)

def ip_addresses(network, subnet_mask):
    import socket, struct
    subnet_mask = struct.unpack('>I', socket.inet_aton(subnet_mask))[0]
    network = struct.unpack('>I', socket.inet_aton(network))[0]
    network = network & subnet_mask
    start = network + 1
    end = (network | (~subnet_mask & 0xffffffff))
    return (socket.inet_ntoa(struct.pack('>I', i)) for i in range(start, end))

class ALL(object):
    """Comparator class
    """
    def __eq__(self, other):
        return True
    def __repr__(self):
        return self.__class__.__name__

ALL = ALL()

class GREATER(object):
    """Comparator class
    """
    def __init__(self, value):
        self.value = value
    def __eq__(self, other):
        return type(self.value)(other) > self.value

class NETWORK(object):
    """Comparator class to check if address within same network
    """
    def __init__(self, network, subnet_mask):
        self.subnet_mask = struct.unpack('>I', inet_aton(subnet_mask))[0]
        self.network = struct.unpack('>I', inet_aton(network))[0]
    def __eq__(self, other):
        ip = struct.unpack('>I', inet_aton(other))[0]
        return ip & self.subnet_mask == self.network and \
               ip - self.network and \
               ip - self.network != ~self.subnet_mask & 0xffffffff

class CASEINSENSITIVE(object):
    """Comparator class
    """
    def __init__(self, s):
        self.s = s.lower()
    def __eq__(self, other):
        return self.s == other.lower()

class CSVDatabase(object):
    """Class handling CSV file database to keep host definitions
    """
    delimiter = ';'

    def __init__(self, file_name):
        """Construct new CSV database with storage in file_name
        """
        self.file_name = file_name
        self.file('a').close() # create file

    def file(self, mode = 'r'):
        """Open CSV file with selected mode
        """
        return open(self.file_name, mode)

    def get(self, pattern):
        """Get CSV entry representing host(MAC) and lease(IP)
        """
        pattern = list(pattern)
        return [line for line in self.all() if pattern == line]

    def add(self, line):
        """Add host entry to CSV file
        """
        with self.file('a') as f:
            f.write(self.delimiter.join(line) + '\n')

    def delete(self, pattern):
        """Delete host entry from CSV file
        """
        lines = self.all()
        lines_to_delete = self.get(pattern)
        self.file('w').close() # empty file
        for line in lines:
            if line not in lines_to_delete:
                self.add(line)

    def all(self):
        """Get all entries from CSV file
        """
        with self.file() as f:
            return [list(line.strip().split(self.delimiter)) for line in f]

class Host(object):
    """Class representing host with MAC address, IP, hostname if available and last used timestamp
    """
    def __init__(self, mac, ip, hostname, last_used):
        self.mac = mac.upper()
        self.ip = ip
        self.hostname = hostname
        self.last_used = int(last_used)

    @classmethod
    def from_tuple(cls, line):
        mac, ip, hostname, last_used = line
        last_used = int(last_used)
        return cls(mac, ip, hostname, last_used)

    @classmethod
    def from_packet(cls, packet):
        return cls(packet.client_mac_address,
                   packet.requested_ip_address or packet.client_ip_address,
                   packet.host_name or '',
                   int(time.time()))

    @staticmethod
    def get_pattern(mac = ALL, ip = ALL, hostname = ALL, last_used = ALL):
        return [mac, ip, hostname, last_used]

    def to_tuple(self):
        """Convert host to tuple
        """
        return [self.mac, self.ip, self.hostname, str(int(self.last_used))]

    def to_pattern(self):
        """Convert host to pattern
        """
        return self.get_pattern(ip = self.ip, mac = self.mac)

    def __hash__(self):
        return hash(self.key)

    def __eq__(self, other):
        return self.to_tuple() == other.to_tuple()

    def has_valid_ip(self):
        """Check if host has valid IP address
        """
        return self.ip and self.ip != '0.0.0.0'

class HostDatabase(object):
    def __init__(self, file_name):
        self.db = CSVDatabase(file_name)

