pyparrot Documentation

Release 1.5.3

Amy McGovern

Jan 05, 2020

Contents

pyparrot was designed by Dr. Amy McGovern to program Parrot Minidrone (primarily Mambo FPV but Swing is also supported) and Parrot Bebop (1 or 2) drones using Python. This interface was developed to teach kids of all ages (K-20) STEM concepts (programming, math, and more) by having them program a drone to fly autonomously. Anyone can use it who is interested in autonomous drone programming!

CHAPTER 1

Main documentation

1.1 Installation

You have two choices for installing pyparrot: using the `source` code directly or downloading with `pip`. **Note** Pyparrot will only work with python 3. This choice was made because the support for multi-threaded programs is improved in python 3.

1.1.1 Requirements

The choice of related packages is dependent on your choice of drone (Mambo, Mambo FPV, Bebop 1 or 2, Swing, Anafi) and to the operating system that you will be using to develop.

Hardware/Drone requirements

- **Parrot Mambo FPV**: If you have a Mambo FPV (e.g. you have the camera), you can use the wifi interface. The wifi interface will work on Mac, Linux, or Windows.
- **Parrot Mambo Fly or Code**: If you have a Mambo without the camera, you will use the BLE interface. pyparrot currently only supports Linux for BLE. The BLE interface was developed on a Raspberry Pi 3 Model B but it has been tested on other Linux machines.
- **Parrot Swing**: To use the Swing you will use the BLE interface.
- **Parrot Bebop 2**: The Bebop interface was tested on a Bebop 2 using a laptop with wifi (any wifi enabled device should work).
- **Parrot Bebop 1**: A Bebop 1 will also work with any wifi enabled device.
- **Parrot Anafi**: Per the development board, the Anafi should work with very minor changes. I will work to officially suppport it once the SDK from parrot is released for it.

Software requirements

Software requirements are listed below by type of connection to the drone.

- All drones: Python 3

  I use the https://www.anaconda.com/download/:: installer and package manager for python. Note, when you install anaconda, install the Visual Studio option, especially if you have windows. Otherwise you will need to install Visual Studio separately. The zeroconf package (listed below) requires developer tools because it needs to be compiled.

- All drones: untangle package (this is used to parse the xml files in the parrot SDK)

```
pip install untangle
```

- Vision: If you intend to process the camera files, you will need to install opencv and then either ffmpeg

or VLC. I installed ffmpeg using brew for the mac but apt-get on linux should also work. For VLC, you MUST install the actual **'VLC <https://www.videolan.org/vlc/index.html'_** program (and not just the library in python) and it needs to be version 3.0.1 or greater.

- Wifi connection: zeroconf To install zeroconf software do the following:

```
pip install zeroconf
```

- BLE connection: pybluez (note this is ONLY for support without the camera!) This is ONLY supported on linux.

To install the BLE software do the following:

```
sudo apt-get install bluetooth
sudo apt-get install bluez
sudo apt-get install python-bluez
```

Note it is also possible that you will need to install bluepy (if it isn't already there). These commands should do it:

```
sudo apt-get install python-pip libglib2.0-dev
sudo pip install bluepy
sudo apt-get update
```

1.1.2 Installing From Source

First download pyparrot by cloning the repository from https://github.com/amymcgovern/pyparrot The instructions for this are below.

```
git clone https://github.com/amymcgovern/pyparrot
cd pyparrot
```

Make sure you install the necessary other packages (wifi or BLE, vision, etc) as specified above.

1.1.3 Installing From Pip

To install from pip, type

```
pip install pyparrot
```

Make sure you install the necessary other packages (wifi or BLE, vision, etc) as specified above.

1.1.4 Installation guide for windows users who might need more help

Thank you to @JackdQuinn for contributing this.

Make sure you install **Visual Studio** either using Anaconda or by downloading it from Microsoft. Note that Visual Studio is free but it is required for compilation of the wifi module zeroconf, and specifically of the netifaces module that zeroconf requires. It is a very large download if you chose to do it outside of anaconda so you will want to start that download first.

If you install python without anaconda, when you install choose Special install Python and click add python to path (this will clear up some command line call issues).

Again, if you chose regular python and not anaconda, you can check installation by typing py in the windows command line.

```
py
```

Once you are sure that python started, you will want to quit python. type: `quit()` to exit python

```
quit()
```

If you chose to use anaconda, bring up the anaconda menu and open an anaconda prompt to verify that it installed. The rest of the instructions depend on whether you chose python or anaconda for your installation. If you chose python, use the windows command prompt for pip. If you chose anaconda, use your anaconda prompt.

If you type the pip command (with no options), it will produce a long list of options. This tells you that you are at the right command prompt to do the rest of the installation. **Note, the pip command will not work inside of python.** This is a command prompt command, not a python command.

```
pip
```

Sometimes pip tells you that it wants to upgrade. For windows, the command is:

```
python -m pip install -U pip
```

To actually install, use the commands described above (and repeated here).

```
pip install untangle
pip install pyparrot
pip install zeroconf
```

Note that visual studio is a requirement for zeroconf

Testing your install

The first step is to connect your connect your controlling device (laptop, computer, etc) to the wifi for the drone. Look for a wifi network named Mambo_number where number changes for each drone.

After connection to your drone its time to run code! You can download all the example code from these docs. Below is a short set of commands of how to run that code.

Run code by cd'ing down to the directory (the folder your python code is in) and running the desired python file from the cmd line

Example:

- open command line either through windows or anaconda (depending on your installation method)
- type: `cd desktop`

- this will Change your Directory to the desktop
- type: `dir`
- this will display a list of all the folders (directories) on the desktop
- type: `cd yourFolderNameHere`
- type: `dir`
- this will display all the files and folders in the directory
- type: `py TheNameOfTheFileYouWantToRun.py` or `python TheNameOfTheFileYouWantToRun.py`
- When you click enter the file will begin to run, if you are using the demo scripts you should see lots of nice feedback as it changes states. You can use the arrow keys to go through your history of commands which can save you lots of time if your file names are long.
- If you have several connects and disconnects try restarting your computer or resetting your ip (for the more technically inclined)
- If you have crashes where the drone is flipping to one side when it shouldn't check the blades and bumpers. The bumpers can shift after a crash and prevent the blades from spinning, or slow down their spin, which causes unintended flips

1.2 Quick Start Guide with a Minidrone

1.2.1 Using the pyparrot library on the Minidrone

Before running any of the sample code, you will need to connect to your drone. If you have a Mambo FPV, I highly recommend using the wifi connection since it sends much more information using wifi than BLE. If you have a Mambo Code or a Mambo Fly or Swing(neither of which has a camera), then you need to use the BLE connection.

wifi connection

If you are using the wifi (e.g. Mambo FPV), you need to connect your controlling device (laptop, computer, etc) to the wifi for the drone. Look for a wifi network named Mambo_number where number changes for each drone.

