



# DHCPKit

## **DHCPKit Documentation**

*Release 1.0.1*

**S.J.M. Steffann**

**Jun 26, 2017**



---

## Contents

---

<b>1</b>	<b>Documentation</b>	<b>3</b>
1.1	IPv6 Server extension configuration . . . . .	3
1.2	dhcpcit_vpp package . . . . .	5
1.3	Changes per version . . . . .	12
1.4	Applicable copyright licences . . . . .	13
	<b>Python Module Index</b>	<b>15</b>



This DHCPKit extension allows DHCPKit to be used as the DHCPv6 server of a Vector Packet Processing (VPP) router. See <https://fd.io/technology> and <https://wiki.fd.io/view/VPP> for more information on VPP.

The [official documentation](#)<sup>1</sup> is hosted by [Read the Docs](#)<sup>2</sup>.

---

<sup>1</sup> <http://dhcpkit.readthedocs.io>

<sup>2</sup> <https://readthedocs.org>



## IPv6 Server extension configuration

This is the documentation of the configuration options of the `dhcpcd_vpp` package.

### Overview of sections

#### Vpp-interface

VPP Interface to listen on

#### Example

```
<vpp-interface GigabitEthernet0/1/2>  
  reply-from fe80::1  
  link-address 2001:db8::1  
</vpp-interface>
```

#### Section parameters

**accept-multicast** Whether to process multicast messages received on this interface

**Default:** “yes”

**accept-unicast** Whether to process unicast messages received on this interface

**Default:** “yes”

**reply-from** The link-local address to send on-link replies from

**Default:** The first link-local address found on the interface

**link-address** A global unicast address used to identify the link to filters and handlers. It doesn't even need to exist.

**Default:** The first global unicast address found on the interface, or :: otherwise

## Overview of section types

### Listeners

Configuration sections that define listeners. These are usually the network interfaces that a DHCPv6 server listens on, like the well-known multicast address on an interface, or a unicast address where a DHCPv6 relay can send its requests to.

### Listen-vpp

This listener sets up a two-way connection to a VPP instance using Unix domain sockets. It will learn the server created by the VPP instance using the VPP Python API. The name of the socket endpoint it creates for itself (so VPP can send messages to DHCPKit) is specified as the name of the section.

With this listener DHCPKit can become a DHCPv6 server for VPP. You must list all VPP interfaces that DHCPKit should respond to.

VPP must be configured to create a punt socket:

```
punt {  
    socket /run/vpp/punt_socket  
}
```

This socket is used to send messages from DHCPKit back to VPP. You don't need to specify this socket in the DHCPKit configuration, it will learn it through the VPP API.

### Example

```
<listen-vpp /run/vpp/client_socket>  
  namespace-prefix foo  
  
  <vpp-interface tap-0 />  
  
  <vpp-interface GigabitEthernet0/1/2>  
    reply-from fe80::1  
    link-address 2001:db8::1  
  </vpp-interface>  
</listen-vpp>
```

### Section parameters

**mark (multiple allowed)** Every incoming request can be marked with different tags. That way you can handle messages differently based on i.e. which listener they came in on. Every listener can set one or more marks. Also see the marked-with filter.

**Default:** “unmarked”

**namespace-prefix** Namespace prefix for the API. When specifying a prefix in the VPP startup configuration:

```
api-segment {  
    prefix foo  
}
```

then specify *foo* here.

**Example:** “namespace-prefix foo”



**api-definitions** Path to the JSON files that define the API. If left empty the default path for your system will be used.

**Example:** “api-definitions /usr/share/vpp/api”

### Possible sub-section types

*Vpp-interface* (page 3) (required, multiple allowed) VPP Interface to listen on

## dhcpkit\_vpp package

### Subpackages

**dhcpkit\_vpp.listeners package**

#### Subpackages

**dhcpkit\_vpp.listeners.vpp package**

Factory for the implementation of a listener on a VPP punt socket

**exception** `dhcpkit_vpp.listeners.vpp.UnknownVPPAction`

Bases: `dhcpkit_vpp.listeners.vpp.UnwantedVPPMessage` (page 5)

Signal that this message is incomplete because it contained an unknown VPP action value.

