API Documentation

API Documentation

December 30, 2013

Contents

1 Package killerbee

1.1 Modules

1.2 Functions

getKillerBee(*channel*)

Returns an instance of a KillerBee device, setup on the given channel. Error handling for KillerBee creation and setting of the channel is wrapped and will raise an Exception().

Return Value

A KillerBee instance initialized to the given channel.

kb_dev_list(*vendor*=`None`, *product*=`None`)

Deprecated. Use show_dev or call kbutils.devlist.

show_dev(*vendor*=`None`, *product*=`None`, *gps*=`None`, *include*=`None`)

A basic function to output the device listing. Placed here for reuse, as many tool scripts were implementing it.

Parameters

`gps`: Provide device names in this argument (previously known as 'gps') which you wish to not be enumerated. Aka, exclude these items.

`include`: Provide device names in this argument if you would like only these to be enumerated. Aka, include only these items.

1.3 Variables

Name	Description
__package__	**Value:** `'killerbee'`

1.4 Class KillerBee

1.4.1 Methods

__init__(*self*, *device*=`None`, *datasource*=`None`, *gps*=`None`)

Instantiates the KillerBee class.

Parameters

`device:` Device identifier, either USB vendor:product, serial device node, or IP address

(type=String)

`datasource:` A known datasource type that is used by dblog to record how the data was captured.

(type=String)

`gps:` Optional serial device identifier for an attached GPS unit. If provided, or if global variable has previously been set, KillerBee skips that device in initalization process.

(type=String)

Return Value

None

(type=None)

dev_list(*self*, *vendor*=`None`, *product*=`None`)

Deprecated in class, use kbutils.devlist() instead.

get_dev_info(*self*)

Returns device information in a list identifying the device. Implemented by the loaded driver.

Return Value

List of 3 strings identifying device.

(type=List)

close(*self*)

Closes the device out.

Return Value

None

(type=None)

check_capability(*self*, *capab*)

Uses the specified capability to determine if the opened device is supported. Returns True when supported, else False.

Return Value

Boolean

get_capabilities(*self*)

Returns a list of capability information for the device.

Return Value

Capability information for the opened device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel. Currently, support is limited to 2.4 GHz channels 11 - 26.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

is_valid_channel(*self*, *channel*)

Based on sniffer capabilities, return if this is an OK channel number.

Return Value

Boolean

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet`: Packet contents to transmit, without FCS.
(type=String)

`channel`: Sets the channel, optional
(type=Integer)

`count`: Transmits a specified number of frames, def=1
(type=Integer)

`delay`: Delay between each frame, def=1
(type=Float)

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns packet data as a string, else None.

Parameters

`timeout`: Timeout to wait for packet reception in usec
(type=Integer)

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

jammer_on(*self*, *channel*=`None`)

Attempts reflexive jamming on all 802.15.4 frames. Targeted frames must be >12 bytes for reliable jamming in current firmware.

Parameters

`channel`: Sets the channel, optional.
(type=Integer)

Return Value

None

1.4.2 Properties

Name	Description
channel	Getter function for the channel that was last set on the device.

2 Module killerbee.GoodFET

2.1 Functions

getClient(*name*=`'GoodFET'`)

2.2 Variables

Name	Description
fmt	**Value:** `('B', '<H', None, '<L')`
__package__	**Value:** `'killerbee'`

2.3 Class SymbolTable

GoodFET Symbol Table

2.3.1 Methods

__init__(*self*, **args*, ***kargs*)

get(*self*, *name*)

define(*self*, *adr*, *name*, *comment*=`''`, *memory*=`'vn'`, *size*=`16`)

2.3.2 Class Variables

Name	Description
db	**Value:** `sqlite3.connect(":memory:")`

2.4 Class GoodFETbtser

py-bluez class for emulating py-serial.

2.4.1 Methods

__init__(*self*, *btaddr*)

write(*self*, *msg*)

Send traffic.

read(*self*, *length*)

Read traffic.

2.5 Class GoodFET

Known Subclasses: killerbee.GoodFETAVR.GoodFETAVR, killerbee.GoodFETCCSPI.GoodFETCCSPI

GoodFET Client Library

2.5.1 Methods

__init__(*self*, **args*, ***kargs*)

getConsole(*self*)

name2adr(*self*, *name*)

timeout(*self*)

serInit(*self*, *port*=`None`, *timeout*=`2`, *attemptlimit*=`None`)

Open a serial port of some kind.

btInit(*self*, *port*, *timeout*, *attemptlimit*)

Open a bluetooth port.

pyserInit(*self*, *port*, *timeout*, *attemptlimit*)

Open the serial port

serClose(*self*)

telosSetSCL(*self*, *level*)

Helper function for support of the TelosB platform.

telosSetSDA(*self*, *level*)

Helper function for support of the TelosB platform.

telosI2CStart(*self*)

Helper function for support of the TelosB platform.

telosI2CStop(*self*)

Helper function for support of the TelosB platform.

telosI2CWriteBit(*self*, *bit*)

Helper function for support of the TelosB platform.

telosI2CWriteByte(*self*, *byte*)

Helper function for support of the TelosB platform.

telosI2CWriteCmd(*self*, *addr*, *cmdbyte*)

Helper function for support of the TelosB platform.

bslResetZ1(*self*, *invokeBSL*=`0`)

Helper function for support of the Z1 mote platform. Applies BSL entry sequence on RST/NMI and TEST/VPP pins. By now only BSL mode is accessed.

Parameters

`invokeBSL`: 1 for a complete sequence, or 0 to only access RST/NMI pin

(type=Integer)

writepicROM(*self*, *address*, *data*)

Writes data to @address

readpicROM(*self*, *address*)

reads a byte from @address

picROMclock(*self*, *masterout*, *slow*=`True`)

picROMfastclock(*self*, *masterout*)

telosBReset(*self*, *invokeBSL*=`0`)

Helper function for support of the TelosB platform.

getbuffer(*self*, *size*=`7168`)

writecmd(*self*, *app*, *verb*, *count*=`0`, *data*=`[]`)

Write a command and some data to the GoodFET.

readcmd(*self*)

Read a reply from the GoodFET.

glitchApp(*self*, *app*)

Glitch into a device by its application.

glitchVerb(*self*, *app*, *verb*, *data*)

Glitch during a transaction.

glitchstart(*self*)

Glitch into the AVR application.

glitchstarttime(*self*)

Measure the timer of the START verb.

glitchTime(*self*, *app*, *verb*, *data*)

Time the execution of a verb.

glitchVoltages(*self*, *low*=2176, *high*=4095)

Set glitching voltages. (0x0fff is max.)

glitchRate(*self*, *count*=2048)

Set glitching count period.

silent(*self*, *s*=0)

Transmissions halted when 1.

mon_connected(*self*)

Announce to the monitor that the connection is good.

out(*self*, *byte*)

Write a byte to P5OUT.

dir(*self*, *byte*)

Write a byte to P5DIR.

call(*self*, *adr*)

Call to an address.

execute(*self*, *code*)

Execute supplied code.

MONpeek8(*self*, *address*)

Read a byte of memory from the monitor.

MONpeek16(*self*, *address*)

Read a word of memory from the monitor.

peek(*self*, *address*)

Read a word of memory from the monitor.

eeprompeek(*self*, *address*)

Read a word of memory from the monitor.

peekbysym(*self*, *name*)

Read a value by its symbol name.

pokebysym(*self*, *name*, *val*)

Write a value by its symbol name.

pokebyte(*self*, *address*, *value*, *memory*=`'vn'`)

Set a byte of memory by the monitor.

poke16(*self*, *address*, *value*)

Set a word of memory by the monitor.

MONpoke16(*self*, *address*, *value*)

Set a word of memory by the monitor.

setsecret(*self*, *value*)

Set a secret word for later retreival. Used by glitcher.

getsecret(*self*)

Get a secret word. Used by glitcher.

dumpmem(*self*, *begin*, *end*)

monitor_ram_pattern(*self*)

Overwrite all of RAM with 0xBEEF.

monitor_ram_depth(*self*)

Determine how many bytes of RAM are unused by looking for 0xBEEF..

setBaud(*self*, *baud*)

Change the baud rate. TODO fix this.

readbyte(*self*)

findbaud(*self*)

monitortest(*self*)

Self-test several functions through the monitor.

monitorecho(*self*)

monitor_info(*self*)

testleds(*self*)

monitor_list_apps(*self*, *full*=`False`)

monitorclocking(*self*)

Return the 16-bit clocking value.

monitorsetclock(*self*, *clock*)

Set the clocking value.

monitorgetclock(*self*)

Get the clocking value.

infostring(*self*)

lock(*self*)

erase(*self*)

setup(*self*)

start(*self*)

test(*self*)

status(*self*)

halt(*self*)

resume(*self*)

getpc(*self*)

flash(*self*, *file*)

Flash an intel hex file to code memory.

dump(*self*, *file*, *start*=`0`, *stop*=`65535`)

Dump an intel hex file from code memory.

peek32(*self*, *address*, *memory*=`'vn'`)

Peek 32 bits.

peek16(*self*, *address*, *memory*=`'vn'`)

Peek 16 bits of memory.

peek8(*self*, *address*, *memory*=`'vn'`)

Peek a byte of memory.

peekblock(*self*, *address*, *length*, *memory*=`'vn'`)

Return a block of data.

pokeblock(*self*, *address*, *bytes*, *memory*=`'vn'`)

Poke a block of a data into memory at an address.

loadsymbols(*self*)

Load symbols from a file.

2.5.2 Class Variables

Name	Description
besilent	**Value:** `0`
app	**Value:** `0`
verb	**Value:** `0`
count	**Value:** `0`
data	**Value:** `''`
verbose	**Value:** `False`
GLITCHAPP	**Value:** `113`
MONITORAPP	**Value:** `0`
symbols	**Value:** `SymbolTable()`
connected	**Value:** `0`
baudrates	**Value:** `[115200, 9600, 19200, 38400, 57600, 115200]`

3 Module killerbee.GoodFETAVR

3.1 Variables

Name	Description
__package__	**Value:** `'killerbee'`

3.2 Class GoodFETAVR

killerbee.GoodFET.GoodFET
└ **killerbee.GoodFETAVR.GoodFETAVR**

Known Subclasses: killerbee.GoodFETatmel128.GoodFETatmel128rfa1

3.2.1 Methods

setup(*self*)

Move the FET into the AVR application.

Overrides: killerbee.GoodFET.GoodFET.setup

trans(*self*, *data*)

Exchange data by AVR. Input should probably be 4 bytes.

start(*self*)

Start the connection.

Overrides: killerbee.GoodFET.GoodFET.start

forcestart(*self*)

Forcibly start a connection.

erase(*self*)

Erase the target chip.

Overrides: killerbee.GoodFET.GoodFET.erase

lockbits(*self*)

Read the target's lockbits.

setlockbits(*self*, *bits*=`0`)

Read the target's lockbits.

lock(*self*)

Overrides: killerbee.GoodFET.GoodFET.lock

eeprompeek(*self*, *adr*)

Read a byte of the target's EEPROM.

Overrides: killerbee.GoodFET.GoodFET.eeprompeek

flashpeek(*self*, *adr*)

Read a byte of the target's Flash memory.

flashpeekblock(*self*, *adr*)

Read a byte of the target's Flash memory.

eeprompoke(*self*, *adr*, *val*)

Write a byte of the target's EEPROM.

identstr(*self*)

Return an identifying string.

Inherited from killerbee.GoodFET.GoodFET(Section 2.5)

MONpeek16(), MONpeek8(), MONpoke16(), __init__(), bslResetZ1(), btInit(), call(), dir(), dump(), dumpmem(), execute(), findbaud(), flash(), getConsole(), getbuffer(), getpc(), getsecret(), glitchApp(), glitchRate(), glitchTime(), glitchVerb(), glitchVoltages(), glitchstart(), glitchstarttime(), halt(), infostring(), loadsymbols(), mon_connected(), monitor_info(), monitor_list_apps(), monitor_ram_depth(), monitor_ram_pattern(), monitorclocking(), monitorecho(), monitorgetclock(), monitorsetclock(), monitortest(), name2adr(), out(), peek(), peek16(), peek32(), peek8(), peekblock(), peekbysym(), picROMclock(), picROMfastclock(), poke16(), pokeblock(), pokebysym(), pokebyte(), pyserInit(), readbyte(), readcmd(), readpicROM(), resume(), serClose(), serInit(), setBaud(), setsecret(), silent(), status(), telosBReset(), telosI2CStart(), telosI2CStop(), telosI2CWriteBit(), telosI2CWriteByte(), telosI2CWriteCmd(), telosSetSCL(), telosSetSDA(), test(), testleds(), timeout(), writecmd(), writepicROM()

3.2.2 Class Variables

Name	Description
AVRAPP	**Value:** `50`
APP	**Value:** `50`
AVRVendors	**Value:** `{0: 'Locked', 30: 'Atmel'}`

continued on next page

Name	Description
AVRDevices	**Value:** `{257: 'ATmega103', 36865: 'AT90S1200', 36866: 'ATtiny19',...`
Inherited from killerbee.GoodFET.GoodFET (Section 2.5) GLITCHAPP, MONITORAPP, app, baudrates, besilent, connected, count, data, symbols, verb, verbose	

4 Module killerbee.GoodFETCCSPI

4.1 Variables

Name	Description
__package__	**Value:** `'killerbee'`

4.2 Class GoodFETCCSPI

killerbee.GoodFET.GoodFET
killerbee.GoodFETCCSPI.GoodFETCCSPI

4.2.1 Methods

setup(*self*)

Move the FET into the CCSPI application.