BLE connection

If you do not have a camera or want to use BLE for other reasons(e.g. swarm), you will first need to find the BLE address of your Minidrone(s). BLE permissions on linux require that this command run in sudo mode. To run this, from the bin directory for your python installation, type:

```
sudo findMinidrone
```

This will identify all BLE devices within hearing of the Pi. The Minidrone's specific address will be printed at the end. Save the address and use it in your connection code (discussed below). If findMinidrone does not report "FOUND A MAMBO!" or "FOUND A SWING!", then be sure your minidrone is turned on when you run the findMambo code and that your Pi (or other linux box) has its BLE interface turned on.

The output should look something like this. I removed my own BLE addresses from my network for security but I am showing the address of the mambo that I use for all the demo scripts.

```
~/miniconda3/bin $ sudo ./find_mambo
Discovered device <address removed>
Discovered device <address removed>
Discovered device <address removed>
Discovered device e0:14:d0:63:3d:d0
Received new data from <address removed>
Discovered device <address removed>
Discovered device <address removed>
Received new data from <address removed>
Discovered device <address removed>
FOUND A MAMBO!
Device e0:14:d0:63:3d:d0 (random), RSSI=-60 dB
  Complete Local Name = Mambo_<numbers>
```

1.2.2 Quick start: Demo Code

I have provided a set of example scripts for both the Mambo and the Bebop. Note that you will need to edit the minidrone scripts to either use your own BLE address or to ensure that use_wifi=True is set, so that it connects using wifi. **Note that you do not need to run any of the other code in sudo mode!** That was only for discovery.

Demo of the trick commands on the mambo

The code shown below is the demoMamboTricks.py. demoMamboTricks.py will take off, demonstrate all 4 types of flips, and then land. It is a good program to verify that your connection to your mambo is working well. Be sure to run it in a room large enough to perform the flips! The highlighted lines need to change for YOUR mambo and connection choices.

```python
"""
Demo the trick flying for the python interface

Author: Amy McGovern
"""

from pyparrot.Minidrone import Mambo

# you will need to change this to the address of YOUR mambo
mamboAddr = "e0:14:d0:63:3d:d0"

# make my mambo object
# remember to set True/False for the wifi depending on if you are using the wifi or␣
↪the BLE to connect
mambo = Mambo(mamboAddr, use_wifi=True)

print("trying to connect")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

if (success):
    # get the state information
    print("sleeping")
    mambo.smart_sleep(2)
    mambo.ask_for_state_update()
    mambo.smart_sleep(2)
```

(continues on next page)

(continued from previous page)

```
    print("taking off!")
    mambo.safe_takeoff(5)

    if (mambo.sensors.flying_state != "emergency"):
        print("flying state is %s" % mambo.sensors.flying_state)
        print("Flying direct: going up")
        mambo.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=20, duration=1)

        print("flip left")
        print("flying state is %s" % mambo.sensors.flying_state)
        success = mambo.flip(direction="left")
        print("mambo flip result %s" % success)
        mambo.smart_sleep(5)

        print("flip right")
        print("flying state is %s" % mambo.sensors.flying_state)
        success = mambo.flip(direction="right")
        print("mambo flip result %s" % success)
        mambo.smart_sleep(5)

        print("flip front")
        print("flying state is %s" % mambo.sensors.flying_state)
        success = mambo.flip(direction="front")
        print("mambo flip result %s" % success)
        mambo.smart_sleep(5)

        print("flip back")
        print("flying state is %s" % mambo.sensors.flying_state)
        success = mambo.flip(direction="back")
        print("mambo flip result %s" % success)
        mambo.smart_sleep(5)

        print("landing")
        print("flying state is %s" % mambo.sensors.flying_state)
        mambo.safe_land(5)
        mambo.smart_sleep(5)

    print("disconnect")
    mambo.disconnect()
```

Demo of the direct flight commands on the mambo

The second example program shows how to directly fly the mambo by controlling the yaw, pitch, roll, and vertical movement parameters. **Make sure you try this one in a large enough room!** This code is provided in demoMamboDirectFlight.py and is also shown below. Again, the highlighted lines must be changed to the parameters for your mambo and connection.

```
"""
Demo the direct flying for the python interface

Author: Amy McGovern
"""

from pyparrot.Minidrone import Mambo
```

(continues on next page)

(continued from previous page)

```
# you will need to change this to the address of YOUR mambo
mamboAddr = "e0:14:d0:63:3d:d0"

# make my mambo object
# remember to set True/False for the wifi depending on if you are using the wifi or␣
↪the BLE to connect
mambo = Mambo(mamboAddr, use_wifi=True)

print("trying to connect")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

if (success):
    # get the state information
    print("sleeping")
    mambo.smart_sleep(2)
    mambo.ask_for_state_update()
    mambo.smart_sleep(2)

    print("taking off!")
    mambo.safe_takeoff(5)

    print("Flying direct: going forward (positive pitch)")
    mambo.fly_direct(roll=0, pitch=50, yaw=0, vertical_movement=0, duration=1)

    print("Showing turning (in place) using turn_degrees")
    mambo.turn_degrees(90)
    mambo.smart_sleep(2)
    mambo.turn_degrees(-90)
    mambo.smart_sleep(2)

    print("Flying direct: yaw")
    mambo.fly_direct(roll=0, pitch=0, yaw=50, vertical_movement=0, duration=1)

    print("Flying direct: going backwards (negative pitch)")
    mambo.fly_direct(roll=0, pitch=-50, yaw=0, vertical_movement=0, duration=0.5)

    print("Flying direct: roll")
    mambo.fly_direct(roll=50, pitch=0, yaw=0, vertical_movement=0, duration=1)

    print("Flying direct: going up")
    mambo.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=50, duration=1)

    print("Flying direct: going around in a circle (yes you can mix roll, pitch, yaw␣
↪in one command!)")
    mambo.fly_direct(roll=25, pitch=0, yaw=50, vertical_movement=0, duration=3)

    print("landing")
    mambo.safe_land(5)
    mambo.smart_sleep(5)

    print("disconnect")
    mambo.disconnect()
```