**exception** `dhcpkit_vpp.listeners.vpp.UnknownVPPInterface`

Bases: `dhcpkit_vpp.listeners.vpp.UnwantedVPPMessage` (page 5)

Signal that this message is incomplete because it came from an unknown VPP interface.

**exception** `dhcpkit_vpp.listeners.vpp.UnwantedVPPMessage`

Bases: `dhcpkit.ipv6.server.listeners.IgnoreMessage`

This is a message that we don't want

### Submodules

**dhcpkit\_vpp.listeners.vpp.config module**

**dhcpkit\_vpp.listeners.vpp.vpp module**

**dhcpkit\_vpp.listeners.vpp.vpp\_interface module**

**dhcpkit\_vpp.protocols package**

Classes and constants for protocol implementations

**class** `dhcpkit_vpp.protocols.Layer2Frame`

Bases: `dhcpkit.protocol_element.ProtocolElement`

Base class for layer 2 frames

**class** `dhcpkit_vpp.protocols.Layer3Packet`

Bases: `dhcpkit.protocol_element.ProtocolElement`

Base class for layer 3 packets

**get\_pseudo\_header** (*for\_payload: dhcpkit\_vpp.protocols.Layer4Protocol*) → bytes  
Return the pseudo header for this protocol

**Parameters for\_payload** – Get the pseudo header for the given layer 4 protocol

**Returns** The pseudo header bytes

**class** dhcpkit\_vpp.protocols.**Layer4Protocol**

Bases: dhcpkit.protocol\_element.ProtocolElement

Base class for layer 4 protocols

**length**

Return the length of this protocol+payload

**Returns** The length

**protocol\_number = 0**

**save** (*zero\_checksum: bool = False, recalculate\_checksum\_for: typing.Union[dhcpkit\_vpp.protocols.Layer3Packet, NoneType] = None*) → bytearray  
Save the internal state of this object as a buffer.

**Parameters**

- **zero\_checksum** – Save with zeroes where the checksum should be
- **recalculate\_checksum\_for** – Recalculate the checksum for the given layer 3 packet headers

**Returns** The buffer with the data from this element

## Submodules

### dhcpkit\_vpp.protocols.layer2 module

Classes and constants for layer 2 frames

**class** dhcpkit\_vpp.protocols.layer2.**Ethernet** (*destination: bytes = b'x00x00x00x00x00x00', source: bytes = b'x00x00x00x00x00x00', ethertype: int = 0, payload: dhcpkit.protocol\_element.ProtocolElement = None*)

Bases: dhcpkit\_vpp.protocols.Layer2Frame (page 5)

The class for ethernet frames.

**classmethod determine\_class** (*buffer: bytes, offset: int = 0*) → type  
Return the appropriate class to parse this element with.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

**Returns** The best known class for this data

**display\_destination** () → dhcpkit.protocol\_element.ElementDataRepresentation  
Nicer representation of destination :return: Representation of destination

**display\_ethertype** () → dhcpkit.protocol\_element.ElementDataRepresentation  
Nicer representation of ethertype :return: Representation of ethertype

**display\_source** () → dhcpkit.protocol\_element.ElementDataRepresentation  
Nicer representation of source :return: Representation of source

**load\_from** (*buffer: bytes, offset: int = 0, length: int = None*) → int

Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

**Returns** The number of bytes used from the buffer

**save** () → bytes

Save the internal state of this object as a buffer.

**Returns** The buffer with the data from this element

**validate** ()

Validate that the contents of this object conform to protocol specs.

## dhcplib\_vpp.protocols.layer3 module

Classes and constants for layer 3 protocols

```
class dhcplib_vpp.protocols.layer3.IPv6(traffic_class: int = 0, flow_label: int = 0,
                                         next_header: int = 0, hop_limit: int = 0,
                                         source: ipaddress.IPv6Address = None, destination: ipaddress.IPv6Address = None, payload: dhcplib.protocol_element.ProtocolElement = None)
```

Bases: *dhcplib\_vpp.protocols.Layer3Packet* (page 5)

The class for IPv6 packets.