Overrides: killerbee.GoodFET.GoodFET.setup

ident(*self*)

identstr(*self*)

trans8(*self*, *byte*)

Read and write 8 bits by CCSPI.

trans(*self*, *data*)

Exchange data by CCSPI.

strobe(*self*, *reg*=`0`)

Strobes a strobe register, returning the status.

CC_RFST_IDLE(*self*)

Switch the radio to idle mode, clearing overflows and errors.

CC_RFST_TX(*self*)

Switch the radio to TX mode.

CC_RFST_RX(*self*)

Switch the radio to RX mode.

CC_RFST_CAL(*self*)

Calibrate strobe the radio.

CC_RFST(*self*, *state*=`0`)

peek(*self*, *reg*, *bytes*=`2`)

Read a CCSPI Register. For long regs, result is flipped.

Overrides: killerbee.GoodFET.GoodFET.peek

poke(*self*, *reg*, *val*, *bytes*=`2`)

Write a CCSPI Register.

status(*self*)

Read the status byte.

Overrides: killerbee.GoodFET.GoodFET.status

RF_setenc(*self*, *code*=`'802.15.4'`)

Set the encoding type.

RF_getenc(*self*)

Get the encoding type.

RF_getrate(*self*)

RF_setrate(*self*, *rate*=`0`)

RF_getsync(*self*)

RF_setsync(*self*, *sync*=`42767`)

Set the SYNC preamble. Use 0xA70F for 0xA7.

RF_setkey(*self*, *key*)

Sets the first key for encryption to the given argument.

RF_setnonce(*self*, *key*)

Sets the first key for encryption to the given argument.

RF_setfreq(*self*, *frequency*)

Set the frequency in Hz.

RF_getfreq(*self*)

Get the frequency in Hz.

RF_setchan(*self*, *channel*)

Set the ZigBee/802.15.4 channel number.

RF_getsmac(*self*)

Return the source MAC address.

RF_setsmac(*self*, *mac*)

Set the source MAC address.

RF_gettmac(*self*)

Return the target MAC address.

RF_settmac(*self*, *mac*)

Set the target MAC address.

RF_getrssi(*self*)

Returns the received signal strength, with a weird offset.

peekram(*self*, *adr*, *count*)

Peeks data from CC2420 RAM.

pokeram(*self*, *adr*, *data*)

Pokes data into CC2420 RAM.

RF_rxpacket(*self*)

Get a packet from the radio. Returns None if none is waiting.

RF_rxpacketrepeat(*self*)

Gets packets from the radio, ignoring all future requests so as not to waste time. Call RF_rxpacket() after this.

RF_rxpacketdec(*self*)

Get and decrypt a packet from the radio. Returns None if none is waiting.

RF_txpacket(*self*, *packet*)

Send a packet through the radio.

RF_reflexjam(*self*, *duration*=`0`)

Place the device into reflexive jamming mode.

RF_reflexjam_autoack(*self*)

Place the device into reflexive jamming mode and that also sends a forged ACK if needed.

RF_modulated_spectrum(*self*)

Hold a carrier wave on the present frequency.

RF_carrier(*self*)

Hold a carrier wave on the present frequency.

RF_promiscuity(*self*, *promiscuous*=`1`)

RF_autocrc(*self*, *autocrc*=`1`)

RF_autoack(*self*, *autoack*=`1`)

RF_setpacketlen(*self*, *len*=`16`)

Set the number of bytes in the expected payload.

RF_getpacketlen(*self*)

Set the number of bytes in the expected payload.

RF_getmaclen(*self*)

Get the number of bytes in the MAC address.

RF_setmaclen(*self*, *len*)

Set the number of bytes in the MAC address.

printpacket(*self*, *packet*, *prefix*=`'#'`)

packet2str(*self*, *packet*, *prefix*=`'#'`)

printdissect(*self*, *packet*)

Inherited from killerbee.GoodFET.GoodFET(Section 2.5)

MONpeek16(), MONpeek8(), MONpoke16(), __init__(), bslResetZ1(), btInit(), call(), dir(), dump(), dumpmem(), eeprompeek(), erase(), execute(), findbaud(), flash(), getConsole(), getbuffer(), getpc(), getsecret(), glitchApp(), glitchRate(), glitchTime(), glitchVerb(), glitchVoltages(), glitchstart(), glitchstarttime(), halt(), infostring(), loadsymbols(), lock(), mon_connected(), monitor_info(), monitor_list_apps(), monitor_ram_depth(), monitor_ram_pattern(), monitorclocking(), monitorecho(), monitorgetclock(), monitorsetclock(), monitortest(), name2adr(), out(), peek16(), peek32(), peek8(), peekblock(), peekbysym(), picROMclock(), picROMfastclock(), poke16(), pokeblock(), pokebysym(), pokebyte(), pyserInit(), readbyte(), readcmd(), readpicROM(), resume(), serClose(), serInit(), setBaud(), setsecret(), silent(), start(), telosBReset(), telosI2CStart(), telosI2CStop(), telosI2CWriteBit(), telosI2CWriteByte(), telosI2CWriteCmd(), telosSetSCL(), telosSetSDA(), test(), testleds(), timeout(), writecmd(), writepicROM()

4.2.2 Class Variables

Name	Description
CCSPIAPP	**Value:** `81`
CCversions	**Value:** `{9021: 'CC2420'}`
lastpacket	**Value:** `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16...`
packetlen	**Value:** `16`
maclen	**Value:** `5`
Inherited from killerbee.GoodFET.GoodFET (Section 2.5) GLITCHAPP, MONITORAPP, app, baudrates, besilent, connected, count, data, symbols, verb, verbose	

5 Module killerbee.GoodFETatmel128

5.1 Variables

Name	Description
fmt	**Value:** `('B', '<H', None, '<L')`
__package__	**Value:** `'killerbee'`

5.2 Class GoodFETatmel128rfa1

killerbee.GoodFET.GoodFET

killerbee.GoodFETAVR.GoodFETAVR

killerbee.GoodFETatmel128.GoodFETatmel128rfa1

5.2.1 Methods

serInit(*self*, *port*=`None`, *timeout*=`2`, *attemptlimit*=`None`)

Open a serial port of some kind.

Overrides: killerbee.GoodFET.GoodFET.serInit extit(inherited documentation)

pyserInit(*self*, *port*, *timeout*, *attemptlimit*)

Open the serial port

Overrides: killerbee.GoodFET.GoodFET.pyserInit

serClose(*self*)

Overrides: killerbee.GoodFET.GoodFET.serClose

writecmd(*self*, *app*, *verb*, *count*=`0`, *data*=`[]`)

Write a command and some data to the GoodFET.

Overrides: killerbee.GoodFET.GoodFET.writecmd

readcmd(*self*)

Read a reply from the GoodFET.

Overrides: killerbee.GoodFET.GoodFET.readcmd

RF_setchannel(*self*, *chan*)

peek(*self*, *reg*, *bytes*=`1`)

Read a Register.

Overrides: killerbee.GoodFET.GoodFET.peek

poke(*self*, *reg*, *val*, *bytes*=`1`)

Write an Register.

setup(*self*)

Move the FET into the AVR application.

Overrides: killerbee.GoodFET.GoodFET.setup

RF_setup(*self*)

RF_rxpacket(*self*)

Get a packet from the radio. Returns None if none is waiting.

RF_txpacket(*self*, *payload*)

RF_getrssi(*self*)

Returns the received signal strength

RF_enable_AACK(*self*, *enable*=`True`)

RF_autocrc(*self*, *autocrc*=`1`)

Inherited from killerbee.GoodFETAVR.GoodFETAVR(Section 3.2)

eeprompeek(), eeprompoke(), erase(), flashpeek(), flashpeekblock(), forcestart(), identstr(), lock(), lockbits(), setlockbits(), start(), trans()

Inherited from killerbee.GoodFET.GoodFET(Section 2.5)

MONpeek16(), MONpeek8(), MONpoke16(), ___init___(), bslResetZ1(), btInit(), call(), dir(), dump(), dumpmem(), execute(), findbaud(), flash(), getConsole(), getbuffer(), getpc(), getsecret(), glitchApp(), glitchRate(), glitchTime(), glitchVerb(), glitchVoltages(), glitchstart(), glitchstarttime(), halt(), infostring(), loadsymbols(), mon_connected(), monitor_info(), monitor_list_apps(), monitor_ram_depth(), monitor_ram_pattern(), monitorclocking(), monitorecho(), monitorgetclock(), monitorsetclock(), monitortest(), name2adr(), out(), peek16(), peek32(), peek8(), peekblock(), peekbysym(), picROMclock(), picROMfastclock(), poke16(), pokeblock(), pokebysym(), pokebyte(), readbyte(), readpicROM(), resume(), setBaud(), setsecret(), silent(), status(), telosBReset(), telosI2CStart(), telosI2CStop(), telosI2CWriteBit(), telosI2CWriteByte(), telosI2CWriteCmd(), telosSetSCL(), telosSetSDA(), test(), testleds(), timeout(), writepicROM()

5.2.2 Class Variables

Name	Description
ATMELRADIOAPP	**Value:** `83`
autocrc	**Value:** `0`
verbose	**Value:** `False`
connected	**Value:** `0`
enable_AACK	**Value:** `False`
Inherited from killerbee.GoodFETAVR.GoodFETAVR (Section 3.2) APP, AVRAPP, AVRDevices, AVRVendors	
Inherited from killerbee.GoodFET.GoodFET (Section 2.5) GLITCHAPP, MONITORAPP, app, baudrates, besilent, count, data, symbols, verb	

6 Module killerbee.config

6.1 Variables

Name	Description
DB_HOST	**Value:** `''`
DB_PORT	**Value:** `3306`
DB_NAME	**Value:** `''`
DB_USER	**Value:** `''`
DB_PASS	**Value:** `''`
DEV_ENABLE_FREAK-DUINO	**Value:** `False`
DEV_ENABLE_ZIGDUI-NO	**Value:** `False`
__package__	**Value:** `None`

7 Module killerbee.daintree

7.1 Variables

Name	Description
___package___	**Value:** `'killerbee'`

7.2 Class DainTreeDumper

7.2.1 Methods

___init___(*self*, *savefile*)

Writes to the specified file in Daintree SNA packet capture file format.

Parameters

`savefile`: Output Daintree SNA packet capture file.
(type=String)

Return Value

None

pcap_dump(*self*, *packet*, *ts_sec*=`None`, *ts_usec*=`None`, *orig_len*=`None`)

This method is a wrapper around the pwrite() method for compatibility with the PcapDumper.pcap_dump method.

pwrite(*self*, *packet*, *channel*=`26`, *rssi*=`0`)

Appends a new packet to the daintree capture file.

Parameters

`packet`: Packet contents
(type=String)

`channel`: Capture file reported channel number (optional, def=26)
(type=Int)

`rssi`: Capture file repored RSSI (optional, def=0)
(type=Int)

Return Value

None

close(*self*)

Close the input packet capture file.

Return Value

None

7.3 Class DainTreeReader

7.3.1 Methods

__init__(*self*, *savefile*)

Reads from a specified Daintree SNA packet capture file.

Parameters

`savefile`: Daintree SNA packet capture filename to read from.

(type=String)

Return Value

None. An exception is raised if the capture file is not in Daintree SNA format.

close(*self*)

Close the output packet capture.

Return Value

None

pnext(*self*)

Retrieves the next packet from the capture file. Returns a list of [Hdr, packet] where Hdr is a list of [timestamp, snaplen, plen] and packet is a string of the payload content. Returns None at the end of the packet capture.

Return Value

List

8 Module killerbee.dblog

8.1 Variables

Name	Description
__package__	**Value:** `'killerbee'`

8.2 Class DBReader

8.2.1 Methods

__init__(*self*)

close(*self*)

query_one(*self*, *table*, *columns*, *where*)

query(*self*, *sql*)

8.3 Class DBLogger

8.3.1 Methods

__init__(*self*, *datasource*=`None`, *channel*=`None`)

close(*self*)

set_channel(*self*, *chan*)

add_packet(*self*, *full*=`None`, *scapy*=`None`, *bytes*=`None`, *rssi*=`None`, *location*=`None`, *datetime*=`None`, *channel*=`None`)

add_location(*self*, *location*)

add_device(*self*, *shortaddr*, *panid*)

insert(*self*, *sql*, *packetbytes*=`None`)

9 Module killerbee.dev_apimote

GoodFET Chipcon RF Radio Client for ApiMote Hardware

(C) 2013 Ryan Speers <ryan at riverloopsecurity.com>

The ApiMote product is a work in progress. This code is being rewritten and refactored.