Demo of the USB claw accessory

If your mambo has the USB accessories (claw and gun), you can control them but you *MUST* be in BLE mode. The mambo can only handle one USB accessory at a time and the camera counts as a USB accessory so you must use the BLE connection only. demoMamboClaw.py show how to use the claw accessory. The highlighted line must be changed to the BLE address for your mambo and the use_wifi parameter must stay at False. In this demo program, the mambo takes off, opens and closes the claw, and lands again.

```
"""
Demo the claw for the python interface

Author: Amy McGovern
"""

from pyparrot.Minidrone import Mambo

# you will need to change this to the address of YOUR mambo
mamboAddr = "e0:14:d0:63:3d:d0"

# make my mambo object
# remember you can't use the claw with the camera installed so this must be BLE␣
↪connected to work
mambo = Mambo(mamboAddr, use_wifi=False)

print("trying to connect")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

# get the state information
print("sleeping")
mambo.smart_sleep(2)
mambo.ask_for_state_update()
mambo.smart_sleep(2)

print("taking off!")
mambo.safe_takeoff(5)

print("open and close the claw")
mambo.open_claw()
# you have to sleep to let the claw open (it needs time to do it)
mambo.smart_sleep(5)

mambo.close_claw()
# you have to sleep to let the claw close (it needs time to do it)
mambo.smart_sleep(5)

print("landing")
mambo.safe_land(5)
mambo.smart_sleep(5)

print("disconnect")
mambo.disconnect()
```

Demo of the USB gun accessory

demoMamboGun.py show how to use the gun accessory. The highlighted line must be changed to the BLE address for your mambo and the use_wifi parameter must stay at False. In this demo program, the mambo takes off, fires the gun, and lands again.

```
"""
Demo the gun for the python interface

Author: Amy McGovern
"""

from pyparrot.Minidrone import Mambo

# you will need to change this to the address of YOUR mambo
mamboAddr = "e0:14:d0:63:3d:d0"

# make my mambo object
# remember you can't use the gun with the camera installed so this must be BLE␣
↪connected to work
mambo = Mambo(mamboAddr, use_wifi=False)

print("trying to connect")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

# get the state information
print ("sleeping")
mambo.smart_sleep(2)
mambo.ask_for_state_update()
mambo.smart_sleep(2)

print("shoot the gun")
mambo.fire_gun()

# sleep to ensure it does the firing
mambo.smart_sleep(15)

print("disconnect")
mambo.disconnect()
```

Demo of the ground-facing camera

demoMamboGroundcam.py show how to use the mambo's ground-facing camera. This feature **ONLY** works in wifi mode. It can be slow to download the frames so do not count on this running at several frames per second. The example code shown below takes off, takes a picture, and then grabs a random picture from the ground facing camera set.

```
"""
Demo of the groundcam
Mambo takes off, takes a picture and shows a RANDOM frame, not the last one
Author: Valentin Benke, https://github.com/Vabe7
Author: Amy McGovern
"""

from pyparrot.Minidrone import Mambo
import cv2

mambo = Mambo(None, use_wifi=True) #address is None since it only works with WiFi␣
↪anyway
```

(continues on next page)

(continued from previous page)

```python
print("trying to connect to mambo now")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

if (success):
    # get the state information
    print("sleeping")
    mambo.smart_sleep(1)
    mambo.ask_for_state_update()
    mambo.smart_sleep(1)
    mambo.safe_takeoff(5)

    # take the photo
    pic_success = mambo.take_picture()

    # need to wait a bit for the photo to show up
    mambo.smart_sleep(0.5)

    picture_names = mambo.groundcam.get_groundcam_pictures_names() #get list of␣
↪availible files
    print(picture_names)

    frame = mambo.groundcam.get_groundcam_picture(picture_names[0],True) #get frame␣
↪which is the first in the array

    if frame is not None:
        if frame is not False:
            cv2.imshow("Groundcam", frame)
            cv2.waitKey(100)

    mambo.safe_land(5)
    mambo.disconnect()
```

Demo of the flying mode on the swing

demoSwingDirectFlight.py You can see how to use the set_flying_mode command. I advise you to have enough space to use this script.

```python
"""
Demo the direct flying for the python interface

Author: Victor804
"""

from pyparrot.Minidrone import Swing

# you will need to change this to the address of YOUR swing
swingAddr = "e0:14:04:a7:3d:cb"

# make my swing object
swing = Swing(swingAddr)

print("trying to connect")
success = swing.connect(num_retries=3)
print("connected: %s" % success)
```

(continues on next page)

(continued from previous page)

```

if (success):
    # get the state information
    print("sleeping")
    swing.smart_sleep(2)
    swing.ask_for_state_update()
    swing.smart_sleep(2)

    print("taking off!")
    swing.safe_takeoff(5)

    print("plane forward")
    swing.set_flying_mode("plane_forward")

    swing.smart_sleep(1)

    print("quadricopter")
    swing.set_flying_mode("quadricopter")

    print("landing")
    swing.safe_land(5)
   swing.smart_sleep(5)

   print("disconnect")
   swing.disconnect()
```

Demo joystick for Swing

demoSwingJoystick.py Example code to control the swig with a joystick. Easy to modify for your needs.

```
import pygame
import sys
from pyparrot.Minidrone import Swing

def joystick_init():
    """
    Initializes the controller, allows the choice of the controller.
    If no controller is detected returns an error.

    :param:
    :return joystick:
    """
    pygame.init()
    pygame.joystick.init()

    joystick_count = pygame.joystick.get_count()

    if joystick_count > 0:
        for i in range(joystick_count):
            joystick = pygame.joystick.Joystick(i)
            joystick.init()

            name = joystick.get_name()
            print([i], name)

```

(continues on next page)