**classmethod determine\_class** (*buffer: bytes, offset: int = 0*) → type

Return the appropriate class to parse this element with.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

**Returns** The best known class for this data

**get\_pseudo\_header** (*l4\_payload: dhcplib\_vpp.protocols.Layer4Protocol*) → bytes

Return the pseudo header for this protocol

**Parameters** **l4\_payload** – The payload protocol to calculate the pseudo header for

**Returns** The pseudo header bytes

**load\_from** (*buffer: bytes, offset: int = 0, length: int = None*) → int

Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

**Returns** The number of bytes used from the buffer

**save** () → bytearray

Save the internal state of this object as a buffer.

**Returns** The buffer with the data from this element

**validate()**

Validate that the contents of this object conform to protocol specs.

**class** `dhcpkit_vpp.protocols.layer3.UnknownLayer3Packet` (*data: bytes = b''*)

Bases: `dhcpkit_vpp.protocols.Layer3Packet` (page 5), `dhcpkit.protocol_element.UnknownProtocolElement`

A layer 3 packet of unknown type

**classmethod** **determine\_class** (*buffer: bytes, offset: int = 0*) → type

Return the appropriate class to parse this element with.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

**Returns** The best known class for this data

**get\_pseudo\_header** (*for\_payload: dhcpkit\_vpp.protocols.Layer4Protocol*) → bytes

We don't have a pseudo header

**Parameters** **for\_payload** – Get the pseudo header for the given layer 4 protocol

**Returns** The pseudo header

## dhcpkit\_vpp.protocols.layer3\_registry module

The protocol layer 3 registry

**class** `dhcpkit_vpp.protocols.layer3_registry.ProtocolLayer3Registry`

Bases: `dhcpkit.registry.Registry`

Registry for Protocols

**entry\_point** = 'dhcpkit\_vpp.protocols.layer3'

## dhcpkit\_vpp.protocols.layer4 module

Classes and constants for layer 4 protocols

**class** `dhcpkit_vpp.protocols.layer4.UDP` (*source\_port: int = 0, destination\_port: int = 0, checksum: int = 0, payload: bytes = b''*)

Bases: `dhcpkit_vpp.protocols.Layer4Protocol` (page 6)

The class for UDP packets.

**calculate\_checksum** (*l3\_packet: dhcpkit\_vpp.protocols.Layer3Packet*)

Calculate the checksum based on the current payload and the provided layer 3 packet.

**Parameters** **l3\_packet** – The layer 3 packet that contains this UDP message

**Returns** The calculated checksum

**classmethod** **determine\_class** (*buffer: bytes, offset: int = 0*) → type

Return the appropriate class to parse this element with.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

**Returns** The best known class for this data

**length**

Return the length of this protocol+payload

**Returns** The length

**load\_from** (*buffer: bytes, offset: int = 0, length: int = None*) → int

Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

**Returns** The number of bytes used from the buffer

**protocol\_number = 17**

**save** (*zero\_checksum: bool = False, recalculate\_checksum\_for: dhcp-kit\_vpp.protocols.Layer3Packet = None*) → bytearray  
Save the internal state of this object as a buffer.

**Parameters**

- **zero\_checksum** – Save with zeroes where the checksum should be
- **recalculate\_checksum\_for** – Recalculate the checksum for the given layer 3 packet headers

**Returns** The buffer with the data from this element

**validate** ()

Validate that the contents of this object conform to protocol specs.

**class** dhcpkit\_vpp.protocols.layer4.**UnknownLayer4Protocol** (*data: bytes = b''*)  
Bases: `dhcpkit_vpp.protocols.Layer4Protocol` (page 6), `dhcpkit.protocol_element.UnknownProtocolElement`

A layer 3 packet of unknown type

**classmethod determine\_class** (*buffer: bytes, offset: int = 0*) → type

Return the appropriate class to parse this element with.

**Parameters**

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

**Returns** The best known class for this data

**length**

The length of our data

**Returns** The length

**save** (*zero\_checksum: bool = False, recalculate\_checksum\_for: dhcp-kit\_vpp.protocols.Layer3Packet = None*) → bytearray  
Save the internal state of this object as a buffer.