TODO list (help is welcomed):

- RF testing and calibration for RSSI/dBm
- Testing carrier jamming and implementing jammer_off()
- Platform recognition (ApiMote versons)

9.1 Variables

Name	Description
DEFAULT_REVISION	**Value:** `2`
CC2420_REG_SYNC	**Value:** `20`
__package__	**Value:** `'killerbee'`

9.2 Class APIMOTE

9.2.1 Methods

__init__(*self*, *dev*, *revision*=2)

Instantiates the KillerBee class for the ApiMote platform running GoodFET firmware.

Parameters

`dev`: Serial device identifier (ex /dev/ttyUSB0)

(type=String)

`revision`: The revision number for the ApiMote, which is used by the called GoodFET libraries to properly communicate with and configure the hardware.

(type=Integer)

Return Value

None

(type=None)

close(*self*)

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device.

Return Value

List of 3 strings identifying device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel (limited to 2.4 GHz channels 11-26)

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet:`	Packet contents to transmit, without FCS. *(type=String)*
`channel:`	Sets the channel, optional *(type=Integer)*
`count:`	Transmits a specified number of frames, def=1 *(type=Integer)*
`delay:`	Delay between each frame, def=1 *(type=Float)*

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns a dictionary containing packet data, else None.

Parameters

`timeout:`	Timeout to wait for packet reception in usec *(type=Integer)*

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a dictionary is returned with the keys bytes (string of packet bytes), validcrc (boolean if a vaid CRC), rssi (unscaled RSSI), and location (may be set to None). For backwards compatibility, keys for 0,1,2 are provided such that it can be treated as if a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

ping(*self*, *da*, *panid*, *sa*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

jammer_on(*self*, *channel*=`None`)

Not yet implemented.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

set_sync(*self*, *sync*=`42767`)

Set the register controlling the 802.15.4 PHY sync byte.

jammer_off(*self*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

10 Module killerbee.dev_freakduino

Support from the Freakduino platform from Abika/Freaklabs.

This is not a maintained platfrom and functionality may be broken or lacking.

10.1 Variables

Name	Description
MODE_NONE	**Value:** `1`
MODE_SNIFF	**Value:** `2`
__package__	**Value:** `'killerbee'`

10.2 Class FREAKDUINO

10.2.1 Methods

__init__(*self*, *serialpath*)

Instantiates the KillerBee class for our sketch running on ChibiArduino on Freakduino hardware.

Parameters

`serialpath`: /dev/ttyUSB* type serial port identifier

Return Value

None

(type=None)

close(*self*)

Closes the serial port. After closing, must reinitialize class again before use.

Return Value

None

(type=None)

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device.

Return Value

List of 3 strings identifying device.

(type=List)

eeprom_dump(*self*)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel (limited to 2.4 GHz channels 11-26)

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet:` Packet contents to transmit, without FCS.

(type=String)

`channel:` Sets the channel, optional

(type=Integer)

`count:` Transmits a specified number of frames, def=1

(type=Integer)

`delay:` Delay between each frame, def=1

(type=Float)

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns packet data as a string, else None.

Parameters

`timeout:` Timeout to wait for packet reception in usec

(type=Integer)

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

pnext_rec(*self*, *timeout*=`100`)

getCaptureDateTime(*self*, *data*)

processLocationUpdate(*self*, *ldata*)

Take a location string passed from the device and update the driver's internal state of last received location. Format of ldata: longlatialtidate

ping(*self*, *da*, *panid*, *sa*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

jammer_on(*self*, *channel*=`None`)

Not yet implemented.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

jammer_off(*self*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

11 Module killerbee.dev_rzusbstick

11.1 Variables

Name	Description
USBVER	**Value:** `1`
RZ_CMD_SET_MODE	RZUSB opcode to specify operating mode **Value:** `7`
RZ_CMD_SET_CHAN-NEL	RZUSB opcode to specify the channel **Value:** `8`
RZ_CMD_OPEN_STRE-AM	RZUSB opcode to open a stream for packet injection **Value:** `9`
RZ_CMD_CLOSE_STR-EAM	RZUSB opcode to close a stream for packet injection **Value:** `10`
RZ_CMD_INJECT_FR-AME	RZUSB opcode to specify a frame to inject **Value:** `13`
RZ_CMD_JAMMER_O-N	RZUSB opcode to turn the jammer function on **Value:** `14`
RZ_CMD_JAMMER_O-FF	RZUSB opcode to turn the jammer function off **Value:** `15`
RZ_CMD_MODE_AC	RZUSB mode for aircapture (inject + sniff) **Value:** `0`
RZ_CMD_MODE_NON-E	RZUSB no mode specified **Value:** `4`
RZ_RESP_LOCAL_TI-MEOUT	RZUSB Response: Local Timeout Error **Value:** `0`
RZ_RESP_SUCCESS	RZUSB Response: Success **Value:** `128`
RZ_RESP_SYNTACTIC-AL_ERROR	RZUSB Response: Syntactical Error **Value:** `129`
RZ_RESP_SEMANTIC-AL_ERROR	RZUSB Response: Semantical Error **Value:** `130`
RZ_RESP_HW_TIMEO-UT	RZUSB Response: Hardware Timeout **Value:** `131`
RZ_RESP_SIGN_ON	RZUSB Response: Sign On **Value:** `132`
RZ_RESP_GET_PARA-METER	RZUSB Response: Get Parameter **Value:** `133`
RZ_RESP_TRX_READ-_REGISTER	RZUSB Response: Transceiver Read Register Error **Value:** `134`

continued on next page

Name	Description
RZ_RESP_TRX_READ-_FRAME	RZUSB Response: Transceiver Read Frame Error **Value:** `135`
RZ_RESP_TRX_READ-_SRAM	RZUSB Response: Transceiver Read SRAM Error **Value:** `136`
RZ_RESP_TRX_GET_-PIN	RZUSB Response: Transceiver Get PIN Error **Value:** `137`
RZ_RESP_TRX_BUSY	RZUSB Response: Transceiver Busy Error **Value:** `138`
RZ_RESP_PRITMITIV-E_FAILED	RZUSB Response: Primitive Failed Error **Value:** `139`
RZ_RESP_PRITMITIV-E_UNKNOWN	RZUSB Response: Primitive Unknown Error **Value:** `140`
RZ_RESP_COMMAND-_UNKNOWN	RZUSB Response: Command Unknown Error **Value:** `141`
RZ_RESP_BUSY_SCA-NING	RZUSB Response: Busy Scanning Error **Value:** `142`
RZ_RESP_BUSY_CAP-TURING	RZUSB Response: Busy Capturing Error **Value:** `143`
RZ_RESP_OUT_OF_M-EMORY	RZUSB Response: Out of Memory Error **Value:** `144`
RZ_RESP_BUSY_JAM-MING	RZUSB Response: Busy Jamming Error **Value:** `145`
RZ_RESP_NOT_INITI-ALIZED	RZUSB Response: Not Initialized Error **Value:** `146`
RZ_RESP_NOT_IMPL-EMENTED	RZUSB Response: Opcode Not Implemented Error **Value:** `147`
RZ_RESP_PRIMITIVE-_FAILED	RZUSB Response: Primitive Failed Error **Value:** `148`
RZ_RESP_VRT_KERN-EL_ERROR	RZUSB Response: Could not execute due to vrt_kernel_error **Value:** `149`
RZ_RESP_BOOT_PAR-AM	RZUSB Response: Boot Param Error **Value:** `150`
RZ_EVENT_STREAM_-AC_DATA	RZUSB Event Opcode: AirCapture Data **Value:** `80`
RESPONSE_MAP	Dictionary of RZUSB error to strings **Value:** `{0: 'Local Timeout Error', 128: 'Success', 129: 'Syntacti...`
RZ_USB_VEND_ID	RZUSB USB VID **Value:** `1003`

continued on next page

Name	Description
RZ_USB_PROD_ID	RZUSB USB PID **Value:** `8458`
RZ_USB_COMMAND_-EP	RZUSB USB Command Endpoint Identifier **Value:** `2`
RZ_USB_RESPONSE_-EP	RZUSB USB Response Endpoint Identifier **Value:** `132`
RZ_USB_PACKET_EP	RZUSB USB Packet Endpoint Identifier **Value:** `129`
___package___	**Value:** `'killerbee'`

11.2 Class RZUSBSTICK

11.2.1 Methods

___init___(*self*, *dev*, *bus*)

Instantiates the KillerBee class for the RZUSBSTICK hardware.

Parameters

`dev`: USB device identifier

(type=TODO)

`bus`: Identifies the USB bus the device is on

(type=TODO)

Return Value

None

(type=None)

close(*self*)

Closes the device handle. To be re-used, class should be re-instantiated.

Return Value

None

(type=None)

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device identifier, product string and serial number in a list of strings.

Return Value

List of 3 strings identifying device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

jammer_on(*self*, *channel*=`None`)

Not yet implemented. Stay tuned.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

jammer_off(*self*, *channel*=`None`)

Not yet implemented. Stay tuned.

Return Value

None

(type=None)

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel. Currently, support is limited to 2.4 GHz channels 11 - 26.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet`: Packet contents to transmit, without FCS.

(type=String)

`channel`: Sets the channel, optional

(type=Integer)

`count`: Transmits a specified number of frames, def=1

(type=Integer)

`delay`: Delay between each frame, def=1

(type=Float)

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns packet data as a string, else None.

Parameters

`timeout`: Timeout to wait for packet reception in usec

(type=Integer)

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

ping(*self*, *da*, *panid*, *sa*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

12 Module killerbee.dev_telosb

Support for the TelosB / Tmote Sky platforms, and close clones.

Utilizes the GoodFET firmware with CCSPI application, and the GoodFET client code.

12.1 Variables

Name	Description
CC2420_REG_SYNC	**Value:** `20`
__package__	**Value:** `'killerbee'`

12.2 Class TELOSB

12.2.1 Methods

__init__(*self*, *dev*)

Instantiates the KillerBee class for our TelosB/TmoteSky running GoodFET firmware.

Parameters

`dev`: Serial device identifier (ex /dev/ttyUSB0)

(type=String)

Return Value

None

(type=None)

close(*self*)

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device.

Return Value

List of 3 strings identifying device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel (limited to 2.4 GHz channels 11-26)

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet`: Packet contents to transcript, without FCS.

(type=String)

`channel`: Sets the channel, optional

(type=Integer)

`count`: Transmits a specified number of frames, def=1

(type=Integer)

`delay`: Delay between each frame, def=1

(type=Float)

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns a dictionary containing packet data, else None.

Parameters

`timeout`: Timeout to wait for packet reception in usec

(type=Integer)

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a dictionary is returned with the keys bytes (string of packet bytes), validcrc (boolean if a vaid CRC), rssi (unscaled RSSI), and location (may be set to None). For backwards compatibility, keys for 0,1,2 are provided such that it can be treated as if a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

ping(*self*, *da*, *panid*, *sa*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

jammer_on(*self*, *channel*=`None`)

Not yet implemented.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

set_sync(*self*, *sync*=`42767`)

Set the register controlling the 802.15.4 PHY sync byte.

jammer_off(*self*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

13 Module killerbee.dev_wislab

13.1 Functions

ntp_to_system_time(*secs*, *msecs*)

convert a NTP time to system time

getFirmwareVersion(*ip*)

getMacAddr(*ip*)

Returns a string for the MAC address of the sniffer.

isWislab(*dev*)

13.2 Variables

Name	Description
DEFAULT_IP	**Value:** `'10.10.10.2'`
DEFAULT_GW	**Value:** `'10.10.10.1'`
DEFAULT_UDP	**Value:** `17754`
TESTED_FW_VERS	**Value:** `['0.5']`
NTP_DELTA	Convert the two parts of an NTP timestamp to a datetime object. Similar code from Wireshark source: 575 /* NTP_BASETIME is in fact epoch - ntp_start_time */ 576 #define NTP_BASETIME 2208988800ul 619 void 620 ntp_to_nstime(tvbuff_t *tvb, gint offset, nstime_t *nstime) 621 { 622 nstime->secs = tvb_get_ntohl(tvb, offset); 623 if (nstime->secs) 624 nstime->secs -= NTP_BASETIME; 625 nstime->nsecs = (int)(tvb_get_ntohl(tvb, offset+4)/(NTP_FLOAT_DENOM/1000000000.0)); 626 } **Value:** `2207520000`
__package__	**Value:** `'killerbee'`

13.3 Class WISLAB

13.3.1 Methods

__init__(*self*, *dev*=`'10.10.10.2'`, *recvport*=`17754`, *recvip*=`'10.10.10.1'`)

Instantiates the KillerBee class for the Wislab Sniffer.

Parameters

`dev`: IP address (ex 10.10.10.2)

(type=String)

`recvport`: UDP port to listen for sniffed packets on.

(type=Integer)

`recvip`: IP address of the host, where the sniffer will send sniffed packets to.