(continued from previous page)

```
            joystick.quit()

    else:
        sys.exit("Error: No joystick detected")

    selected_joystick = eval(input("Enter your joystick number:"))

    if selected_joystick not in range(joystick_count):
        sys.exit("Error: Your choice is not valid")

    joystick = pygame.joystick.Joystick(selected_joystick)
    joystick.init()

    return joystick


def mapping_button(joystick, dict_commands):
    """
    Associating a controller key with a command in dict_commands.

    :param joystick, dict_commands:
    :return mapping:
    """
    mapping = {}

    for command in dict_commands:
        print("Press the key", command)
        done = False
        while not done:
            for event in pygame.event.get():
                if event.type == pygame.JOYBUTTONDOWN:
                    if event.button not in (value for value in mapping.values()):
                        mapping[command] = event.button
                        done = True

    return mapping


def mapping_axis(joystick, axes=["pitch", "roll", "yaw", "vertical"]):
    """
    Associating the analog thumbsticks of the controller with a command in dict␣
↪commands

    :param joystick, dict_commands:
    :return mapping:
    """
    mapping = {}

    for i in axes:
        print("Push the", i, "axis")
        done = False
        while not done:
            for event in pygame.event.get():
                if event.type == pygame.JOYAXISMOTION:
                    if event.axis not in (value for value in mapping.values()):
                        mapping[i] = event.axis
                        done = True
```

(continues on next page)

(continued from previous page)

```
    return mapping


def _parse_button(dict_commands, button):
    """
    Send the commands to the drone.
    If multiple commands are assigned to a key each command will be sent one by one␣
↪to each press.

    :param dict_commands, button:
    :return:
    """
    commands = dict_commands[button][0]
    args = dict_commands[button][-1]

    command = commands[0]
    arg = args[0]

    if len(commands) == 1:
        if len(args) == 1:
            command(arg)

        else:
            command(arg)
            dict_commands[button][-1] = args[1:]+[arg]

    else:
        if len(commands) == 1:
            command(arg)
            dict_commands[button][0] = commands[1:]+[command]

        else:
            command(arg)
            dict_commands[button][0] = commands[1:]+[command]
            dict_commands[button][-1] = args[1:]+[arg]


def main_loop(joystick, dict_commands, mapping_button, mapping_axis):
    """
    First connects to the drone and makes a flat trim.
    Then in a loop read the events of the controller to send commands to the drone.

    :param joystick, dict_commands, mapping_button, mapping_axis:
    :return:
    """
    swing.connect(10)
    swing.flat_trim()

    while True:
        pygame.event.get()

        pitch = joystick.get_axis(mapping_axis["pitch"])*-100
        roll = joystick.get_axis(mapping_axis["roll"])*100
        yaw = joystick.get_axis(mapping_axis["yaw"])*100
        vertical = joystick.get_axis(mapping_axis["vertical"])*-100
```

(continues on next page)

(continued from previous page)

```
        swing.fly_direct(roll, pitch, yaw, vertical, 0.1)

        for button, value in mapping_button.items():
            if joystick.get_button(value):
                _parse_button(dict_commands, button)


if __name__ == "__main__":
    swing = Swing("e0:14:04:a7:3d:cb")

    #Example of dict_commands
    dict_commands = {
                        "takeoff_landing":[ #Name of the button
                                            [swing.safe_takeoff, swing.safe_land],
↪#Commands execute one by one
                                            [5]#Argument for executing the function
                                           ],
                        "fly_mode":[
                                    [swing.set_flying_mode],
                                    ["quadricopter", "plane_forward"]
                                   ],
                        "plane_gear_box_up":[
                                             [swing.set_plane_gear_box],
                                             [((swing.sensors.plane_gear_box[:-
↪1]+str(int(swing.sensors.plane_gear_box[-1])+1)) if swing.sensors.plane_gear_box[-
↪1] != "3" else "gear_3")]#"gear_1" => "gear_2" => "gear_3"
                                            ],
                        "plane_gear_box_down":[
                                               [swing.set_plane_gear_box],
                                               [((swing.sensors.plane_gear_box[:-
↪1]+str(int(swing.sensors.plane_gear_box[-1])-1)) if swing.sensors.plane_gear_box[-
↪1] != "1" else "gear_1")]#"gear_3" => "gear_2" => "gear_1"
                                            ]
                    }

    joystick = joystick_init()

    mapping_button = mapping_button(joystick, dict_commands)
    mapping_axis = mapping_axis(joystick)

    main_loop(joystick, dict_commands, mapping_button, mapping_axis)
```

1.3 Quick Start Guide with a Bebop

1.3.1 Using the pyparrot library on the Bebop

Before running any of the sample code, you will need to connect to your drone. To control the Bebop, you need to connect your controlling device (laptop, computer, etc) to the wifi for the drone. Look for the wifi network named Bebop_number where number varies for each drone.

1.3.2 Quick start: Demo Code

I have provided a set of example scripts for both the Mambo and the Bebop.

Demo of the trick commands on the bebop

The code shown below is the demoBebopTricks.py. demoBebopTricks.py will take off, demonstrate all 4 types of flips, and then land. It is a good program to verify that your connection to your bebop is working well. The bebop can flip just like the Mambo! This does the exact same thing as the Mambo tricks demo: take off, flip in all 4 directions, land. **Be sure to run it in a room large enough to perform the flips!**

```
"""
Demos the tricks on the bebop. Make sure you have enough room to perform them!

Author: Amy McGovern
"""

from pyparrot.Bebop import Bebop

bebop = Bebop()

print("connecting")
success = bebop.connect(10)
print(success)

print("sleeping")
bebop.smart_sleep(5)

bebop.ask_for_state_update()

bebop.safe_takeoff(10)

print("flip left")
print("flying state is %s" % bebop.sensors.flying_state)
success = bebop.flip(direction="left")
print("mambo flip result %s" % success)
bebop.smart_sleep(5)

print("flip right")
print("flying state is %s" % bebop.sensors.flying_state)
success = bebop.flip(direction="right")
print("mambo flip result %s" % success)
bebop.smart_sleep(5)

print("flip front")
print("flying state is %s" % bebop.sensors.flying_state)
success = bebop.flip(direction="front")
print("mambo flip result %s" % success)
bebop.smart_sleep(5)

print("flip back")
print("flying state is %s" % bebop.sensors.flying_state)
success = bebop.flip(direction="back")
print("mambo flip result %s" % success)
bebop.smart_sleep(5)

bebop.smart_sleep(5)
bebop.safe_land(10)

print("DONE - disconnecting")
bebop.disconnect()
```