**Parameters**

- **zero\_checksum** – Save with zeroes where the checksum should be
- **recalculate\_checksum\_for** – Recalculate the checksum for the given layer 3 packet headers

**Returns** The buffer with the data from this element

## dhcpkit\_vpp.protocols.layer4\_registry module

The protocol layer 4 registry

```
class dhcpkit_vpp.protocols.layer4_registry.ProtocolLayer4Registry
    Bases: dhcpkit.registry.Registry
    Registry for Protocols
    entry_point = 'dhcpkit_vpp.protocols.layer4'
```

## dhcpkit\_vpp.protocols.utils module

```
dhcpkit_vpp.protocols.utils.ones_complement_checksum(msg: typing.Union[bytes,
                                                                    bytearray])
    Calculate the 16-bit one's complement of the one's complement sum of a message.
    Parameters msg – The message
    Returns The checksum
```

## dhcpkit\_vpp.tests package

All the unit tests go here

## Subpackages

### dhcpkit\_vpp.tests.protocols package

```
class dhcpkit_vpp.tests.protocols.FrameTestCase (methodName='runTest')
    Bases: unittest.case.TestCase
    check_unsigned_integer_property (property_name: str, size: int = None)
        Perform basic verification of validation of an unsigned integer
        Parameters
            • property_name – The property under test
            • size – The number of bits of this integer field
    parse_packet ()
    setUp ()
    test_class ()
    test_length ()
    test_parse ()
    test_save_fixture ()
    test_save_parsed ()
    test_validate ()
```

## Submodules

### `dhcpkit_vpp.tests.protocols.test_base_classes` module

```
class dhcpkit_vpp.tests.protocols.test_base_classes.Layer3PacketTestCase (methodName='runTest')
    Bases: unittest.case.TestCase

    test_abstract_get_pseudo_header ()

class dhcpkit_vpp.tests.protocols.test_base_classes.Layer4ProtocolTestCase (methodName='runTest')
    Bases: unittest.case.TestCase

    test_abstract_length ()

    test_abstract_save ()
```

### `dhcpkit_vpp.tests.protocols.test_layer2` module

Test whether layer 2 parsing and generating works

```
class dhcpkit_vpp.tests.protocols.test_layer2.Layer2FrameTestCase (methodName='runTest')
    Bases: dhcpkit_vpp.tests.protocols.FrameTestCase (page 10)

    setUp ()

    test_display_destination ()

    test_display_ethertype ()

    test_display_source ()

    test_ethernet_length ()

    test_13_payload_type ()

    test_unknown_payload_type ()

    test_validate_destination ()

    test_validate_ethertype ()

    test_validate_payload ()

    test_validate_source ()
```

### `dhcpkit_vpp.tests.protocols.test_layer3` module

Test whether layer 3 parsing and generating works

```
class dhcpkit_vpp.tests.protocols.test_layer3.IPv6TestCase (methodName='runTest')
    Bases: dhcpkit_vpp.tests.protocols.FrameTestCase (page 10)

    setUp ()

    test_ipv6_length ()

    test_14_payload_type ()

    test_protocol_version ()

    test_trailing_data ()

    test_unknown_payload_type ()

    test_validate_destination ()

    test_validate_flow_label ()
```

```
test_validate_hop_limit()
test_validate_next_header()
test_validate_payload()
test_validate_source()
test_validate_traffic_class()
```

```
class dhcpkit_vpp.tests.protocols.test_layer3.UnknownLayer3PacketTestCase (methodName='runTest')
    Bases: dhcpkit_vpp.tests.protocols.FrameTestCase (page 10)
    setUp()
    test_pseudo_header()
```

## dhcpkit\_vpp.tests.protocols.test\_layer4 module

Test whether layer 4 parsing and generating works

```
class dhcpkit_vpp.tests.protocols.test_layer4.UDPTestCase (methodName='runTest')
    Bases: dhcpkit_vpp.tests.protocols.FrameTestCase (page 10)
    setUp()
    test_checksum_calculation()
    test_save_with_checksum_calculation()
    test_save_zero_checksum()
    test_udp_length()
    test_validate_checksum()
    test_validate_destination_port()
    test_validate_payload()
    test_validate_source_port()
```

```
class dhcpkit_vpp.tests.protocols.test_layer4.UnknownLayer4ProtocolTestCase (methodName='runTest')
    Bases: dhcpkit_vpp.tests.protocols.FrameTestCase (page 10)
    setUp()
    test_length()
```