    def get(self, **kw):
        pattern = Host.get_pattern(**kw)
        return list(map(Host.from_tuple, self.db.get(pattern)))

    def add(self, host):
        self.db.add(host.to_tuple())

    def delete(self, host = None, **kw):
        if host is None:
            pattern = Host.get_pattern(**kw)
        else:
            pattern = host.to_pattern()
        self.db.delete(pattern)

    def all(self):
        return list(map(Host.from_tuple, self.db.all()))

    def replace(self, host):
        self.delete(host)
        self.add(host)

def sorted_hosts(hosts):
    hosts = list(hosts)
    hosts.sort(key = lambda host: (host.hostname.lower(), host.mac.lower(), host.ip.lower()))
    return hosts

class DHCPServer(object):

    def __init__(self, configuration = None):
        if configuration == None:
            configuration = DHCPServerConfiguration()

        self.configuration = configuration
        self.socket = socket(type = SOCK_DGRAM)
        self.socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
        self.socket.bind(('', 67))
        self.delay_worker = TransactionDelayWorker()
        self.closed = False
        self.transactions = collections.defaultdict(lambda: DHCPTransaction(self)) # id: transaction
        self.hosts = HostDatabase(self.configuration.host_file)
        self.time_started = time.time()

    def close(self):
        self.socket.close()
        self.closed = True
        self.delay_worker.close()
        for transaction in list(self.transactions.values()):
            transaction.close()

    def update(self, timeout = 0):
        try:
            reads = select.select([self.socket], [], [], timeout)[0]
        except ValueError:
            # ValueError: file descriptor cannot be a negative integer (-1)
            return
        for socket in reads:
            try:
                packet = ReadBootProtocolPacket(*socket.recvfrom(4096))
            except OSError:
                # OSError: [WinError 10038] An operation was attempted on something that is not a
 socket
                pass
            else:
                self.received(packet)
        for transaction_id, transaction in list(self.transactions.items()):
            if transaction.is_done():
                transaction.close()
                self.transactions.pop(transaction_id)

    def received(self, packet):
        if not self.transactions[packet.transaction_id].receive(packet):
            self.configuration.debug('received:\n {}'.format(str(packet).replace('\n', '\n\t')))

    def client_has_chosen(self, packet):
        self.configuration.debug('client_has_chosen:\n {}'.format(str(packet).replace('\n', '\n\t')))
        host = Host.from_packet(packet)
        if not host.has_valid_ip():
            return
        self.hosts.replace(host)

    def is_valid_client_address(self, address):
        if address is None:
            return False
        a = address.split('.')
        s = self.configuration.subnet_mask.split('.')
        n = self.configuration.network.split('.')
        return all(s[i] == '0' or a[i] == n[i] for i in range(4))

    def get_ip_address(self, packet):
        mac_address = packet.client_mac_address
        requested_ip_address = packet.requested_ip_address
        known_hosts = self.hosts.get(mac = CASEINSENSITIVE(mac_address))
        ip = None
        if known_hosts:
            # 1. choose known ip address
            for host in known_hosts:
                if self.is_valid_client_address(host.ip):
                    ip = host.ip
            print(’known ip:’, ip)
        if ip is None and self.is_valid_client_address(requested_ip_address):
            # 2. choose valid requested ip address
            ip = requested_ip_address
            print(’valid ip:’, ip)
        if ip is None:
            # 3. choose new, free ip address
            chosen = False
            network_hosts = self.hosts.get(ip = self.configuration.network_filter())
            for ip in self.configuration.all_ip_addresses():
                if not any(host.ip == ip for host in network_hosts):
                    chosen = True
                    break
            if not chosen:
                # 4. reuse old valid ip address
                network_hosts.sort(key = lambda host: host.last_used)
                ip = network_hosts[0].ip
                assert self.is_valid_client_address(ip)
            print(’new ip:’, ip)
        if not any([host.ip == ip for host in known_hosts]):
            print(’add’, mac_address, ip, packet.host_name)
            self.hosts.replace(Host(mac_address, ip, packet.host_name or ”, time.time()))
        return ip