Outdoor or large area demo of the direct flight commands on the bebop

The second example program shows how to directly fly the bebop by controlling the yaw, pitch, roll, and vertical movement parameters. **Make sure you try this one in a large enough room!** This code is provided in demoBebopDirectFlight.py and is also shown below.

```
"""
Flies the bebop in a fairly wide arc.  You want to be sure you have room for this.␣
↪(it is commented
out but even what is here is still going to require a large space)

Author: Amy McGovern
"""
from pyparrot.Bebop import Bebop

bebop = Bebop()

print("connecting")
success = bebop.connect(10)
print(success)

print("sleeping")
bebop.smart_sleep(5)

bebop.ask_for_state_update()

bebop.safe_takeoff(10)

print("Flying direct: going forward (positive pitch)")
bebop.fly_direct(roll=0, pitch=50, yaw=0, vertical_movement=0, duration=1)

print("Flying direct: yaw")
bebop.fly_direct(roll=0, pitch=0, yaw=50, vertical_movement=0, duration=1)

print("Flying direct: going backwards (negative pitch)")
bebop.fly_direct(roll=0, pitch=-50, yaw=0, vertical_movement=0, duration=0.5)

print("Flying direct: roll")
bebop.fly_direct(roll=50, pitch=0, yaw=0, vertical_movement=0, duration=1)

print("Flying direct: going up")
bebop.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=50, duration=1)

print("Turning relative")
bebop.move_relative(0, 0, 0, math.radians(90))

# this works but requires a larger test space than I currently have. Uncomment with␣
↪care and test only in large spaces!
#print("Flying direct: going around in a circle (yes you can mix roll, pitch, yaw in␣
↪one command!)")
#bebop.fly_direct(roll=25, pitch=0, yaw=50, vertical_movement=0, duration=5)

bebop.smart_sleep(5)
bebop.safe_land(10)

print("DONE - disconnecting")
bebop.disconnect()
```

Indoor demo of the direct flight commands on the bebop

If you couldn't run the outdoor or large space demo to test your bebop, this one is designed for a smaller space. It simply takes off, turns, and lands. **Make sure you are still flying in a safe place!** This code is provided in demoBebopIndoors.py and is also shown below.

```python
"""
Demo the Bebop indoors (sets small speeds and then flies just a small amount)
Note, the bebop will hurt your furniture if it hits it.  Even though this is a very␣
↪small
amount of flying, be sure you are doing this in an open area and are prepared to␣
↪catch!

Author: Amy McGovern
"""

from pyparrot.Bebop import Bebop

bebop = Bebop()

print("connecting")
success = bebop.connect(10)
print(success)

if (success):
    print("turning on the video")
    bebop.start_video_stream()

    print("sleeping")
    bebop.smart_sleep(2)

    bebop.ask_for_state_update()

    bebop.safe_takeoff(10)

    # set safe indoor parameters
    bebop.set_max_tilt(5)
    bebop.set_max_vertical_speed(1)

    # trying out the new hull protector parameters - set to 1 for a hull protection␣
↪and 0 without protection
    bebop.set_hull_protection(1)

    print("Flying direct: Slow move for indoors")
    bebop.fly_direct(roll=0, pitch=20, yaw=0, vertical_movement=0, duration=2)

    bebop.smart_sleep(5)

    bebop.safe_land(10)

    print("DONE - disconnecting")
    bebop.stop_video_stream()
    bebop.smart_sleep(5)
    bebop.disconnect()
```

1.4 Workshop Materials and Slides

We have taught several workshops on using pyparrot. As we continue to teach and develop materials for pyparrot, we will continue to share the curriculum materials. The materials are all stored in the coursework directory on at GitHub The files in ai_class include the project descriptions for the Spring 2018 AI Class taught by Dr. McGovern using the drones. The files in workshop_slides contain the slides from prior workshops including OKlahoma's 2018 and 2019 workshops.

1.5 Minidrone Commands and Sensors

1.5.1 Minidrone commands

Each of the public commands available to control the minidrone is listed below with its documentation. The code is also well documented and you can also look at the API through readthedocs. All of the functions preceeded with an underscore are intended to be internal functions and are not listed below.

Creating a Mambo object

`Mambo(address="", use_wifi=True/False)` create a mambo object with the specific harware address (found using findMinidrone). The use_wifi argument defaults to False (which means BLE is the default). Set to True to use wifi. You can only use wifi if you have a FPV camera installed on your Mambo! If you are using wifi, the hardware address argument can be ignored (it defaults to an empty string).

Creating a Swing object

`Swing(address="")` create a Swing object with the specific harware address (found using findMinidrone).

Connecting and disconnecting

`connect(num_retries)` connect to the Minidrone either using BLE services and characteristics or wifi (specified when you created the Mambo object). This can take several seconds to ensure the connection is working. You can specify a maximum number of re-tries. Returns true if the connection suceeded or False otherwise.

`disconnect()` disconnect from the BLE or wifi connection

Takeoff and landing

`safe_takeoff(timeout)` This is the recommended method for takeoff. It sends a command and then checks the sensors (via flying state) to ensure the minidrone is actually taking off. Then it waits until the minidrone is flying or hovering to return. It will timeout and return if the time exceeds timeout seconds.

`safe_land(timeout)` This is the recommended method to land the minidrone. Sends commands until the minidrone has actually reached the landed state. It will timeout and return if the time exceeds timeout seconds.

`takeoff()` Sends a single takeoff command to the minidrone. This is not the recommended method.

`land()` Sends a single land command to the minidrone. This is not the recommended method.

`turn_on_auto_takeoff()` This puts the minidrone in throw mode. When it is in throw mode, the eyes will blink.

Flying

`hover()` and `set_flat_trim()` both tell the drone to assume the current configuration is a flat trim and it will use this as the default when not receiving commands. This enables good hovering when not sending commands.

`flip(direction)` Sends the flip command to the minidrone. Valid directions to flip are: front, back, right, left.

`turn_degrees(degrees)` Turns the minidrone in place the specified number of degrees. The range is -180 to 180. This can be accomplished in direct_fly() as well but this one uses the internal minidrone sensors (which are not sent out right now) so it is more accurate.

`fly_direct(roll, pitch, yaw, vertical_movement, duration)` Fly the minidrone directly using the specified roll, pitch, yaw, and vertical movements. The commands are repeated for duration seconds. Note there are currently no sensors reported back to the user to ensure that these are working but hopefully that is addressed in a future firmware upgrade. Each value ranges from -100 to 100 and is essentially a percentage and direction of the max_tilt (for roll/pitch) or max_vertical_speed (for vertical movement).

`set_max_tilt(degrees)` Set the maximum tilt in degrees. Be careful as this makes your drone go slower or faster! It is important to note that the fly_direct command uses this value in conjunction with the -100 to 100 percentages.

`set_max_vertical_speed(speed)` Set the maximum vertical speed in m/s. Be careful as this makes your drone go up/down faster!