## Submodules

dhcpkit\_vpp.vpp\_papi module

## Changes per version

### 1.0.1 - 2017-06-21

#### Fixes

- Package missing component.xml



## 1.0.0 - 2017-06-21

### New features

- Initial release

## Applicable copyright licences

### DHCPKit License

Copyright (c) 2017, S.J.M. Steffann

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

### VPP-PAPI module license

Copyright (c) 2016 Cisco and/or its affiliates. Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.



### d

- `dhcpkit_vpp`, [5](#)
- `dhcpkit_vpp.listeners`, [5](#)
- `dhcpkit_vpp.listeners.vpp`, [5](#)
- `dhcpkit_vpp.protocols`, [5](#)
- `dhcpkit_vpp.protocols.layer2`, [6](#)
- `dhcpkit_vpp.protocols.layer3`, [7](#)
- `dhcpkit_vpp.protocols.layer3_registry`,  
[8](#)
- `dhcpkit_vpp.protocols.layer4`, [8](#)
- `dhcpkit_vpp.protocols.layer4_registry`,  
[10](#)
- `dhcpkit_vpp.protocols.utils`, [10](#)
- `dhcpkit_vpp.tests`, [10](#)
- `dhcpkit_vpp.tests.protocols`, [10](#)
- `dhcpkit_vpp.tests.protocols.test_base_classes`,  
[11](#)
- `dhcpkit_vpp.tests.protocols.test_layer2`,  
[11](#)
- `dhcpkit_vpp.tests.protocols.test_layer3`,  
[11](#)
- `dhcpkit_vpp.tests.protocols.test_layer4`,  
[12](#)



## C

calculate\_checksum() (dhcp-kit\_vpp.protocols.layer4.UDP method), 8

check\_unsigned\_integer\_property() (dhcp-kit\_vpp.tests.protocols.FrameTestCase method), 10

## D

determine\_class() (dhcp-kit\_vpp.protocols.layer2.Ethernet class method), 6

determine\_class() (dhcpkit\_vpp.protocols.layer3.IPv6 class method), 7

determine\_class() (dhcp-kit\_vpp.protocols.layer3.UnknownLayer3Packet class method), 8

determine\_class() (dhcpkit\_vpp.protocols.layer4.UDP class method), 8

determine\_class() (dhcp-kit\_vpp.protocols.layer4.UnknownLayer4Protocol class method), 9

dhcpkit\_vpp (module), 5

dhcpkit\_vpp.listeners (module), 5

dhcpkit\_vpp.listeners.vpp (module), 5

dhcpkit\_vpp.protocols (module), 5

dhcpkit\_vpp.protocols.layer2 (module), 6

dhcpkit\_vpp.protocols.layer3 (module), 7

dhcpkit\_vpp.protocols.layer3\_registry (module), 8

dhcpkit\_vpp.protocols.layer4 (module), 8

dhcpkit\_vpp.protocols.layer4\_registry (module), 10

dhcpkit\_vpp.protocols.utils (module), 10

dhcpkit\_vpp.tests (module), 10

dhcpkit\_vpp.tests.protocols (module), 10

dhcpkit\_vpp.tests.protocols.test\_base\_classes (module), 11

dhcpkit\_vpp.tests.protocols.test\_layer2 (module), 11

dhcpkit\_vpp.tests.protocols.test\_layer3 (module), 11

dhcpkit\_vpp.tests.protocols.test\_layer4 (module), 12

display\_destination() (dhcp-kit\_vpp.protocols.layer2.Ethernet method), 6

display\_ethertype() (dhcp-kit\_vpp.protocols.layer2.Ethernet method),

6

display\_source() (dhcp-kit\_vpp.protocols.layer2.Ethernet method), 6

## E

entry\_point (dhcpkit\_vpp.protocols.layer3\_registry.ProtocolLayer3Registry attribute), 8

entry\_point (dhcpkit\_vpp.protocols.layer4\_registry.ProtocolLayer4Registry attribute), 10