    @property
    def server_identifiers(self):
        return get_host_ip_addresses()

    def broadcast(self, packet):
        self.configuration.debug(’broadcasting:\n {}’.format(str(packet).replace(’\n’, ’\n\t’)))
        for addr in self.server_identifiers:
            broadcast_socket = socket(type = SOCK_DGRAM)
            broadcast_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
            broadcast_socket.setsockopt(SOL_SOCKET, SO_BROADCAST, 1)
            packet.server_identifier = addr
            broadcast_socket.bind((addr, 67))
            try:
                data = packet.to_bytes()
                broadcast_socket.sendto(data, (’255.255.255.255’, 68))
                broadcast_socket.sendto(data, (addr, 68))
            finally:
                broadcast_socket.close()

    def run(self):
        while not self.closed:
            try:
                self.update(1)
            except KeyboardInterrupt:
                break
            except:
                traceback.print_exc()

    def run_in_thread(self):
        thread = threading.Thread(target = self.run)
        thread.start()
        return thread

    def debug_clients(self):
        for line in self.ips.all():
            line = ’\t’.join(line)
            if line:
                self.configuration.debug(line)

    def get_all_hosts(self):
        return sorted_hosts(self.hosts.get())

    def get_current_hosts(self):
        return sorted_hosts(self.hosts.get(last_used = GREATER(self.time_started)))

class ThreadedTcpRequestHandler(socketserver.StreamRequestHandler):
    """Control socket client connection handler
    """
    def handle(self):
        """Method used to handle client connection parsing commands and giving response to them
        """
        self.request.sendall(bytes("Welcome to micro python dhcp server", ’ascii’))
        try:
            while(True):
                self.request.sendall(bytes("\r\npydhcp ?> ", ’ascii’))
                data = self.rfile.readline().strip()
                if(data.decode() == "hosts"):
                    self.request.sendall(bytes("Active
Hosts:\r\n{}".format("\r\n".join(self.server.hosts.all())),'ascii'))
                elif(data.decode() == "events"):
                    self.request.sendall(bytes("Events last 24h:\r\n{}".format("\r\n".join(self.server.events.items())),'ascii'))
                elif(data.decode() == "configuration"):
                    self.request.sendall(bytes("Current configuration\r\n", 'ascii'))
                    for value in options:
                        if(hasattr(self.server.configuration,value[0])):
                            self.request.sendall(bytes("{}: {}\r\n".format(value[0],getattr(self.server.configuration,value[0])),'ascii'))
                elif(data.decode() == "help"):
                    self.request.sendall(bytes("hosts\t\tdisplay host database\r\n",'ascii'))
                    self.request.sendall(bytes("events\t\tdisplay DHCP event log\r\n",'ascii'))
                    self.request.sendall(bytes("configuration\tdisplay current server configuration\r\n",'ascii'))
                    self.request.sendall(bytes("help\t\tthis command\r\n",'ascii'))
                    self.request.sendall(bytes("quit\t\tdisconnect from current session\r\n",'ascii'))
                elif(data.decode() == "quit"):
                    self.request.sendall(bytes("bye\r\n", 'ascii'))
                    break
                else:
                    self.request.sendall(bytes("unknown command: {}".format(data.decode('ascii')), 'ascii'))
        except Exception as e:
            pass

class ThreadedTcpServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    """DHCP server control interface TCP server
    """
    def setEvents(self,data):
        """Set DHCP events dictionary reference
        """
        self.events = data

    def setHosts(self,data):
        """Set DHCP host database with active leases reference
        """
        self.hosts = data

    def setConfiguration(self,data):
        """Set DHCP UDP global configuration reference
        """
        self.configuration = data

if __name__ == '__main__':

    messages = TTLOrderedDict(default_ttl=86400) #keep messages for 24h

    def debug_msg(msg,type):
        if bool(type):
            type = 'debug'
        messages[time.time()] = { 'type': type, 'msg': msg }