Pausing or sleeping in a thread safe manner

`smart_sleep(seconds)` This sleeps the number of seconds (which can be a floating point) but wakes for all BLE or wifi notifications. **Note, if you are using BLE: This comamnd is VERY important**. **NEVER** use regular time.sleep() as your BLE will disconnect regularly! Use smart_sleep instead! time.sleep() is ok if you are using wifi but smart_sleep() handles that for you.

USB accessories: Claw and Gun

`open_claw()` Open the claw. Note that the claw should be attached for this to work. The id is obtained from a prior `ask_for_state_update()` call. Note that you cannot use the claw with the FPV camera attached.

`close_claw()` Close the claw. Note that the claw should be attached for this to work. The id is obtained from a prior `ask_for_state_update()` call. Note that you cannot use the claw with the FPV camera attached.

`fire_gun()` Fires the gun. Note that the gun should be attached for this to work. The id is obtained from a prior `ask_for_state_update()` call. Note that you cannot use the gun with the FPV camera attached.

Swing specific commands

`set_plane_gear_box(state)` Choose the swing angle in plane mode. There are 3 tilt modes: gear_1, gear_2, gear_3. Warning gear_3 is very fast.

`set_flying_mode(mode)` Choose flight mode between: quadricopter, plane_forward, plane_backward.

Ground facing camera

`take_picture()`` The minidrone will take a picture with the downward facing camera. It only stores up to 40 pictures internally so this function deletes them after 35 have been taken. Make sure you are downloading them either using the mobile interface or through the python code.

Note: Parrot broke the ability to access the groundcam in their latest (3.0.25) firmware upgrade. We will reenable these functions as soon as parrot fixes the firmware but for now, they will only work in versions 3.0.24 and below.

`get_groundcam_pictures_names()` Returns the names of the pictures stored internally from the groundcam. Only for the mambo.

`get_groundcam_picture(name)` Returns the picture with the specified name. Only for the mambo.

Sensor related commands

`ask_for_state_update()` This sends a request to the minidrone to send back ALL states (this includes the claw and gun states). This really only needs to be called once at the start of the program to initialize some of the state variables. If you are on wifi, many of the other variables are sent at 2Hz. If you are on BLE, you will want to use this command to get more state information but keep in mind it will be slow. This command will return immediately but you should wait a few seconds before using the new state information as it has to be updated.

1.5.2 Mambo sensors

All of the sensor data that is passed back to the program is saved. Note that Parrot sends back more information via wifi than via BLE, due to the limited BLE bandwidth. The sensors are saved in Minidrone.sensors. This is an instance of a MamboSensors class, which can be seen at the top of the Minidrone.py file.

The easiest way to interact with the sensors is to call:

`minidrone.set_user_sensor_callback(function, args)`. This sets a user callback function with optional arguments that is called each time a sensor is updated. The refresh rate on wifi is 2Hz.

The sensors are:

- battery (defaults to 100 and stays at that level until a real reading is received from the drone)
- flying_state: This is updated as frequently as the drone sends it out and can be one of "landed", "takingoff", "hovering", "flying", "landing", "emergency", "rolling", "init". These are the values as specified in minidrone.xml.
- gun_id: defaults to 0 (as far as I can tell, it is only ever 0 when it comes from the drone anyway)
- gun_state: "READY" or "BUSY" as sent by the drone, if a gun is attached. Defaults to None.
- claw_id: defaults to 0
- claw_state: "OPENING", "OPENED", "CLOSING", "CLOSED" as sent by the drone, if a claw is attached. Defaults to None.
- speed_x, speed_y, speed_z, speed_ts: the speed in x (forward > 0), y (right > 0), and z (down > 0). The ts is the timestamp that the speed was valid.
- altitude, altitude_ts: wifi only, altitude in meters. Zero is where you took off. The ts is the timestamp where the altitude was valid.
- quaternion_w, quaternion_x, quaternion_y, quaternion_z, quaternion_ts: wifi only. Quaternion as estimated from takeoff (which is set to 0). Ranges from -1 to 1. ts is the timestamp where this was valid.
- `get_estimated_z_orientation()`: returns the estimated orientation using the unit quaternions. Note that 0 is the direction the drone is facing when you boot it up
- sensors_dict: all other sensors are saved by name in a dictionary. The names come from the minidrone.xml and common.xml.

1.6 Bebop Commands and Sensors

1.6.1 Bebop commands

Each of the public commands available to control the bebop is listed below with its documentation. The code is also well documented and you can also look at the API through readthedocs. All of the functions preceeded with an underscore are intended to be internal functions and are not listed below.

Creating a Bebop object

`Bebop(drone_type="Bebop2")` create a Bebop object with an optional drone_type argument that can be used to create a bebop one or bebop 2 object. Default is Bebop 2. Note, there is limited support for the original bebop since I do not own one for testing.

Connecting and disconnecting

`connect(num_retries)` connect to the Bebop's wifi services. This performs a handshake. This can take several seconds to ensure the connection is working. You can specify a maximum number of re-tries. Returns true if the connection suceeded or False otherwise.

`disconnect()` disconnect from the wifi connection

Takeoff and landing

`takeoff()` Sends a single takeoff command to the bebop. This is not the recommended method.

`safe_takeoff(timeout)` This is the recommended method for takeoff. It sends a command and then checks the sensors (via flying state) to ensure the bebop is actually taking off. Then it waits until the bebop is flying or hovering to return. It will timeout and return if the time exceeds timeout seconds.

`land()` Sends a single land command to the bebop. This is not the recommended method.

`safe_land(timeout)` This is the recommended method to land the bebop. Sends commands until the bebop has actually reached the landed state. It will timeout and return if the time exceeds timeout seconds.

Flying

`flip(direction)` Sends the flip command to the bebop. Valid directions to flip are: front, back, right, left.

`fly_direct(roll, pitch, yaw, vertical_movement, duration)` Fly the bebop directly using the specified roll, pitch, yaw, and vertical movements. The commands are repeated for duration seconds. Note there are currently no sensors reported back to the user to ensure that these are working but hopefully that is addressed in a future firmware upgrade. Each value ranges from -100 to 100 and is essentially a percentage and direction of the max_tilt (for roll/pitch) or max_vertical_speed (for vertical movement).

`move_relative(dx, dy, dz, dradians)` Moves the bebop a relative number of meters in x (forward/backward, forward is positive), y (right/left, right is positive), dz (up/down, positive is down), and dradians. If you use this command INDOORS, make sure you either have FULL GPS coverage or NO GPS coverage (e.g. cover the front of the bebop

> with tin foil to keep it from getting a lock). If it has mixed coverage, it randomly flies at high speed in random

directions after the command executes. This is a known issue in Parrot's firmware and they state that a fix is coming.