Ethernet (class in dhcpkit\_vpp.protocols.layer2), 6

## F

FrameTestCase (class in dhcpkit\_vpp.tests.protocols), 10

## G

get\_pseudo\_header() (dhcp-kit\_vpp.protocols.layer3.IPv6 method), 7

get\_pseudo\_header() (dhcp-kit\_vpp.protocols.layer3.UnknownLayer3Packet method), 8

get\_pseudo\_header() (dhcp-kit\_vpp.protocols.Layer3Packet method), 5

## I

IPv6 (class in dhcpkit\_vpp.protocols.layer3), 7

IPv6TestCase (class in dhcp-kit\_vpp.tests.protocols.test\_layer3), 11

## L

Layer2Frame (class in dhcpkit\_vpp.protocols), 5

Layer2FrameTestCase (class in dhcp-kit\_vpp.tests.protocols.test\_layer2), 11

Layer3Packet (class in dhcpkit\_vpp.protocols), 5

Layer3PacketTestCase (class in dhcp-kit\_vpp.tests.protocols.test\_base\_classes), 11

Layer4Protocol (class in dhcpkit\_vpp.protocols), 6

Layer4ProtocolTestCase (class in dhcp-kit\_vpp.tests.protocols.test\_base\_classes), 11

length (dhcpkit\_vpp.protocols.layer4.UDP attribute), 8  
length (dhcpkit\_vpp.protocols.layer4.UnknownLayer4Protocol attribute), 9  
length (dhcpkit\_vpp.protocols.Layer4Protocol attribute), 6  
load\_from() (dhcpkit\_vpp.protocols.layer2.Ethernet method), 6  
load\_from() (dhcpkit\_vpp.protocols.layer3.IPv6 method), 7  
load\_from() (dhcpkit\_vpp.protocols.layer4.UDP method), 9

**O**

ones\_complement\_checksum() (in module dhcpkit\_vpp.protocols.utils), 10

**P**

parse\_packet() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
protocol\_number (dhcpkit\_vpp.protocols.layer4.UDP attribute), 9  
protocol\_number (dhcpkit\_vpp.protocols.Layer4Protocol attribute), 6  
ProtocolLayer3Registry (class in dhcpkit\_vpp.protocols.layer3\_registry), 8  
ProtocolLayer4Registry (class in dhcpkit\_vpp.protocols.layer4\_registry), 10

**S**

save() (dhcpkit\_vpp.protocols.layer2.Ethernet method), 7  
save() (dhcpkit\_vpp.protocols.layer3.IPv6 method), 7  
save() (dhcpkit\_vpp.protocols.layer4.UDP method), 9  
save() (dhcpkit\_vpp.protocols.layer4.UnknownLayer4Protocol method), 9  
save() (dhcpkit\_vpp.protocols.Layer4Protocol method), 6  
setUp() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
setUp() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
setUp() (dhcpkit\_vpp.tests.protocols.test\_layer3.IPv6TestCase method), 11  
setUp() (dhcpkit\_vpp.tests.protocols.test\_layer3.UnknownLayer3PacketTestCase method), 12  
setUp() (dhcpkit\_vpp.tests.protocols.test\_layer4.UDPTestCase method), 12  
setUp() (dhcpkit\_vpp.tests.protocols.test\_layer4.UnknownLayer4ProtocolTestCase method), 12