    configuration = DHCPServerConfiguration()
    configuration.debug = debug_msg
    configuration.adjust_if_this_computer_is_a_router()
    configuration.load(sys.argv[1])
    configuration.router #+= ['192.168.0.1']
    configuration.ip_address_lease_time = 60
    server = DHCPServer(configuration)

    for ip in server.configuration.all_ip_addresses():
        assert ip == server.configuration.network_filter()

    with ThreadedTcpServer(("127.0.0.1", 6767), ThreadedTcpRequestHandler) as cserver:
        cserver.setEvents(messages)
        cserver.setHosts(server.hosts.db)
        cserver.setConfiguration(configuration)
        cserver.serve_forever()

    #server.run_in_thread()
    server.run()
```

8.3 listener.py File Reference

Classes

- class listener.ReadBootProtocolPacket

Namespaces

- namespace listener

Functions

- def listener.inet_ntoaX (data)
- def listener.inet_atonX (ips)
- def listener.macunpack (data)
- def listener.macpack (mac)
- def listener.unpackbool (data)
- def listener.packbool (bool)

Variables

- dictionary listener.dhcp_message_types
- listener.reversed_dhcp_message_types = dict()
- listener.shortunpack = lambda data: (data[0] << 8) + data[1]
- listener.shortpack = lambda i: bytes([i >> 8, i & 255])
- list listener.options
- listener.data = base64.b16decode(b'02010600f7b41ad100000000c0a80064000000000000000000000000007c7a914bca6c00000
- listener.p = ReadBootProtocolPacket(data)
- listener.s1 = socket(type = SOCK_DGRAM)
- listener.reads = select.select([s1], [ ], [ ], 1)[0]
- listener.packet = ReadBootProtocolPacket(∗s.recvfrom(4096))

8.4 listener.py

Go to the documentation of this file.

```
#!/usr/bin/env python3
from socket import *

import struct
import base64
import select

# see https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
# section DHCP options

def inet_ntoaX(data):
    return ['.'.join(map(str, data[i:i + 4])) for i in range(0, len(data), 4)]

def inet_atonX(ips):
    return b''.join(map(inet_aton, ips))

dhcp_message_types = {
    1 : 'DHCPDISCOVER',
    2 : 'DHCPOFFER',
    3 : 'DHCPREQUEST',
    4 : 'DHCPDECLINE',
    5 : 'DHCPACK',
    6 : 'DHCPNAK',
    7 : 'DHCPRELEASE',
    8 : 'DHCPINFORM',
}
reversed_dhcp_message_types = dict()
for i, v in dhcp_message_types.items():
    reversed_dhcp_message_types[v] = i

shortunpack = lambda data: (data[0] « 8) + data[1]
shortpack = lambda i: bytes([i » 8, i & 255])


def macunpack(data):
    s = base64.b16encode(data)
    return ':'.join([s[i:i+2].decode('ascii') for i in range(0, 12, 2)])

def macpack(mac):
    return base64.b16decode(mac.replace(':', '').replace('-', '').encode('ascii'))

def unpackbool(data):
    return data[0]

def packbool(bool):
    return bytes([bool])