(type=String)

Return Value

None

(type=None)

close(*self*)

Actually close the receiving UDP socket.

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device.

Return Value

List of 3 strings identifying device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data.

Parameters

`channel:` Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off.

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel (limited to 2.4 GHz channels 11-26)

Parameters

`channel:` Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Not implemented.

pnext(*self*, *timeout*=`100`)

Returns a dictionary containing packet data, else None.

Parameters

`timeout`: Timeout to wait for packet reception in usec

(type=Integer)

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a dictionary is returned with the keys bytes (string of packet bytes), validcrc (boolean if a vaid CRC), rssi (unscaled RSSI), and location (may be set to None). For backwards compatibility, keys for 0,1,2 are provided such that it can be treated as if a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

jammer_on(*self*, *channel*=`None`)

Not yet implemented.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

jammer_off(*self*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

14 Module killerbee.dev_zigduino

Support is currently only tested with Zigduino r1. Zigduino support is contributed by neighbor bx. If you can test with or can provide us a Zigduino r2 for testing, that would be great.

Items still TODO:

- sniffer_off() needs to instruct the firmware to stop sending packets
- calibrate the RSSI reading on the r2 hardware and adjust for it
- add jamming support

14.1 Variables

Name	Description
ATMEL_REG_SYNC	**Value:** `11`
__package__	**Value:** `'killerbee'`

14.2 Class ZIGDUINO

14.2.1 Methods

__init__(*self*, *dev*)

Instantiates the KillerBee class for Zigduino running GoodFET firmware.

Parameters

`dev`: Serial device identifier (ex /dev/ttyUSB0)

(type=String)

Return Value

None

(type=None)

close(*self*)

check_capability(*self*, *capab*)

get_capabilities(*self*)

get_dev_info(*self*)

Returns device information in a list identifying the device.

Return Value

List of 3 strings identifying device.

(type=List)

sniffer_on(*self*, *channel*=`None`)

Turns the sniffer on such that pnext() will start returning observed data. Will set the command mode to Air Capture if it is not already set.

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

sniffer_off(*self*)

Turns the sniffer off, freeing the hardware for other functions. It is not necessary to call this function before closing the interface with close().

Return Value

None

set_channel(*self*, *channel*)

Sets the radio interface to the specifid channel (limited to 2.4 GHz channels 11-26)

Parameters

`channel`: Sets the channel, optional

(type=Integer)

Return Value

None

inject(*self*, *packet*, *channel*=`None`, *count*=`1`, *delay*=`0`)

Injects the specified packet contents.

Parameters

`packet:`	Packet contents to transmit, without FCS. *(type=String)*
`channel:`	Sets the channel, optional *(type=Integer)*
`count:`	Transmits a specified number of frames, def=1 *(type=Integer)*
`delay:`	Delay between each frame, def=1 *(type=Float)*

Return Value

None

pnext(*self*, *timeout*=`100`)

Returns a dictionary containing packet data, else None.

Parameters

`timeout:`	Timeout to wait for packet reception in usec *(type=Integer)*

Return Value

Returns None is timeout expires and no packet received. When a packet is received, a dictionary is returned with the keys bytes (string of packet bytes), validcrc (boolean if a vaid CRC), rssi (unscaled RSSI), and location (may be set to None). For backwards compatibility, keys for 0,1,2 are provided such that it can be treated as if a list is returned, in the form [String: packet contents | Bool: Valid CRC | Int: Unscaled RSSI]

(type=List)

jammer_on(*self*, *channel*=`None`)

Not yet implemented.

Parameters

`channel:`	Sets the channel, optional *(type=Integer)*

Return Value

None

set_sync(*self*, *sync*=`167`)

Set the register controlling the 802.15.4 PHY sync byte.

jammer_off(*self*, *channel*=`None`)

Not yet implemented.

Return Value

None

(type=None)

15 Module killerbee.dot154decode

15.1 Variables

Name	Description
DOT154_FCF_TYPE_-MASK	Frame type mask **Value:** 7
DOT154_FCF_SEC_EN	Set for encrypted payload **Value:** 8
DOT154_FCF_FRAME-_PND	Frame pending **Value:** 16
DOT154_FCF_ACK_R-EQ	ACK request **Value:** 32
DOT154_FCF_INTRA_-PAN	Intra-PAN activity **Value:** 64
DOT154_FCF_DADDR-_MASK	Destination addressing mode mask **Value:** 3072
DOT154_FCF_VERSIO-N_MASK	Frame version **Value:** 12288
DOT154_FCF_SADDR_-MASK	Source addressing mask mode **Value:** 49152
DOT154_FCF_TYPE_-MASK_SHIFT	Frame type mask mode shift **Value:** 0
DOT154_FCF_DADDR-_MASK_SHIFT	Destination addressing mode mask **Value:** 10
DOT154_FCF_VERSIO-N_MASK_SHIFT	Frame versions mask mode shift **Value:** 12
DOT154_FCF_SADDR_-MASK_SHIFT	Source addressing mask mode shift **Value:** 14
DOT154_FCF_ADDR_-NONE	Not sure when this is used **Value:** 0
DOT154_FCF_ADDR_S-HORT	4-byte addressing **Value:** 2
DOT154_FCF_ADDR_-EXT	8-byte addressing **Value:** 3
DOT154_FCF_TYPE_B-EACON	Beacon frame **Value:** 0
DOT154_FCF_TYPE_D-ATA	Data frame **Value:** 1
DOT154_FCF_TYPE_A-CK	Acknowledgement frame **Value:** 2
DOT154_FCF_TYPE_-MACCMD	MAC Command frame **Value:** 3

continued on next page

Name	Description
DOT154_CRYPT_NON-E	No encryption, no MIC **Value:** `0`
DOT154_CRYPT_MIC3-2	No encryption, 32-bit MIC **Value:** `1`
DOT154_CRYPT_MIC6-4	No encryption, 64-bit MIC **Value:** `2`
DOT154_CRYPT_MIC1-28	No encryption, 128-bit MIC **Value:** `3`
DOT154_CRYPT_ENC	Encryption, no MIC **Value:** `4`
DOT154_CRYPT_ENC-_MIC32	Encryption, 32-bit MIC **Value:** `5`
DOT154_CRYPT_ENC-_MIC64	Encryption, 64-bit MIC **Value:** `6`
DOT154_CRYPT_ENC-_MIC128	Encryption, 128-bit MIC **Value:** `7`
__package__	**Value:** `'killerbee'`

15.2 Class Dot154PacketParser

15.2.1 Methods

__init__(*self*)

Instantiates the Dot154PacketParser class.

decrypt(*self*, *packet*, *key*)

Decrypts the specified packet. Returns empty string if the packet is not encrypted, or if decryption MIC validation fails.

Parameters

`packet`: Packet contents.

(type=String)

`key`: Key contents.

(type=String)

Return Value

Decrypted packet contents, empty string if not encrypted or if decryped MIC fails validation.

(type=String)

pktchop(*self*, *packet*)

Chops up the specified packet contents into a list of fields. Does not attempt to re-order the field values for parsing. ".join(X) will reassemble original packet string. Fields which may or may not be present (such as the Source PAN field) are empty if they are not present, keeping the list elements consistent, as follows: FCF | Seq# | DPAN | DA | SPAN | SA | [Beacon Data] | PHY Payload

If the packet is a beacon frame, the Beacon Data field will be populated as a list element in the format:

Superframe Spec | GTS Fields | Pending Addr Counts | Proto ID | Stack Profile/Profile Version | Device Capabilities | Ext PAN ID | TX Offset | Update ID

An exception is raised if the packet contents are too short to decode.

Parameters

`packet`: Packet contents.

(type=String)

Return Value

Chopped contents of the 802.15.4 packet into list elements.

(type=list)

hdrlen(*self*, *packet*)

Returns the length of the 802.15.4 header.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the 802.15.4 header.

(type=Int)

payloadlen(*self*, *packet*)

Returns the length of the 802.15.4 payload.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the 802.15.4 payload.

(type=Int)

nonce(*self*, *packet*)

Returns the nonce of the 802.15.4 packet. Returns empty string for unencrypted frames.

Parameters

`packet`: Packet contents to evaluate for nonce.

(type=String)

Return Value

Nonce, empty when the frame is not encrypted.

(type=String)

16 Module killerbee.kbutils

16.1 Functions

devlist__usb__v1x(*vendor*=`None`, *product*=`None`)

Private function. Do not call from tools/scripts/etc.

devlist__usb__v0x(*vendor*=`None`, *product*=`None`)

Private function. Do not call from tools/scripts/etc.

isIpAddr(*ip*)

Return True if the given string is a valid IPv4 or IPv6 address.

devlist(*vendor*=`None`, *product*=`None`, *gps*=`None`, *include*=`None`)

Return device information for all present devices, filtering if requested by vendor and/or product IDs on USB devices, and running device fingerprint functions on serial devices.

Parameters

`gps`: Optional serial device identifier for an attached GPS unit. If provided, or if global variable has previously been set, KillerBee skips that device in device enumeration process.

(type=String)

`include`: Optional list of device handles to be appended to the normally found devices. This is useful for providing IP addresses for remote scanners.

(type=List of Strings)

Return Value

List of device information present. For USB devices, get [busdir:devfilename, productString, serialNumber] For serial devices, get [serialFileName, deviceDescription, `""`]

(type=List)

get__serial__devs(*seriallist*)

isSerialDeviceString(*s*)

get_serial_ports(*include*=`None`)

Private function. Do not call from tools/scripts/etc. This should return a list of device paths for serial devices that we are interested in, aka USB serial devices using FTDI chips such as the TelosB, ApiMote, etc. This should handle returning a list of devices regardless of the *nix it is running on. Support for more *nix and winnt needed.

Parameters

`include`: A list of device strings, of which any which appear to be serial device handles will be added to the set of serial ports returned by the normal search. This may be useful if we're not including some oddly named serial port which you have a KillerBee device on. Optional.

(type=List of Strings, or None)

isgoodfetccspi(*serialdev*)

Determine if a given serial device is running the GoodFET firmware with the CCSPI application. This should either be a TelosB/Tmote Sky GOODFET or an Api-Mote design.

Parameters

`serialdev`: Path to a serial device, ex /dev/ttyUSB0.

(type=String)

Return Value

Tuple with the fist element==True if it is some goodfetccspi device. The second element is the subtype, and is 0 for telosb devices and 1 for apimote devices.

(type=Tuple)

iszigduino(*serialdev*)

Determine if a given serial device is running the GoodFET firmware with the atmel_radio application. This should be a Zigduino (only tested on hardware r1 currently).

Parameters

`serialdev`: Path to a serial device, ex /dev/ttyUSB0.

(type=String)

Return Value

Boolean with the fist element==True if it is a goodfet atmel128 device.

(type=Boolean)

isfreakduino(*serialdev*)

Determine if a given serial device is a Freakduino attached with the right sketch loaded.

Parameters

`serialdev`: Path to a serial device, ex /dev/ttyUSB0.

(type=String)

Return Value

Boolean

search_usb(*device*)

Takes either None, specifying that any USB device in the global vendor and product lists are acceptable, or takes a string that identifies a device in the format <BusNumber>:<DeviceNumber>, and returns the pyUSB objects for bus and device that correspond to the identifier string.

search_usb_bus_v0x(*bus*, *busNum*, *devNum*)

Helper function for USB enumeration in pyUSB 0.x enviroments.

hexdump(*src*, *length*=`16`)

Creates a tcpdump-style hex dump string output.

Parameters

`src`: Input string to convert to hexdump output.

(type=String)

`length`: Optional length of data for a single row of output, def=16

(type=Int)

Return Value

String

randbytes(*size*)

Returns a random string of size bytes. Not cryptographically safe.

Parameters

`size`: Length of random data to return.

(type=Int)

Return Value

String

randmac(*length*=8)

Returns a random MAC address using a list valid OUI's from ZigBee device manufacturers. Data is returned in air-format byte order (LSB first).

Parameters

`length`: Optional length of MAC address, def=8. Minimum address return length is 3 bytes for the valid OUI.

(type=String)

Return Value

A randomized MAC address in a little-endian byte string.

(type=String)

makeFCS(*data*)

Do a CRC-CCITT Kermit 16bit on the data given Implemented using pseudocode from: June 1986, Kermit Protocol Manual See also: http://regregex.bbcmicro.net/crc-catalogue.htm#crc.cat.kermit

Return Value

a CRC that is the FCS for the frame, as two hex bytes in little-endian order.

16.2 Variables

Name	Description
USBVER	**Value:** `1`
RZ_USB_VEND_ID	**Value:** `1003`
RZ_USB_PROD_ID	**Value:** `8458`
ZN_USB_VEND_ID	**Value:** `1240`
ZN_USB_PROD_ID	**Value:** `14`
FTDI_X_USB_VEND_I-D	**Value:** `1027`
FTDI_X_USB_PROD_I-D	**Value:** `24597`
usbVendorList	**Value:** `[1003, 1240]`
usbProductList	**Value:** `[8458, 14]`
gps_devstring	**Value:** `None`
__package__	**Value:** `'killerbee'`

16.3 Class KBCapabilities

Class to store and report on the capabilities of a specific KillerBee device.