**T**

test\_abstract\_get\_pseudo\_header() (dhcpkit\_vpp.tests.protocols.test\_base\_classes.Layer3PacketTestCase method), 11  
test\_abstract\_length() (dhcpkit\_vpp.tests.protocols.test\_base\_classes.Layer4ProtocolTestCase method), 11  
test\_abstract\_save() (dhcpkit\_vpp.tests.protocols.test\_base\_classes.Layer4ProtocolTestCase method), 11  
test\_checksum\_calculation() (dhcpkit\_vpp.tests.protocols.test\_layer4.UDPTestCase method), 12  
test\_class() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
test\_display\_destination() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
test\_display\_ethertype() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
test\_display\_source() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
test\_ethernet\_length() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
test\_ipv6\_length() (dhcpkit\_vpp.tests.protocols.test\_layer3.IPv6TestCase method), 11  
test\_l3\_payload\_type() (dhcpkit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase method), 11  
test\_l4\_payload\_type() (dhcpkit\_vpp.tests.protocols.test\_layer3.IPv6TestCase method), 11  
test\_length() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
test\_length() (dhcpkit\_vpp.tests.protocols.test\_layer4.UnknownLayer4ProtocolTestCase method), 12  
test\_parse() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
test\_protocol\_version() (dhcpkit\_vpp.tests.protocols.test\_layer3.IPv6TestCase method), 11  
test\_pseudo\_header() (dhcpkit\_vpp.tests.protocols.test\_layer3.UnknownLayer3PacketTestCase method), 12  
test\_save\_fixture() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
test\_save\_parsed() (dhcpkit\_vpp.tests.protocols.FrameTestCase method), 10  
test\_save\_with\_checksum\_calculation() (dhcpkit\_vpp.tests.protocols.test\_layer4.UDPTestCase method), 12  
test\_save\_zero\_checksum() (dhcpkit\_vpp.tests.protocols.test\_layer4.UDPTestCase method), 12  
test\_trailing\_data() (dhcpkit\_vpp.tests.protocols.test\_layer3.IPv6TestCase method), 12

method), 11

test\_udp\_length() (dhcp- UDP (class in dhcpkit\_vpp.protocols.layer4), 8  
kit\_vpp.tests.protocols.test\_layer4.UDPTestCase (class in dhcp-  
method), 12 kit\_vpp.tests.protocols.test\_layer4), 12

test\_unknown\_payload\_type() (dhcp- UnknownLayer3Packet (class in dhcp-  
kit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase kit\_vpp.protocols.layer3), 8  
method), 11 UnknownLayer3PacketTestCase (class in dhcp-  
test\_unknown\_payload\_type() (dhcp- kit\_vpp.tests.protocols.test\_layer3), 12  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase UnknownLayer4Protocol (class in dhcp-  
method), 11 kit\_vpp.protocols.layer4), 9

test\_validate() (dhcp- UnknownLayer4ProtocolTestCase (class in dhcp-  
kit\_vpp.tests.protocols.FrameTestCase kit\_vpp.tests.protocols.test\_layer4), 12  
method), 10 UnknownVPPAction, 5

test\_validate\_checksum() (dhcp- UnknownVPPInterface, 5  
kit\_vpp.tests.protocols.test\_layer4.UDPTestCase UnwantedVPPMessage, 5  
method), 12

test\_validate\_destination() (dhcp- V  
kit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase validate() (dhcpkit\_vpp.protocols.layer2.Ethernet  
method), 11 method), 7

test\_validate\_destination() (dhcp- validate() (dhcpkit\_vpp.protocols.layer3.IPv6 method),  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase 8  
method), 11

test\_validate\_destination\_port() (dhcp- validate() (dhcpkit\_vpp.protocols.layer4.UDP method),  
kit\_vpp.tests.protocols.test\_layer4.UDPTestCase 9  
method), 12

test\_validate\_ethertype() (dhcp-  
kit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase  
method), 11

test\_validate\_flow\_label() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 11

test\_validate\_hop\_limit() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 11

test\_validate\_next\_header() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 12

test\_validate\_payload() (dhcp-  
kit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase  
method), 11

test\_validate\_payload() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 12

test\_validate\_payload() (dhcp-  
kit\_vpp.tests.protocols.test\_layer4.UDPTestCase  
method), 12

test\_validate\_source() (dhcp-  
kit\_vpp.tests.protocols.test\_layer2.Layer2FrameTestCase  
method), 11

test\_validate\_source() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 12

test\_validate\_source\_port() (dhcp-  
kit\_vpp.tests.protocols.test\_layer4.UDPTestCase  
method), 12

test\_validate\_traffic\_class() (dhcp-  
kit\_vpp.tests.protocols.test\_layer3.IPv6TestCase  
method), 12