options = [
# RFC1497 vendor extensions
    ('pad', None, None),
    ('subnet_mask', inet_ntoa, inet_aton),
    ('time_offset', None, None),
    ('router', inet_ntoaX, inet_atonX),
    ('time_server', inet_ntoaX, inet_atonX),
    ('name_server', inet_ntoaX, inet_atonX),
    ('domain_name_server', inet_ntoaX, inet_atonX),
    ('log_server', inet_ntoaX, inet_atonX),
    ('cookie_server', inet_ntoaX, inet_atonX),
    ('lpr_server', inet_ntoaX, inet_atonX),
    ('impress_server', inet_ntoaX, inet_atonX),
    ('resource_location_server', inet_ntoaX, inet_atonX),
    ('host_name', lambda d: d.decode('ASCII'), lambda d: d.encode('ASCII')),
    ('boot_file_size', None, None),
    ('merit_dump_file', None, None),
    ('domain_name', None, None),
    ('swap_server', inet_ntoa, inet_aton),
    ('root_path', None, None),
    ('extensions_path', None, None),
# IP Layer Parameters per Host
    ('ip_forwarding_enabled', unpackbool, packbool),
    ('non_local_source_routing_enabled', unpackbool, packbool),
    ('policy_filer', None, None),
    ('maximum_datagram_reassembly_size', shortunpack, shortpack),
    ('default_ip_time_to_live', lambda data: data[0], lambda i: bytes([i])),
    ('path_mtu_aging_timeout', None, None),
    ('path_mtu_plateau_table', None, None),
# IP Layer Parameters per Interface
    ('interface_mtu', None, None),
    ('all_subnets_are_local', unpackbool, packbool),
    ('broadcast_address', inet_ntoa, inet_aton),
    ('perform_mask_discovery', unpackbool, packbool),
    ('mask_supplier', None, None),
    ('perform_router_discovery', None, None),
    ('router_solicitation_address', inet_ntoa, inet_aton),
    ('static_route', None, None),
# Link Layer Parameters per Interface
    ('trailer_encapsulation_option', None, None),
    ('arp_cache_timeout', None, None),
    ('ethernet_encapsulation', None, None),
# TCP Parameters
    ('tcp_default_ttl', None, None),
    ('tcp_keep_alive_interval', None, None),
    ('tcp_keep_alive_garbage', None, None),
# Application and Service Parameters Part 1
    ('network_information_service_domain', None, None),
    ('network_informtaion_servers', inet_ntoaX, inet_atonX),
    ('network_time_protocol_servers', inet_ntoaX, inet_atonX),
    ('vendor_specific_information', None, None),
    ('netbios_over_tcp_ip_name_server', inet_ntoaX, inet_atonX),
    ('netbios_over_tcp_ip_datagram_distribution_server', inet_ntoaX, inet_atonX),
    ('netbios_over_tcp_ip_node_type', None, None),
    ('netbios_over_tcp_ip_scope', None, None),
    ('x_window_system_font_server', inet_ntoaX, inet_atonX),
    ('x_window_system_display_manager', inet_ntoaX, inet_atonX),
# DHCP Extensions
    ('requested_ip_address', inet_ntoa, inet_aton),
    ('ip_address_lease_time', lambda d: struct.unpack('>I', d)[0], lambda i: struct.pack('>I', i)),
    ('option_overload', None, None),
    ('dhcp_message_type', lambda data: dhcp_message_types.get(data[0], data[0]), (lambda name:
  bytes([reversed_dhcp_message_types.get(name, name)]))),
    ('server_identifier', inet_ntoa, inet_aton),
    ('parameter_request_list', list, bytes),
    ('message', None, None),
    ('maximum_dhcp_message_size', shortunpack, shortpack),
    ('renewal_time_value', None, None),
    ('rebinding_time_value', None, None),
    ('vendor_class_identifier', None, None),
    ('client_identifier', macunpack, macpack),
    ('tftp_server_name', None, None),
    ('boot_file_name', None, None),
# Application and Service Parameters Part 2
    ('network_information_service_domain', None, None),
    ('network_information_servers', inet_ntoaX, inet_atonX),
    ('', None, None),
    ('', None, None),
    ('mobile_ip_home_agent', inet_ntoaX, inet_atonX),
    ('smtp_server', inet_ntoaX, inet_atonX),
    ('pop_servers', inet_ntoaX, inet_atonX),
    ('nntp_server', inet_ntoaX, inet_atonX),
    ('default_www_server', inet_ntoaX, inet_atonX),
    ('default_finger_server', inet_ntoaX, inet_atonX),
    ('default_irc_server', inet_ntoaX, inet_atonX),
    ('streettalk_server', inet_ntoaX, inet_atonX),
    ('stda_server', inet_ntoaX, inet_atonX),
    ]