16.3.1 Methods

__init__(*self*)

check(*self*, *capab*)

getlist(*self*)

setcapab(*self*, *capab*, *value*)

require(*self*, *capab*)

is_valid_channel(*self*, *channel*)

Based on sniffer capabilities, return if this is an OK channel number.

Return Value

Boolean

16.3.2 Class Variables

Name	Description
NONE	Capabilities Flag: No Capabilities **Value:** `0`
SNIFF	Capabilities Flag: Can Sniff **Value:** `1`
SETCHAN	Capabilities Flag: Can Set the Channel **Value:** `2`
INJECT	Capabilities Flag: Can Inject Frames **Value:** `3`
PHYJAM	Capabilities Flag: Can Jam PHY Layer **Value:** `4`
SELFACK	Capabilities Flag: Can ACK Frames Automatically **Value:** `5`
PHYJAM_REFLEX	Capabilities Flag: Can Jam PHY Layer Reflexively **Value:** `6`
SET_SYNC	Capabilities Flag: Can set the register controlling 802.15.4 sync byte **Value:** `7`

continued on next page

Name	Description
FREQ_2400	Capabilities Flag: Can preform 2.4 GHz sniffing (ch 11-26) **Value:** `8`
FREQ_900	Capabilities Flag: Can preform 900 MHz sniffing (ch 1-10) **Value:** `9`

16.4 Class findFromList

object ─┐
killerbee.kbutils.findFromList

Known Subclasses: killerbee.kbutils.findFromListAndBusDevId

Custom matching function for pyUSB 1.x. Used by usb.core.find's custom_match parameter.

16.4.1 Methods

__init__(*self*, *vendors_*, *products_*)

Takes a list of vendor IDs and product IDs.

Overrides: object.__init__

__call__(*self*, *device*)

Returns True if the device being searched is in these lists.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.4.2 Properties

Name	Description
Inherited from object __class__	

16.5 Class findFromListAndBusDevId

object
killerbee.kbutils.findFromList
killerbee.kbutils.findFromListAndBusDevId

Custom matching function for pyUSB 1.x. Used by usb.core.find's custom_match parameter.

16.5.1 Methods

__init__(*self*, *busNum_*, *devNum_*, *vendors_*, *products_*)

Takes a list of vendor IDs and product IDs.

Overrides: object.__init__

__call__(*self*, *device*)

Returns True if the device being searched is in these lists.

Overrides: killerbee.kbutils.findFromList.__call__

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.5.2 Properties

Name	Description
Inherited from object	
__class__	

16.6 Class KBException

```
object ─┐
        │
exceptions.BaseException ─┐
                          │
        exceptions.Exception ─┐
                              │
                    killerbee.kbutils.KBException
```

Known Subclasses: killerbee.kbutils.KBInterfaceError

Base class for all KillerBee specific exceptions.

16.6.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

16.6.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

16.7 Class KBInterfaceError

```
object ─┐
exceptions.BaseException ─┐
        exceptions.Exception ─┐
    killerbee.kbutils.KBException ─┐
                                   killerbee.kbutils.KBInterfaceError
```

Custom exception for KillerBee having issues communicating with an interface, such as opening a port, syncing with the firmware, etc.

16.7.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

16.7.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

17 Package killerbee.openear

17.1 Modules

- **capture** *(Section 18, p. 69)*
- **gps** *(Section 19, p. 71)*
  - **client** *(Section 20, p. 73)*
  - **gps** *(Section **??**, p. **??**)*
  - **gps'** *(Section 21, p. 76)*
  - **misc** *(Section 22, p. 79)*
- **scanner** *(Section 23, p. 80)*

17.2 Variables

Name	Description
__package__	**Value:** `'killerbee.openear'`

18 Module killerbee.openear.capture

18.1 Functions

startCapture(*dev*, *capChan*)

interrupt(*signum*, *frame*)

18.2 Variables

Name	Description
triggers	**Value:** `[]`
__package__	**Value:** `'killerbee.openear'`

18.3 Class CaptureThread

object

threading._Verbose

threading.Thread

killerbee.openear.capture.CaptureThread

18.3.1 Methods

__init__(*self*, *channel*, *devstring*, *fname*, *trigger*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

run(*self*)

Overrides: threading.Thread.run

Inherited from threading.Thread

__repr__(), getName(), isAlive(), isDaemon(), is_alive(), join(), setDaemon(), setName(), start()

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

18.3.2 Properties

Name	Description
Inherited from threading.Thread daemon, ident, name	
Inherited from object ___class___	

19 Package killerbee.openear.gps

19.1 Modules

- **client** *(Section 20, p. 73)*
- **gps** *(Section ??, p. ??)*
- **gps'** *(Section 21, p. 76)*
- **misc** *(Section 22, p. 79)*

19.2 Variables

Name	Description
api_major_version	**Value:** `4`
api_minor_version	**Value:** `1`
AIS_SET	**Value:** `268435456`
ALTITUDE_SET	**Value:** `16`
ATTITUDE_SET	**Value:** `16384`
AUXDATA_SET	**Value:** `2147483648`
CLIMBERR_SET	**Value:** `2097152`
CLIMB_SET	**Value:** `128`
DEVICEID_SET	**Value:** `16777216`
DEVICELIST_SET	**Value:** `8388608`
DEVICE_SET	**Value:** `4194304`
DOP_SET	**Value:** `1024`
ERROR_SET	**Value:** `33554432`
GPSD_PORT	**Value:** `'2947'`
HERR_SET	**Value:** `4096`
KNOTS_TO_KPH	**Value:** `1.852`
KNOTS_TO_MPH	**Value:** `1.1507794`
KNOTS_TO_MPS	**Value:** `0.51444444`
LATLON_SET	**Value:** `8`
MAXCHANNELS	**Value:** `20`
METERS_TO_FEET	**Value:** `3.2808399`
METERS_TO_MILES	**Value:** `0.00062137119`
MODE_2D	**Value:** `2`
MODE_3D	**Value:** `3`
MODE_NO_FIX	**Value:** `1`
MODE_SET	**Value:** `512`
MPS_TO_KNOTS	**Value:** `1.9438445`
MPS_TO_KPH	**Value:** `3.6`
MPS_TO_MPH	**Value:** `2.2369363`
NaN	**Value:** `nan`
ONLINE_SET	**Value:** `1`

continued on next page

Name	Description
PACKET_SET	**Value:** `536870912`
POLICY_SET	**Value:** `32768`
RAW_SET	**Value:** `131072`
RTCM2_SET	**Value:** `67108864`
RTCM3_SET	**Value:** `134217728`
SATELLITE_SET	**Value:** `65536`
SIGNAL_STRENGTH_-UNKNOWN	**Value:** `nan`
SPEEDERR_SET	**Value:** `524288`
SPEED_SET	**Value:** `32`
STATUS_DGPS_FIX	**Value:** `2`
STATUS_FIX	**Value:** `1`
STATUS_NO_FIX	**Value:** `0`
STATUS_SET	**Value:** `256`
TIMERR_SET	**Value:** `4`
TIME_SET	**Value:** `2`
TRACKERR_SET	**Value:** `1048576`
TRACK_SET	**Value:** `64`
UNION_SET	**Value:** `511707136`
USED_SET	**Value:** `262144`
VERR_SET	**Value:** `8192`
VERSION_SET	**Value:** `2048`
WATCH_DEVICE	**Value:** `64`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NEWSTYLE	**Value:** `128`
WATCH_NMEA	**Value:** `4`
WATCH_OLDSTYLE	**Value:** `65536`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
___package___	**Value:** `'killerbee.openear.gps'`
session	**Value:** `gps.gps()`

20 Module killerbee.openear.gps.client

20.1 Variables

Name	Description
GPSD_PORT	**Value:** `'2947'`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NMEA	**Value:** `4`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
WATCH_DEVICE	**Value:** `64`
__package__	**Value:** `'killerbee.openear.gps'`

20.2 Class gpscommon

Known Subclasses: killerbee.openear.gps.client.gpsjson

Isolate socket handling and buffering from the protcol interpretation.

20.2.1 Methods

__init__(*self*, *host*=`'127.0.0.1'`, *port*=`'2947'`, *verbose*=`0`)

connect(*self*, *host*, *port*)

Connect to a host on a given port.

If the hostname ends with a colon (':') followed by a number, and there is no port specified, that suffix will be stripped off and the number interpreted as the port number to use.

close(*self*)

__del__(*self*)

waiting(*self*)

Return True if data is ready for the client.

read(*self*)

Wait for and read data being streamed from the daemon.

send(*self*, *commands*)

Ship commands to the daemon.

20.3 Class gpsjson

killerbee.openear.gps.client.gpscommon
↳ **killerbee.openear.gps.client.gpsjson**

Known Subclasses: killerbee.openear.gps.gps'.gps

Basic JSON decoding.

20.3.1 Methods

___iter___(*self*)

json_unpack(*self*, *buf*)

stream(*self*, *flags*=`0`, *outfile*=`None`)

Control streaming reports from the daemon,

Inherited from killerbee.openear.gps.client.gpscommon(Section 20.2)

___del___(), ___init___(), close(), connect(), read(), send(), waiting()

20.4 Class dictwrapper

Wrapper that yields both class and dictionary behavior,

20.4.1 Methods

___init___(*self*, ***ddict*)

get(*self*, *k*, *d*=`None`)

keys(*self*)

__getitem__(*self*, *key*)

Emulate dictionary, for new-style interface.

__setitem__(*self*, *key*, *val*)

Emulate dictionary, for new-style interface.

__contains__(*self*, *key*)

__str__(*self*)

__repr__(*self*)

21 Module killerbee.openear.gps.gps'

21.1 Functions

isnan(x)

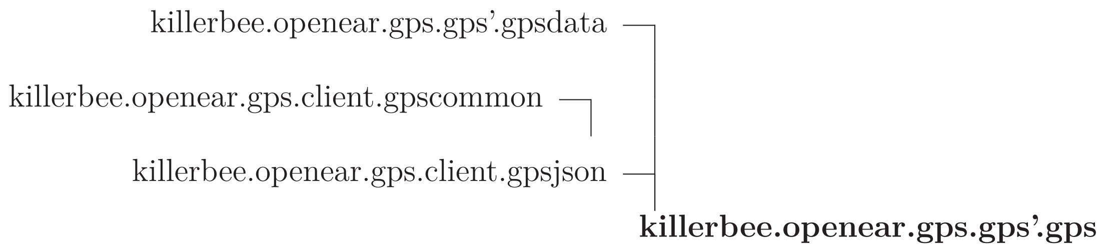
21.2 Variables

Name	Description
AIS_SET	**Value:** `268435456`
ALTITUDE_SET	**Value:** `16`
ATTITUDE_SET	**Value:** `16384`
AUXDATA_SET	**Value:** `2147483648`
CLIMBERR_SET	**Value:** `2097152`
CLIMB_SET	**Value:** `128`
DEVICEID_SET	**Value:** `16777216`
DEVICELIST_SET	**Value:** `8388608`
DEVICE_SET	**Value:** `4194304`
DOP_SET	**Value:** `1024`
ERROR_SET	**Value:** `33554432`
GPSD_PORT	**Value:** `'2947'`
HERR_SET	**Value:** `4096`
LATLON_SET	**Value:** `8`
MAXCHANNELS	**Value:** `20`
MODE_2D	**Value:** `2`
MODE_3D	**Value:** `3`
MODE_NO_FIX	**Value:** `1`
MODE_SET	**Value:** `512`
NaN	**Value:** `nan`
ONLINE_SET	**Value:** `1`
PACKET_SET	**Value:** `536870912`
POLICY_SET	**Value:** `32768`
RAW_SET	**Value:** `131072`
RTCM2_SET	**Value:** `67108864`
RTCM3_SET	**Value:** `134217728`
SATELLITE_SET	**Value:** `65536`
SIGNAL_STRENGTH_-UNKNOWN	**Value:** `nan`
SPEEDERR_SET	**Value:** `524288`
SPEED_SET	**Value:** `32`
STATUS_DGPS_FIX	**Value:** `2`
STATUS_FIX	**Value:** `1`

continued on next page

Name	Description
STATUS_NO_FIX	**Value:** `0`
STATUS_SET	**Value:** `256`
TIMERR_SET	**Value:** `4`
TIME_SET	**Value:** `2`
TRACKERR_SET	**Value:** `1048576`
TRACK_SET	**Value:** `64`
UNION_SET	**Value:** `511707136`
USED_SET	**Value:** `262144`
VERR_SET	**Value:** `8192`
VERSION_SET	**Value:** `2048`
WATCH_DEVICE	**Value:** `64`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NEWSTYLE	**Value:** `128`
WATCH_NMEA	**Value:** `4`
WATCH_OLDSTYLE	**Value:** `65536`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
__package__	**Value:** `'killerbee.openear.gps'`

21.3 Class gps


killerbee.openear.gps.gps'.gpsdata
killerbee.openear.gps.client.gpscommon
killerbee.openear.gps.client.gpsjson
killerbee.openear.gps.gps'.gps

Client interface to a running gpsd instance.