`set_max_altitude(altitude)` Set the maximum allowable altitude in meters. The altitude must be between 0.5 and 150 meters.

`set_max_distance(distance)` Set max distance between the takeoff and the drone in meters. The distance must be between 10 and 2000 meters.

`enable_geofence(value)` If geofence is enabled, the drone won't fly over the given max distance. Valid value: 1 if the drone can't fly further than max distance, 0 if no limitation on the drone should be done.

`set_max_tilt(tilt)` Set the maximum allowable tilt in degrees for the drone (this limits speed). The tilt must be between 5 (very slow) and 30 (very fast) degrees.

`set_max_tilt_rotation_speed(speed)` Set the maximum allowable tilt rotation speed in degree/s. The tilt rotation speed must be between 80 and 300 degree/s.

`set_max_vertical_speed(speed)` Set the maximum allowable vertical speed in m/s. The vertical speed must be between 0.5 and 2.5 m/s.

`set_max_rotation_speed(speed)` Set the maximum allowable rotation speed in degree/s. The rotation speed must be between 10 and 200 degree/s.

`set_flat_trim(duration=0)` Tell the Bebop to run with a flat trim. If duration > 0, waits for the comand to be acknowledged

`set_hull_protection(present)` Set the presence of hull protection (only for bebop 1). The value must be 1 if hull protection is present or 0 if not present. This is only useful for the bebop 1.

`set_indoor(is_outdoor)` Set the bebop 1 (ignored on bebop 2) to indoor or outdoor mode. The value must be 1 if bebop 1 is outdoors or 0 if it is indoors. This is only useful for the bebop 1.

Pausing or sleeping in a thread safe manner

`smart_sleep(seconds)` This sleeps the number of seconds (which can be a floating point) but wakes for all wifi notifications. You should use this instead of time.sleep to be consistent with the mambo but it is not required (whereas time.sleep() will break a mambo using BLE).

Video camera

`start_video_stream()`: tells the bebop to start streaming the video. These are really intended to be called within the DroneVision or DroneVisionGUI functions and not directly by the user (but you can call them directly if you are writing your own vision routines).

`stop_video_stream()`: tells the bebop to stop streaming the video. Same as above: intended to be called by the DroneVision or DroneVisionGUI routines.

`set_video_stream_mode(mode)`: set the video mode to one of three choices: "low_latency", "high_reliability", "high_reliability_low_framerate". low_latency is the default.

`pan_tilt_camera(tilt_degrees, pan_degrees)`: Send the command to pan/tilt the camera by the specified number of degrees in pan/tilt. Note, this only seems to work in small increments. Use pan_tilt_velocity to get the camera to look straight downward.

`pan_tilt_camera_velocity(self, tilt_velocity, pan_velocity, duration=0)`: Send the command to tilt the camera by the specified number of degrees per second in pan/tilt. This function has two modes. First, if duration is 0, the initial velocity is sent and then the function returns (meaning the camera will keep moving). If duration is greater than 0, the command executes for that amount of time and then sends a stop command to the camera and then returns.

`set_picture_format(format)`: Change the picture format to raw, jpeg, snapshot or jpeg_fisheye.

`set_white_balance(type)`: Change the type of white balance between: auto, tungsten, daylight, cloudy or cool_white.

`set_exposition(value)`: Change the image exposition between -1.5 and 1.5.

`set_saturation(value)`: Change the image saturation between -100 and 100.

`set_timelapse(enable, interval)`: To start a timelapse set enable at 1 and an interval between 8 and 300 sec. To stop the timelapse just set enable to 0.

`set_video_stabilization(mode)`: Change the video stabilization between 4 modes: roll_pitch, pitch, roll, none.

`set_video_recording(mode)`: Change the video recording mode between quality and time.

`set_video_framerate(framerate)`: Change the video framerate between: 24_FPS, 25_FPS or 30_FPS.

`set_video_resolutions(type)`: Change the video resolutions for stream and rec between rec1080_stream480, rec720_stream720.

Sensor commands

`ask_for_state_update()` This sends a request to the bebop to send back ALL states. The data returns fairly quickly although not instantly. The bebop already has a sensor refresh rate of 10Hz but not all sensors are sent automatically. If you are looking for a specific sensor that is not automatically sent, you can call this but I don't recommend sending it over and over. Most of the sensors you need should be sent at either the 10Hz rate or as an event is called that triggers that sensor.

1.6.2 Bebop sensors

All of the sensor data that is passed back to the Bebop is saved in a python dictionary. As needed, other variables are stored outside the dictionary but you can get everything you need from the dictionary itself. All of the data is stored in the BebopSensors class.

The easiest way to interact with the sensors is to call:

`bebop.set_user_sensor_callback(function, args)`. This sets a user callback function with optional arguments that is called each time a sensor is updated. The refresh rate on wifi is 10Hz.

The sensors are:

- battery (defaults to 100 and stays at that level until a real reading is received from the drone)
- flying_state: This is updated as frequently as the drone sends it out and can be one of "landed", "takingoff", "hovering", "flying", "landing", "emergency", "usertakeoff", "motor_ramping", "emergency_landing". These are the values as specified in ardrone3.xml.
- sensors_dict: all other sensors are saved by name in a dictionary. The names come from the ardrone3.xml and common.xml.

1.7 Using Vision on the Mambos and Bebop

The vision system uses a common interface for the Mambo and the Bebop. There are two approaches to the vision. The first relies on ffmpeg and the second on libVLC. Both approaches assume opencv is also installed. Note, the reason we do not simply rely on opencv directly is that there is a known existing bug in opencv with RTSP streams that makes them unreliable. The behavior we reliably saw by using opencv directly was that the stream would open, obtain between 10 and 15 frames and then stop collecting any new frames. Since this is not tenable for flying, we bypass opencv's direct open of the stream with two other approaches described below.

1.7.1 Using ffmpeg for vision

ffmpeg is an open-source cross-platform library that can be used to process a wide variety of video and audio streams. We use ffmpeg to bypass the issue with opencv by opening the video stream in ffmpeg. The advantage of this approach is that the ffmpeg encoder can read the raw video streams (RTSP for Mambo and RTP for Bebop) and convert it directly to a format that opencv can easily read. We chose to convert to png format. The disadvantage is that ffmpeg has to save the converted images to disk, which introduces a slight delay to the image processing. If you use a high-end laptop with a solid state drive, this delay can be as low as 0.5 seconds. Lower-processing speed laptops can introduce longer delays. However, if you want access to the images after your flight or you do not need real-time access, this is the better choice.