assert options[18][0] == 'extensions_path', options[18][0]
assert options[25][0] == 'path_mtu_plateau_table', options[25][0]
assert options[33][0] == 'static_route', options[33][0]
assert options[50][0] == 'requested_ip_address', options[50][0]
assert options[64][0] == 'network_information_service_domain', options[64][0]
assert options[76][0] == 'stda_server', options[76][0]


class ReadBootProtocolPacket(object):

    for i, o in enumerate(options):
        locals()[o[0]] = None
        locals()['option_{0}'.format(i)] = None

    del i, o

    def __init__(self, data, address = ('0.0.0.0', 0)):
        self.data = data
        self.address = address
        self.host = address[0]
        self.port = address[1]

        # wireshark = wikipedia = data[...]

        self.message_type = self.OP =                data[0]
        self.hardware_type = self.HTYPE =            data[1]
        self.hardware_address_length = self.HLEN =   data[2]
        self.hops = self.HOPS =                      data[3]

        self.XID = self.transaction_id = struct.unpack('>I', data[4:8])[0]

        self.seconds_elapsed = self.SECS = shortunpack(data[8:10])
        self.bootp_flags = self.FLAGS =    shortunpack(data[8:10])

        self.client_ip_address = self.CIADDR = inet_ntoa(data[12:16])
        self.your_ip_address   = self.YIADDR = inet_ntoa(data[16:20])
        self.next_server_ip_address = self.SIADDR = inet_ntoa(data[20:24])
        self.relay_agent_ip_address = self.GIADDR = inet_ntoa(data[24:28])

        self.client_mac_address = self.CHADDR = macunpack(data[28: 28 + self.hardware_address_length])
        index = 236
        self.magic_cookie = self.magic_cookie = inet_ntoa(data[index:index + 4]); index += 4
        self.options = dict()
        self.named_options = dict()
        while index < len(data):
            option = data[index]; index += 1
            if option == 0:
                # padding
                # Can be used to pad other options so that they are aligned to the word boundary; is
 not followed by length byte
                continue
            if option == 255:
                # end
                break
            option_length = data[index]; index += 1
            option_data = data[index: index + option_length]; index += option_length
            self.options[option] = option_data
            if option < len(options):
                option_name, function, _ = options[option]
                if function:
                    option_data = function(option_data)
                if option_name:
                    setattr(self, option_name, option_data)
                    self.named_options[option_name] = option_data
            setattr(self, 'option_{}'.format(option), option_data)

    def __getitem__(self, key):
        print(key, dir(self))
        return getattr(self, key, None)

    def __contains__(self, key):
        return key in self.__dict__

    @property
    def formatted_named_options(self):
        return "\n".join("{}:\t{}".format(name.replace('_', ' '), value) for name, value in
     sorted(self.named_options.items()))

    def __str__(self):
        return """Message Type: {self.message_type}
client MAC address: {self.client_mac_address}
client IP address: {self.client_ip_address}
your IP address: {self.your_ip_address}
next server IP address: {self.next_server_ip_address}
{self.formatted_named_options}
""".format(self = self)

    def __gt__(self, other):
        return id(self) < id(other)

data =
 base64.b16decode(b'02010600f7b41ad100000000c0a80064000000000000000000000000 7c7a914bca6c00000000000000000000000000000000
assert data[0] == 2
p = ReadBootProtocolPacket(data)
assert p.message_type == 2
assert p.hardware_type == 1
assert p.hardware_address_length == 6
assert p.hops == 0
assert p.transaction_id == 4155775697
assert p.seconds_elapsed == 0
assert p.bootp_flags == 0
assert p.client_ip_address == '192.168.0.100'
assert p.your_ip_address == '0.0.0.0'
assert p.next_server_ip_address == '0.0.0.0'
assert p.relay_agent_ip_address == '0.0.0.0'
assert p.client_mac_address.lower() == '7c:7a:91:4b:ca:6c'
assert p.magic_cookie == '99.130.83.99'
assert p.dhcp_message_type == 'DHCPACK'
assert p.options[53] == b'\x05'
assert p.server_identifier == '192.168.0.1'
assert p.subnet_mask == '255.255.255.0'
assert p.router == ['192.168.0.1']
assert p.domain_name_server == ['192.168.0.1']
str(p)