21.3.1 Methods

__init__(*self*, *host*=`'127.0.0.1'`, *port*=`'2947'`, *verbose*=`0`, *mode*=`0`)

Overrides: killerbee.openear.gps.client.gpscommon.__init__

next(*self*)

poll(*self*)

Read and interpret data from the daemon.

set_raw_hook(*self*, *hook*)

stream(*self*, *flags*=`0`, *outfile*=`None`)

Ask gpsd to stream reports at your client.

Overrides: killerbee.openear.gps.client.gpsjson.stream

Inherited from killerbee.openear.gps.gps'.gpsdata(Section 21.4)

___repr___()

Inherited from killerbee.openear.gps.client.gpsjson(Section 20.3)

___iter___(), json_unpack()

Inherited from killerbee.openear.gps.client.gpscommon(Section 20.2)

___del___(), close(), connect(), read(), send(), waiting()

21.4 Class gpsdata

Known Subclasses: killerbee.openear.gps.gps'.gps

Position, track, velocity and status information returned by a GPS.

21.4.1 Methods

___init___(*self*)

___repr___(*self*)

21.5 Class gpsfix

21.5.1 Methods

___init___(*self*)

22 Module killerbee.openear.gps.misc

22.1 Functions

Deg2Rad(*x*)

Degrees to radians.

Rad2Deg(*x*)

Radians to degress.

CalcRad(*lat*)

Radius of curvature in meters at specified latitude.

EarthDistance(*(lat1, lon1)*, *(lat2, lon2)*)

Distance in meters between two points specified in degrees.

MeterOffset(*(lat1, lon1)*, *(lat2, lon2)*)

Return offset in meters of second arg from first.

isotime(*s*)

Convert timestamps in ISO8661 format to and from Unix time.

22.2 Variables

Name	Description
METERS_TO_FEET	**Value:** `3.2808399`
METERS_TO_MILES	**Value:** `0.00062137119`
KNOTS_TO_MPH	**Value:** `1.1507794`
KNOTS_TO_KPH	**Value:** `1.852`
KNOTS_TO_MPS	**Value:** `0.51444444`
MPS_TO_KPH	**Value:** `3.6`
MPS_TO_MPH	**Value:** `2.2369363`
MPS_TO_KNOTS	**Value:** `1.9438445`
___package___	**Value:** `'killerbee.openear.gps'`

23 Module killerbee.openear.scanner

23.1 Functions

broadcast_event(*data*)

Send broadcast data to all active threads

signal_handler(*signal*, *frame*)

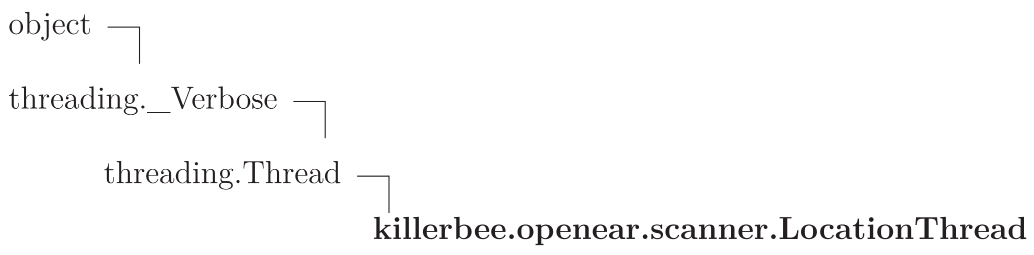
Signal handler called on keyboard interrupt to exit threads and exit scanner script

main(*args*)

23.2 Variables

Name	Description
session	**Value:** `''`
active_queues	**Value:** `[]`
arg_verbose	**Value:** `False`
arg_ppi	**Value:** `False`
arg_db	**Value:** `False`
arg_gps	**Value:** `False`
arg_gps_devstring	**Value:** `''`
latitude	**Value:** `''`
longitude	**Value:** `''`
altitude	**Value:** `''`
last_seen	**Value:** `''`
__package__	**Value:** `'killerbee.openear'`

23.3 Class LocationThread



Thread to update gps location from gpsd

23.3.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

run(*self*)

Overrides: threading.Thread.run

Inherited from threading.Thread

__repr__(), getName(), isAlive(), isDaemon(), is_alive(), join(), setDaemon(), setName(), start()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

23.3.2 Properties

Name	Description
Inherited from threading.Thread	
daemon, ident, name	
Inherited from object	
__class__	

23.4 Class CaptureThread

```
object ─┐
threading._Verbose ─┐
        threading.Thread ─┐
                          killerbee.openear.scanner.CaptureThread
```

Thread to capture on a given channel, using a given device, to a given pcap file, exits when it receives a broadcast shutdown message via Queue.Queue

23.4.1 Methods

__init__(*self*, *dev*, *channel*, *pd*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

run(*self*)

Overrides: threading.Thread.run

Inherited from threading.Thread

__repr__(), getName(), isAlive(), isDaemon(), is_alive(), join(), setDaemon(), setName(), start()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

23.4.2 Properties

Name	Description
Inherited from threading.Thread daemon, ident, name	
Inherited from object __class__	

24 Module killerbee.pcapdlt

24.1 Variables

Name	Description
DLT_NULL	**Value:** 0
DLT_EN10MB	**Value:** 1
DLT_EN3MB	**Value:** 2
DLT_AX25	**Value:** 3
DLT_PRONET	**Value:** 4
DLT_CHAOS	**Value:** 5
DLT_IEEE802	**Value:** 6
DLT_ARCNET	**Value:** 7
DLT_SLIP	**Value:** 8
DLT_PPP	**Value:** 9
DLT_FDDI	**Value:** 10
DLT_ATM_RFC1483	**Value:** 11
DLT_RAW	**Value:** 12
DLT_SLIP_BSDOS	**Value:** 15
DLT_PPP_BSDOS	**Value:** 16
DLT_ATM_CLIP	**Value:** 19
DLT_REDBACK_SMA-RTEDGE	**Value:** 32
DLT_PPP_SERIAL	**Value:** 50
DLT_PPP_ETHER	**Value:** 51
DLT_SYMANTEC_FIR-EWALL	**Value:** 99
DLT_C_HDLC	**Value:** 104
DLT_CHDLC	**Value:** 104
DLT_IEEE802_11	**Value:** 105
DLT_FRELAY	**Value:** 107
DLT_LOOP	**Value:** 108
DLT_ENC	**Value:** 109
DLT_LINUX_SLL	**Value:** 113
DLT_LTALK	**Value:** 114
DLT_ECONET	**Value:** 115
DLT_IPFILTER	**Value:** 116
DLT_OLD_PFLOG	**Value:** 17
DLT_PFSYNC	**Value:** 18
DLT_PFLOG	**Value:** 117
DLT_CISCO_IOS	**Value:** 118
DLT_PRISM_HEADER	**Value:** 119
DLT_AIRONET_HEAD-ER	**Value:** 120

continued on next page

Name	Description
DLT_HHDLC	**Value:** 121
DLT_IP_OVER_FC	**Value:** 122
DLT_SUNATM	**Value:** 123
DLT_RIO	**Value:** 124
DLT_PCI_EXP	**Value:** 125
DLT_AURORA	**Value:** 126
DLT_IEEE802_11_RAD-IO	**Value:** 127
DLT_TZSP	**Value:** 128
DLT_ARCNET_LINUX	**Value:** 129
DLT_JUNIPER_MLPP-P	**Value:** 130
DLT_JUNIPER_MLFR	**Value:** 131
DLT_JUNIPER_ES	**Value:** 132
DLT_JUNIPER_GGSN	**Value:** 133
DLT_JUNIPER_MFR	**Value:** 134
DLT_JUNIPER_ATM2	**Value:** 135
DLT_JUNIPER_SERVI-CES	**Value:** 136
DLT_JUNIPER_ATM1	**Value:** 137
DLT_APPLE_IP_OVE-R_IEEE1394	**Value:** 138
DLT_MTP2_WITH_PH-DR	**Value:** 139
DLT_MTP2	**Value:** 140
DLT_MTP3	**Value:** 141
DLT_SCCP	**Value:** 142
DLT_DOCSIS	**Value:** 143
DLT_LINUX_IRDA	**Value:** 144
DLT_IBM_SP	**Value:** 145
DLT_IBM_SN	**Value:** 146
DLT_USER0	**Value:** 147
DLT_USER1	**Value:** 148
DLT_USER2	**Value:** 149
DLT_USER3	**Value:** 150
DLT_USER4	**Value:** 151
DLT_USER5	**Value:** 152
DLT_USER6	**Value:** 153
DLT_USER7	**Value:** 154
DLT_USER8	**Value:** 155
DLT_USER9	**Value:** 156
DLT_USER10	**Value:** 157

continued on next page

Name	Description
DLT_USER11	**Value:** 158
DLT_USER12	**Value:** 159
DLT_USER13	**Value:** 160
DLT_USER14	**Value:** 161
DLT_USER15	**Value:** 162
DLT_IEEE802_11_RAD-IO_AVS	**Value:** 163
DLT_JUNIPER_MONIT-OR	**Value:** 164
DLT_BACNET_MS_TP	**Value:** 165
DLT_PPP_PPPD	**Value:** 166
DLT_PPP_WITH_DIR-ECTION	**Value:** 166
DLT_LINUX_PPP_WI-THDIRECTION	**Value:** 166
DLT_JUNIPER_PPPOE	**Value:** 167
DLT_JUNIPER_PPPOE-_ATM	**Value:** 168
DLT_GPRS_LLC	**Value:** 169
DLT_GPF_T	**Value:** 170
DLT_GPF_F	**Value:** 171
DLT_GCOM_T1E1	**Value:** 172
DLT_GCOM_SERIAL	**Value:** 173
DLT_JUNIPER_PIC_P-EER	**Value:** 174
DLT_ERF_ETH	**Value:** 175
DLT_ERF_POS	**Value:** 176
DLT_LINUX_LAPD	**Value:** 177
DLT_JUNIPER_ETHER	**Value:** 178
DLT_JUNIPER_PPP	**Value:** 179
DLT_JUNIPER_FRELA-Y	**Value:** 180
DLT_JUNIPER_CHDL-C	**Value:** 181
DLT_MFR	**Value:** 182
DLT_JUNIPER_VP	**Value:** 183
DLT_A429	**Value:** 184
DLT_A653_ICM	**Value:** 185
DLT_USB	**Value:** 186
DLT_BLUETOOTH_HC-I_H4	**Value:** 187
DLT_IEEE802_16_MAC-_CPS	**Value:** 188

continued on next page

Name	Description
DLT_USB_LINUX	**Value:** `189`
DLT_CAN20B	**Value:** `190`
DLT_IEEE802_15_4_LINUX	**Value:** `191`
DLT_PPI	**Value:** `192`
DLT_IEEE802_16_MAC_CPS_RADIO	**Value:** `193`
DLT_JUNIPER_ISM	**Value:** `194`
DLT_IEEE802_15_4	**Value:** `195`
DLT_SITA	**Value:** `196`
DLT_ERF	**Value:** `197`
DLT_RAIF1	**Value:** `198`
DLT_IPMB	**Value:** `199`
DLT_JUNIPER_ST	**Value:** `200`
DLT_BLUETOOTH_HCI_H4_WITH_PHDR	**Value:** `201`
__package__	**Value:** `None`

25 Module killerbee.pcapdump

25.1 Variables

Name	Description
PCAPH_MAGIC_NUM	**Value:** `2712847316`
PCAPH_VER_MAJOR	**Value:** `2`
PCAPH_VER_MINOR	**Value:** `4`
PCAPH_THISZONE	**Value:** `0`
PCAPH_SIGFIGS	**Value:** `0`
PCAPH_SNAPLEN	**Value:** `65535`
DOT11COMMON_TAG	**Value:** `2`
GPS_TAG	**Value:** `30002`
__package__	**Value:** `'killerbee'`

25.2 Class PcapReader

25.2.1 Methods

__init__(*self*, *savefile*)

Opens the specified file, validates a libpcap header is present.

Parameters

`savefile`: Input libpcap filename to open

(type=String)

Return Value

None

datalink(*self*)

Returns the data link type for the packet capture.

Return Value

Int

close(*self*)

Closes the output packet capture; wrapper for pcap_close().

Return Value

None

pcap_close(*self*)

Closes the output packet capture.

Return Value

None

pnext(*self*)

Wrapper for pcap_next to mimic method for Daintree SNA. See pcap_next()

pcap_next(*self*)

Retrieves the next packet from the capture file. Returns a list of [Hdr, packet] where Hdr is a list of [timestamp, snaplen, plen] and packet is a string of the payload content. Returns None at the end of the packet capture.

Return Value

List

25.3 Class PcapDumper

25.3.1 Methods

__init__(*self*, *datalink*, *savefile*, *ppi*=`False`)

Creates a libpcap file using the specified datalink type.

Parameters

`datalink`: Datalink type, one of DLT_* defined in pcap-bpf.h

(type=Integer)

`savefile`: Output libpcap filename to open

(type=String)

Return Value

None

pcap_dump(*self*, *packet*, *ts_sec*=`None`, *ts_usec*=`None`, *orig_len*=`None`, *freq_mhz*=`None`, *ant_dbm*=`None`, *location*=`None`)

Appends a new packet to the libpcap file. Optionally specify ts_sec and tv_usec for timestamp information, otherwise the current time is used. Specify orig_len if your snaplen is smaller than the entire packet contents.

Parameters

`ts_sec`:	Timestamp, number of seconds since Unix epoch. Default is the current timestamp. *(type=Integer)*
`ts_usec`:	Timestamp microseconds. Defaults to current timestamp. *(type=Integer)*
`orig_len`:	Length of the original packet, used if the packet you are writing is smaller than the original packet. Defaults to the specified packet's length. *(type=Integer)*
`location`:	3-tuple of (longitude, latitude, altitude). *(type=Tuple)*
`packet`:	Packet contents *(type=String)*

Return Value

None

close(*self*)

Closes the output packet capture; wrapper for pcap_close().

Return Value

None

pcap_close(*self*)

Closed the output packet capture.

Return Value

None

26 Module killerbee.scapy_extensions

26.1 Functions

kbdev()

List KillerBee recognized devices

kbsendp(*pkt*, *channel*=`None`, *inter*=`0`, *loop*=`0`, *iface*=`None`, *verbose*=`None`, *realtime*=`None`)

Send a packet with KillerBee

Parameters

`channel`:	802.15.4 channel to transmit/receive on
`inter`:	time to wait between tranmissions
`loop`:	number of times to process the packet list
`iface`:	KillerBee interface to use, or KillerBee() class instance
`verbose`:	set verbosity level
`realtime`:	use packet's timestamp, bending time with realtime value

kbsrp(*pkt*, *channel*=`None`, *inter*=`0`, *count*=`0`, *iface*=`None`, *store*=`1`, *prn*=`None`, *lfilter*=`None`, *timeout*=`None`, *verbose*=`None`, *realtime*=`None`)

Send and receive packets with KillerBee

Parameters

`channel`:	802.15.4 channel to transmit/receive on
`inter`:	time to wait between tranmissions
`count`:	number of packets to capture. 0 means infinity
`iface`:	KillerBee interface to use, or KillerBee() class instance
`store`:	wether to store sniffed packets or discard them
`prn`:	function to apply to each packet. If something is returned, it is displayed. Ex: ex: prn = lambda x: x.summary()
`lfilter`:	python function applied to each packet to determine if further action may be done ex: lfilter = lambda x: x.haslayer(Padding)
`timeout`:	stop sniffing after a given time (default: None)
`verbose`:	set verbosity level
`realtime`:	use packet's timestamp, bending time with realtime value

kbsrp1(*pkt*, *channel*=`None`, *inter*=`0`, *iface*=`None`, *store*=`1`, *prn*=`None`, *lfilter*=`None`, *timeout*=`None`, *verbose*=`None`, *realtime*=`None`)

Send and receive packets with KillerBee and return only the first answer

kbsniff(*channel*=`None`, *count*=`0`, *iface*=`None`, *store*=`1`, *prn*=`None`, *lfilter*=`None`, *stop_filter*=`None`, *verbose*=`None`, *timeout*=`None`)

Sniff packets with KillerBee.

Parameters

`channel:`	802.15.4 channel to transmit/receive on
`count:`	number of packets to capture. 0 means infinity
`iface:`	KillerBee interface to use, or KillerBee() class instance
`store:`	wether to store sniffed packets or discard them
`prn:`	function to apply to each packet. If something is returned, it is displayed. Ex: ex: prn = lambda x: x.summary()
`lfilter:`	python function applied to each packet to determine if further action may be done ex: lfilter = lambda x: x.haslayer(Padding)
`timeout:`	stop sniffing after a given time (default: None)

kbrdpcap(*filename*, *count*=`-1`, *skip*=`0`, *nofcs*=`False`)

Read a pcap file with the KillerBee library. Wraps the PcapReader to return scapy packet object from pcap files. This uses the killerbee internal methods instead of the scapy native methods. This is not necessarily better, and suggestions are welcome. Specify nofcs parameter as True if for some reason the packets in the PCAP don't have FCS (checksums) at the end.

Return Value

Scapy packetlist of Dot15d4 packets parsed from the given PCAP file.

kbwrpcap(*save_file*, *pkts*)

Write a pcap using the KillerBee library.

kbrddain(*filename*, *count*=`-1`, *skip*=`0`)

Read a dain tree file with the KillerBee library Wraps the DainTreeReader to return scapy packet object from daintree files.

kbwrdain(*save_file*, *pkts*)

Write a daintree file using the KillerBee library.

kbkeysearch(*packet*, *searchdata*, *ispath*=`True`, *skipfcs*=`True`, *raw*=`False`)

Search a binary file for the encryption key to an encrypted packet.

kbgetnetworkkey(*pkts*)

Search packets for a plaintext key exchange returns the first one found.

kbtshark(*store*=`0`, **args*, ***kargs*)

Sniff packets using KillerBee and print them calling pkt.show()

kbrandmac(*length*=`8`)

Returns a random MAC address using a list valid OUI's from ZigBee device manufacturers.

kbdecrypt(*pkt*, *key*=`None`, *verbose*=`None`)

Decrypt Zigbee frames using AES CCM* with 32-bit MIC

kbencrypt(*pkt*, *data*, *key*=`None`, *verbose*=`None`)

Encrypt Zigbee frames using AES CCM* with 32-bit MIC

26.2 Variables

Name	Description
DEFAULT_KB_CHANN-EL	**Value:** `11`
DEFAULT_KB_DEVIC-E	**Value:** `None`
log_killerbee	**Value:** `logging.getLogger('scapy.killerbee')`
__package__	**Value:** `'killerbee'`

27 Package killerbee.zbwardrive

27.1 Modules

27.2 Variables

Name	Description
__package__	**Value:** `'killerbee.zbwardrive'`

28 Module killerbee.zbwardrive.capture

28.1 Functions

startCapture(*zbdb*, *channel*, *dblog*=`False`, *gps*=`False`)

Before calling, you should have already ensured the channel or the channel which the key is associated with does not already have an active capture occuring.

interrupt(*signum*, *frame*)

28.2 Variables

Name	Description
triggers	**Value:** `[]`
__package__	**Value:** `'killerbee.zbwardrive'`

28.3 Class CaptureThread

object
threading._Verbose
threading.Thread
killerbee.zbwardrive.capture.CaptureThread

28.3.1 Methods

__init__(*self*, *channel*, *devstring*, *trigger*, *dblog*=`False`, *gps*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

run(*self*)

Overrides: threading.Thread.run

Inherited from threading.Thread

___repr___(), getName(), isAlive(), isDaemon(), is_alive(), join(), setDaemon(), setName(), start()

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

28.3.2 Properties

Name	Description
Inherited from threading.Thread daemon, ident, name	
Inherited from object ___class___	

29 Module killerbee.zbwardrive.db

29.1 Functions

toHex(*bin*)

29.2 Variables

Name	Description
__package__	**Value:** `'killerbee.zbwardrive'`

29.3 Class ZBScanDB

API to interact with the "database" storing information for the zbscanning program.

29.3.1 Methods

__init__(*self*)

close(*self*)

store_devices(*self*, *devid*, *devstr*, *devserial*)

get_devices_nextFree(*self*)

update_devices_status(*self*, *devid*, *newstatus*)

update_devices_start_capture(*self*, *devid*, *channel*)

store_networks(*self*, *key*, *spanid*, *source*, *channel*, *packet*)

get_networks_channel(*self*, *key*)

channel_status_logging(*self*, *chan*)

Returns False if we have not seen the network or are not currently logging it's channel, and returns True if we are currently logging it. @return boolean

30 Package killerbee.zbwardrive.gps

30.1 Modules

- **client** *(Section 31, p. 99)*
- **gps** *(Section* **??***, p.* **??***)*
- **gps'** *(Section 32, p. 102)*
- **misc** *(Section 33, p. 105)*

30.2 Variables

Name	Description
api_major_version	**Value:** `4`
api_minor_version	**Value:** `1`
AIS_SET	**Value:** `268435456`
ALTITUDE_SET	**Value:** `16`
ATTITUDE_SET	**Value:** `16384`
AUXDATA_SET	**Value:** `2147483648`
CLIMBERR_SET	**Value:** `2097152`
CLIMB_SET	**Value:** `128`
DEVICEID_SET	**Value:** `16777216`
DEVICELIST_SET	**Value:** `8388608`
DEVICE_SET	**Value:** `4194304`
DOP_SET	**Value:** `1024`
ERROR_SET	**Value:** `33554432`
GPSD_PORT	**Value:** `'2947'`
HERR_SET	**Value:** `4096`
KNOTS_TO_KPH	**Value:** `1.852`
KNOTS_TO_MPH	**Value:** `1.1507794`
KNOTS_TO_MPS	**Value:** `0.51444444`
LATLON_SET	**Value:** `8`
MAXCHANNELS	**Value:** `20`
METERS_TO_FEET	**Value:** `3.2808399`
METERS_TO_MILES	**Value:** `0.00062137119`
MODE_2D	**Value:** `2`
MODE_3D	**Value:** `3`
MODE_NO_FIX	**Value:** `1`
MODE_SET	**Value:** `512`
MPS_TO_KNOTS	**Value:** `1.9438445`
MPS_TO_KPH	**Value:** `3.6`
MPS_TO_MPH	**Value:** `2.2369363`
NaN	**Value:** `nan`
ONLINE_SET	**Value:** `1`

continued on next page

Name	Description
PACKET_SET	**Value:** `536870912`
POLICY_SET	**Value:** `32768`
RAW_SET	**Value:** `131072`
RTCM2_SET	**Value:** `67108864`
RTCM3_SET	**Value:** `134217728`
SATELLITE_SET	**Value:** `65536`
SIGNAL_STRENGTH_-UNKNOWN	**Value:** `nan`
SPEEDERR_SET	**Value:** `524288`
SPEED_SET	**Value:** `32`
STATUS_DGPS_FIX	**Value:** `2`
STATUS_FIX	**Value:** `1`
STATUS_NO_FIX	**Value:** `0`
STATUS_SET	**Value:** `256`
TIMERR_SET	**Value:** `4`
TIME_SET	**Value:** `2`
TRACKERR_SET	**Value:** `1048576`
TRACK_SET	**Value:** `64`
UNION_SET	**Value:** `511707136`
USED_SET	**Value:** `262144`
VERR_SET	**Value:** `8192`
VERSION_SET	**Value:** `2048`
WATCH_DEVICE	**Value:** `64`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NEWSTYLE	**Value:** `128`
WATCH_NMEA	**Value:** `4`
WATCH_OLDSTYLE	**Value:** `65536`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
___package___	**Value:** `'killerbee.zbwardrive.gps'`

31 Module killerbee.zbwardrive.gps.client

31.1 Variables

Name	Description
GPSD_PORT	**Value:** `'2947'`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NMEA	**Value:** `4`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
WATCH_DEVICE	**Value:** `64`
___package___	**Value:** `'killerbee.zbwardrive.gps'`

31.2 Class gpscommon

Known Subclasses: killerbee.zbwardrive.gps.client.gpsjson

Isolate socket handling and buffering from the protcol interpretation.

31.2.1 Methods

___init___(*self*, *host*=`'127.0.0.1'`, *port*=`'2947'`, *verbose*=`0`)

connect(*self*, *host*, *port*)

Connect to a host on a given port.

If the hostname ends with a colon (':') followed by a number, and there is no port specified, that suffix will be stripped off and the number interpreted as the port number to use.

close(*self*)

___del___(*self*)

waiting(*self*)

Return True if data is ready for the client.

read(*self*)

Wait for and read data being streamed from the daemon.

send(*self*, *commands*)

Ship commands to the daemon.

31.3 Class gpsjson

killerbee.zbwardrive.gps.client.gpscommon
└ **killerbee.zbwardrive.gps.client.gpsjson**

Known Subclasses: killerbee.zbwardrive.gps.gps'.gps

Basic JSON decoding.

31.3.1 Methods

__iter__(*self*)

json_unpack(*self*, *buf*)

stream(*self*, *flags*=`0`, *outfile*=`None`)

Control streaming reports from the daemon,

Inherited from killerbee.zbwardrive.gps.client.gpscommon(Section 31.2)

__del__(), __init__(), close(), connect(), read(), send(), waiting()

31.4 Class dictwrapper

Wrapper that yields both class and dictionary behavior,

31.4.1 Methods

__init__(*self*, ***ddict*)

get(*self*, *k*, *d*=`None`)

keys(*self*)

__getitem__(*self*, *key*)

Emulate dictionary, for new-style interface.

__setitem__(*self*, *key*, *val*)

Emulate dictionary, for new-style interface.