if __name__ == '__main__':
    s1 = socket(type = SOCK_DGRAM)
    s1.setsockopt(SOL_IP, SO_REUSEADDR, 1)
    s1.bind(('', 67))
    #s2 = socket(type = SOCK_DGRAM)
    #s2.setsockopt(SOL_IP, SO_REUSEADDR, 1)
    #s2.bind(('', 68))
    while 1:
        reads = select.select([s1], [], [], 1)[0]
        for s in reads:
            packet = ReadBootProtocolPacket(*s.recvfrom(4096))
            print(packet)
```

8.5 README.md File Reference

8.6 ttldict.py File Reference

Classes

- class ttldict.TTLOrderedDict

Namespaces

- namespace ttldict

8.7 ttldict.py

Go to the documentation of this file.

```
from collections import OrderedDict
from threading import RLock
import time

__all__ = ['TTLOrderedDict']


class TTLOrderedDict(OrderedDict):
    """
    OrderedDict with TTL
    Extra args and kwargs are passed to initial .update() call
    """
    def __init__(self, default_ttl, *args, **kwargs):
        """
        Be warned, if you use this with Python versions earlier than 3.6
        when passing **kwargs order is not preseverd.
        """
        assert isinstance(default_ttl, int)
        self._default_ttl = default_ttl
        self._lock = RLock()
        super().__init__()
        self.update(*args, **kwargs)

    def __repr__(self):
        return '<TTLOrderedDict@%#08x; ttl=%r, OrderedDict=%r;>' % (
            id(self), self._default_ttl, self.items())

    def __len__(self):
        with self._lock:
            self._purge()
            return super().__len__()

    def set_ttl(self, key, ttl, now=None):
        """Set TTL for the given key"""
        if now is None:
            now = time.time()
        with self._lock:
            value = self[key]
            super().__setitem__(key, (now + ttl, value))

    def get_ttl(self, key, now=None):
        """Return remaining TTL for a key"""
        if now is None:
            now = time.time()
        with self._lock:
            expire, _value = super().__getitem__(key)
            return expire - now

    def expire_at(self, key, timestamp):
        """Set the key expire timestamp"""
        with self._lock:
            value = self.__getitem__(key)
            super().__setitem__(key,  (timestamp, value))

    def is_expired(self, key, now=None):
        """ Check if key has expired, and return it if so"""
        with self._lock:
            if now is None:
                now = time.time()

            expire, _value = super().__getitem__(key)

            if expire:
                if expire < now:
                    return key

    def _purge(self):
        _keys = list(super().__iter__())
        _remove = [key for key in _keys if self.is_expired(key)]  # noqa
        [self.__delitem__(key) for key in _remove]

    def __iter__(self):
        """
        Yield only non expired keys, without purging the expired ones
        """
        with self._lock:
            for key in super().__iter__():
                if not self.is_expired(key):
                    yield key

    def __setitem__(self, key, value):
        with self._lock:
            if self._default_ttl is None:
                expire = None
            else:
                expire = time.time() + self._default_ttl
            super().__setitem__(key,  (expire, value))

    def __delitem__(self, key):
        with self._lock:
            super().__delitem__(key)

    def __getitem__(self, key):
        with self._lock:
            if self.is_expired(key):
                self.__delitem__(key)
                raise KeyError
            item = super().__getitem__(key)[1]
            return item

    def keys(self):
        with self._lock:
            self._purge()
            return super().keys()

    def items(self):
        with self._lock:
            self._purge()
            _items = list(super(OrderedDict, self).items())
            return [(k, v[1]) for (k, v) in _items]

    def values(self):
        with self._lock:
            self._purge()
            _values = list(super(OrderedDict, self).values())
            return [v[1] for v in _values]

    def get(self, key, default=None):
        try:
            return self[key]
        except KeyError:
            return default
```

Index