__contains__(*self*, *key*)

__str__(*self*)

__repr__(*self*)

32 Module killerbee.zbwardrive.gps.gps'

32.1 Functions

isnan(*x*)

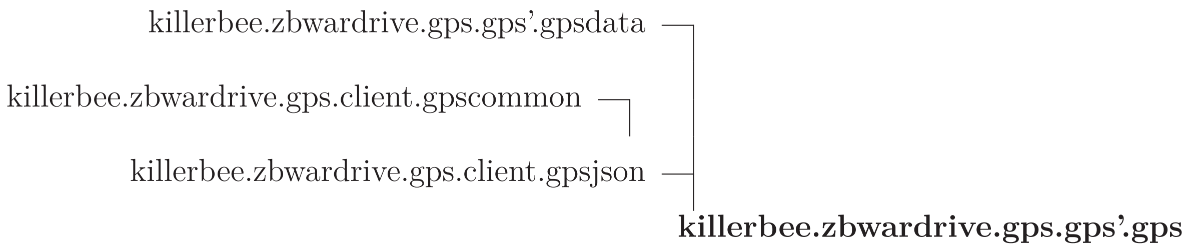
32.2 Variables

Name	Description
AIS_SET	**Value:** `268435456`
ALTITUDE_SET	**Value:** `16`
ATTITUDE_SET	**Value:** `16384`
AUXDATA_SET	**Value:** `2147483648`
CLIMBERR_SET	**Value:** `2097152`
CLIMB_SET	**Value:** `128`
DEVICEID_SET	**Value:** `16777216`
DEVICELIST_SET	**Value:** `8388608`
DEVICE_SET	**Value:** `4194304`
DOP_SET	**Value:** `1024`
ERROR_SET	**Value:** `33554432`
GPSD_PORT	**Value:** `'2947'`
HERR_SET	**Value:** `4096`
LATLON_SET	**Value:** `8`
MAXCHANNELS	**Value:** `20`
MODE_2D	**Value:** `2`
MODE_3D	**Value:** `3`
MODE_NO_FIX	**Value:** `1`
MODE_SET	**Value:** `512`
NaN	**Value:** `nan`
ONLINE_SET	**Value:** `1`
PACKET_SET	**Value:** `536870912`
POLICY_SET	**Value:** `32768`
RAW_SET	**Value:** `131072`
RTCM2_SET	**Value:** `67108864`
RTCM3_SET	**Value:** `134217728`
SATELLITE_SET	**Value:** `65536`
SIGNAL_STRENGTH_-UNKNOWN	**Value:** `nan`
SPEEDERR_SET	**Value:** `524288`
SPEED_SET	**Value:** `32`
STATUS_DGPS_FIX	**Value:** `2`
STATUS_FIX	**Value:** `1`

continued on next page

Name	Description
STATUS_NO_FIX	**Value:** `0`
STATUS_SET	**Value:** `256`
TIMERR_SET	**Value:** `4`
TIME_SET	**Value:** `2`
TRACKERR_SET	**Value:** `1048576`
TRACK_SET	**Value:** `64`
UNION_SET	**Value:** `511707136`
USED_SET	**Value:** `262144`
VERR_SET	**Value:** `8192`
VERSION_SET	**Value:** `2048`
WATCH_DEVICE	**Value:** `64`
WATCH_DISABLE	**Value:** `0`
WATCH_ENABLE	**Value:** `1`
WATCH_JSON	**Value:** `2`
WATCH_NEWSTYLE	**Value:** `128`
WATCH_NMEA	**Value:** `4`
WATCH_OLDSTYLE	**Value:** `65536`
WATCH_RARE	**Value:** `8`
WATCH_RAW	**Value:** `16`
WATCH_SCALED	**Value:** `32`
__package__	**Value:** `'killerbee.zbwardrive.gps'`

32.3 Class gps


killerbee.zbwardrive.gps.gps'.gpsdata
killerbee.zbwardrive.gps.client.gpscommon
killerbee.zbwardrive.gps.client.gpsjson
killerbee.zbwardrive.gps.gps'.gps

Client interface to a running gpsd instance.

32.3.1 Methods

__init__(*self*, *host*=`'127.0.0.1'`, *port*=`'2947'`, *verbose*=`0`, *mode*=`0`)

Overrides: killerbee.zbwardrive.gps.client.gpscommon.__init__

next(*self*)

poll(*self*)

Read and interpret data from the daemon.

set_raw_hook(*self*, *hook*)

stream(*self*, *flags*=`0`, *outfile*=`None`)

Ask gpsd to stream reports at your client.

Overrides: killerbee.zbwardrive.gps.client.gpsjson.stream

Inherited from killerbee.zbwardrive.gps.gps'.gpsdata(Section 32.4)

___repr___()

Inherited from killerbee.zbwardrive.gps.client.gpsjson(Section 31.3)

___iter___(), json_unpack()

Inherited from killerbee.zbwardrive.gps.client.gpscommon(Section 31.2)

___del___(), close(), connect(), read(), send(), waiting()

32.4 Class gpsdata

Known Subclasses: killerbee.zbwardrive.gps.gps'.gps

Position, track, velocity and status information returned by a GPS.

32.4.1 Methods

___init___(*self*)

___repr___(*self*)

32.5 Class gpsfix

32.5.1 Methods

___init___(*self*)

33 Module killerbee.zbwardrive.gps.misc

33.1 Functions

Deg2Rad(*x*)

Degrees to radians.

Rad2Deg(*x*)

Radians to degress.

CalcRad(*lat*)

Radius of curvature in meters at specified latitude.

EarthDistance(*(lat1, lon1)*, *(lat2, lon2)*)

Distance in meters between two points specified in degrees.

MeterOffset(*(lat1, lon1)*, *(lat2, lon2)*)

Return offset in meters of second arg from first.

isotime(*s*)

Convert timestamps in ISO8661 format to and from Unix time.

33.2 Variables

Name	Description
METERS_TO_FEET	**Value:** `3.2808399`
METERS_TO_MILES	**Value:** `0.00062137119`
KNOTS_TO_MPH	**Value:** `1.1507794`
KNOTS_TO_KPH	**Value:** `1.852`
KNOTS_TO_MPS	**Value:** `0.51444444`
MPS_TO_KPH	**Value:** `3.6`
MPS_TO_MPH	**Value:** `2.2369363`
MPS_TO_KNOTS	**Value:** `1.9438445`
__package__	**Value:** `'killerbee.zbwardrive.gps'`

34 Module killerbee.zbwardrive.scanning

34.1 Functions

doScan_processResponse(*packet*, *channel*, *zbdb*, *kbscan*, *verbose*=`False`, *dblog*=`False`)

doScan(*zbdb*, *currentGPS*, *verbose*=`False`, *dblog*=`False`, *agressive*=`False`, *staytime*=`2`)

34.2 Variables

Name	Description
MIN_ITERATIONS_AG-RESSIVE	**Value:** `0`
__package__	**Value:** `'killerbee.zbwardrive'`

35 Module killerbee.zbwardrive.testGPS

35.1 Variables

Name	Description
session	**Value:** `gps.gps()`

36 Module killerbee.zbwardrive.zbwardrive

36.1 Functions

gpsdPoller(*currentGPS*)

@type currentGPS multiprocessing.Manager dict manager @arg currentGPS store relavent pieces of up-to-date GPS info

startScan(*zbdb*, *currentGPS*, *verbose*=`False`, *dblog*=`False`, *agressive*=`False`, *include*=`[]`, *ignore*=`None`)

36.2 Variables

Name	Description
GPS_FREQUENCY	**Value:** `3`
___package___	**Value:** `'killerbee.zbwardrive'`

37 Module killerbee.zigbeedecode

37.1 Variables

Name	Description
ZBEE_NWK_FCF_FRAME_TYPE	ZigBee NWK Frame Control Frame Type **Value:** `3`
ZBEE_NWK_FCF_VERSION	ZigBee NWK Frame Control Version **Value:** `60`
ZBEE_NWK_FCF_DISCOVER_ROUTE	ZigBee NWK Frame Control Route Topology Discovery Flag **Value:** `192`
ZBEE_NWK_FCF_MULTICAST	ZigBee NWK Frame Control Multicast Flag, ZigBee 2006 and Later **Value:** `256`
ZBEE_NWK_FCF_SECURITY	ZigBee NWK Frame Control Security Bit **Value:** `512`
ZBEE_NWK_FCF_SOURCE_ROUTE	ZigBee NWK Frame Control Source Route Bit, ZigBee 2006 and Later **Value:** `1024`
ZBEE_NWK_FCF_EXT_DEST	ZigBee NWK Frame Control Extended Destination Addressing, ZigBee 2006 and Later **Value:** `2048`
ZBEE_NWK_FCF_EXT_SOURCE	ZigBee NWK Frame Control Extended Source Addressing, ZigBee 2006 and Later **Value:** `4096`
ZBEE_NWK_FCF_DATA	ZigBee NWK Frame Control Field Frame Type: Data **Value:** `0`
ZBEE_NWK_FCF_CMD	ZigBee NWK Frame Control Field Frame Type: Command **Value:** `1`
ZBEE_APS_FCF_FRAME_TYPE	ZigBee APS Frame Control Frame Type **Value:** `3`
ZBEE_APS_FCF_DELIVERY_MODE	ZigBee APS Frame Control Delivery Mode **Value:** `12`
ZBEE_APS_FCF_INDIRECT_MODE	ZigBee APS Frame Control Indirect Delivery Mode Flag, ZigBee 2004 and earlier. **Value:** `16`
ZBEE_APS_FCF_ACK_MODE	ZigBee APS Frame Control ACK Mode, ZigBee 2007 and later. **Value:** `16`
ZBEE_APS_FCF_SECURITY	ZigBee APS Frame Control Security Bit **Value:** `32`

continued on next page

Name	Description
ZBEE_APS_FCF_ACK-_REQ	ZigBee APS Frame Control ACK Required Bit **Value:** `64`
ZBEE_APS_FCF_EXT-_HEADER	ZigBee APS Frame Control Extended Header Bit **Value:** `128`
ZBEE_APS_FCF_DAT-A	ZigBee APS Frame Control Field Frame Type: Data **Value:** `0`
ZBEE_APS_FCF_CMD	ZigBee APS Frame Control Field Frame Type: Command **Value:** `1`
ZBEE_APS_FCF_ACK	ZigBee APS Frame Control Field Frame Type: ACK **Value:** `2`
ZBEE_APS_FCF_UNIC-AST	ZigBee APS Frame Control Field Delivery Mode: Unicast Delivery **Value:** `0`
ZBEE_APS_FCF_INDI-RECT	ZigBee APS Frame Control Field Delivery Mode: Indirect Delivery **Value:** `1`
ZBEE_APS_FCF_BCA-ST	ZigBee APS Frame Control Field Delivery Mode: Broadcast Delivery **Value:** `2`
ZBEE_APS_FCF_GRO-UP	ZigBee APS Frame Control Field Delivery Mode: Group Delivery, ZigBee 2006 and later. **Value:** `3`
___package___	**Value:** `'killerbee'`

37.2 Class ZigBeeNWKPacketParser

37.2.1 Methods

___init___(*self*)

Instantiates the ZigBeeNWKPacketParser class.

pktchop(*self*, *packet*)

Chops up the specified packet contents into a list of fields. Does not attempt to re-order the field values for parsing. ".join(X) will reassemble original packet string. Fields which may or may not be present (such as the destination address) are empty if they are not present, keeping the list elements consistent, as follows: Frame Control | DA | SA | Radius | Seq # | Dst IEEE Address | Src IEEE Address | MCast Ctrl | Src Route Subframe | Payload

An exception is raised if the packet contents are too short to decode.

Parameters

`packet`: Packet contents.

(type=String)

Return Value

Chopped contents of the ZigBee NWK packet into list elements.

(type=list)

hdrlen(*self*, *packet*)

Returns the length of the ZigBee NWK header.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the ZigBEE NWK header.

(type=Int)

payloadlen(*self*, *packet*)

Returns the length of the NWK payload.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the NWK payload.

(type=Int)

37.3 Class ZigBeeAPSPacketParser

37.3.1 Methods

__init__(*self*)

Instantiates the ZigBeeAPSPacketParser class.

pktchop(*self*, *packet*)

Chops up the specified packet contents into a list of fields. Does not attempt to re-order the field values for parsing. ”.join(X) will reassemble original packet string. Fields which may or may not be present (such as the destination endpoint) are empty if they are not present, keeping the list elements consistent, as follows: Frame Control | Dst Endpoint | Group Address | Cluster Identifier | Profile Identifier | Source Endpoint | APS Counter | Payload

An exception is raised if the packet contents are too short to decode.

Parameters

`packet`: Packet contents.

(type=String)

Return Value

Chopped contents of the ZigBee APS packet into list elements.

(type=list)

hdrlen(*self*, *packet*)

Returns the length of the ZigBee NWK header.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the ZigBEE NWK header.

(type=Int)

payloadlen(*self*, *packet*)

Returns the length of the APS payload.

Parameters

`packet`: Packet contents to evaluate for header length.

(type=String)

Return Value

Length of the APS payload.

(type=Int)

Index