The vision code itself can be found in DroneVision.py. Running this code requires both the ffmpeg software and the opencv package. This approach does NOT show a live stream of the vision to the user but you can visualize the images using the VisionServer.py.

One thing to note: since ffmpeg requires the images to be written to a folder, it will save images to a directory named images inside the pyparrot package. **You will want to clean this folder out after each flight!**

Mambo ffmpeg vision demo

The demo code for the mambo, demoMamboVision.py shown below, has the mambo take off, move upwards for a short time, flip, and then land. While all of this flight code is running, the vision processing is running in a separate thread.

The first highlighted line:

```
testFlying = False
```

can be changed to True to have the mambo fly or set to False to do the vision without the mambo moving. **One big note: if the mambo is NOT flying, it will turn the camera into a sleep mode after several minutes and you either need to reboot the mambo or connect and takeoff to restart the camera.**

The highlighted line with the code:

```
mamboVision.set_user_callback_function(userVision.save_pictures, user_callback_
↪args=None)
```

is where the user sets the function to be called with every new vision frame. In this demo, the images are simply read in and saved back to a new file name.

```
"""
Demo of the ffmpeg based mambo vision code (basically flies around and saves out␣
↪photos as it flies)

Author: Amy McGovern
"""
from pyparrot.Mambo import Mambo
from pyparrot.DroneVision import DroneVision
import threading
import cv2
import time
```

(continues on next page)

(continued from previous page)

```
# set this to true if you want to fly for the demo
testFlying = False

class UserVision:
    def __init__(self, vision):
        self.index = 0
        self.vision = vision

    def save_pictures(self, args):
        print("in save pictures on image %d " % self.index)

        img = self.vision.get_latest_valid_picture()

        if (img is not None):
            filename = "test_image_%06d.png" % self.index
            cv2.imwrite(filename, img)
            self.index +=1
            #print(self.index)



# you will need to change this to the address of YOUR mambo
mamboAddr = "e0:14:d0:63:3d:d0"

# make my mambo object
# remember to set True/False for the wifi depending on if you are using the wifi or␣
↪the BLE to connect
mambo = Mambo(mamboAddr, use_wifi=True)
print("trying to connect to mambo now")
success = mambo.connect(num_retries=3)
print("connected: %s" % success)

if (success):
    # get the state information
    print("sleeping")
    mambo.smart_sleep(1)
    mambo.ask_for_state_update()
    mambo.smart_sleep(1)

    print("Preparing to open vision")
    mamboVision = DroneVision(mambo, is_bebop=False, buffer_size=30)
    userVision = UserVision(mamboVision)
    mamboVision.set_user_callback_function(userVision.save_pictures, user_callback_
↪args=None)
    success = mamboVision.open_video()
    print("Success in opening vision is %s" % success)

    if (success):
        print("Vision successfully started!")
        #removed the user call to this function (it now happens in open_video())
        #mamboVision.start_video_buffering()

        if (testFlying):
            print("taking off!")
            mambo.safe_takeoff(5)
```

(continues on next page)

(continued from previous page)

```
            if (mambo.sensors.flying_state != "emergency"):
                print("flying state is %s" % mambo.sensors.flying_state)
                print("Flying direct: going up")
                mambo.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=20,␣
↪duration=1)

                print("flip left")
                print("flying state is %s" % mambo.sensors.flying_state)
                success = mambo.flip(direction="left")
                print("mambo flip result %s" % success)
                mambo.smart_sleep(5)

            print("landing")
            print("flying state is %s" % mambo.sensors.flying_state)
            mambo.safe_land(5)
        else:
            print("Sleeeping for 15 seconds - move the mambo around")
            mambo.smart_sleep(15)

        # done doing vision demo
        print("Ending the sleep and vision")
        mamboVision.close_video()

        mambo.smart_sleep(5)

    print("disconnecting")
    mambo.disconnect()
```

Bebop ffmpeg vision demo

The demo code for the bebop for the ffmpeg vision works nearly identically to the mambo demo except it does not fly the bebop around. Instead, it starts the camera and then sleeps for 30 seconds for the user to move around or move the drone around. This is intended as a safe demo for indoors. It also moves the camera around so that it is obvious the vision is recording different photos. The code is available at demoBebopVision.py and is shown below. The highlighted line is again where the user sets the callback function of how to process the vision frames.

Updated in Version 1.5.13: you can tell DroneVision to either remove all the old vision files (now the default) or not by sending the parameter cleanup_old_images=True or False.

```
"""
Demo of the Bebop ffmpeg based vision code (basically flies around and saves out␣
↪photos as it flies)

Author: Amy McGovern
"""
from pyparrot.Bebop import Bebop
from pyparrot.DroneVision import DroneVision
import threading
import cv2
import time

isAlive = False

class UserVision:
    def __init__(self, vision):
```

(continues on next page)

(continued from previous page)

```
        self.index = 0
        self.vision = vision

    def save_pictures(self, args):
        #print("saving picture")
        img = self.vision.get_latest_valid_picture()

        if (img is not None):
            filename = "test_image_%06d.png" % self.index
            #cv2.imwrite(filename, img)
            self.index +=1


# make my bebop object
bebop = Bebop()

# connect to the bebop
success = bebop.connect(5)

if (success):
    # start up the video
    bebopVision = DroneVision(bebop, is_bebop=True)

    userVision = UserVision(bebopVision)
    bebopVision.set_user_callback_function(userVision.save_pictures, user_callback_
↪args=None)
    success = bebopVision.open_video()

    if (success):
        print("Vision successfully started!")
        #removed the user call to this function (it now happens in open_video())
        #bebopVision.start_video_buffering()

        # skipping actually flying for safety purposes indoors - if you want
        # different pictures, move the bebop around by hand
        print("Fly me around by hand!")
        bebop.smart_sleep(5)

        print("Moving the camera using velocity")
        bebop.pan_tilt_camera_velocity(pan_velocity=0, tilt_velocity=-2, duration=4)
        bebop.smart_sleep(25)
        print("Finishing demo and stopping vision")
        bebopVision.close_video()

    # disconnect nicely so we don't need a reboot
    bebop.disconnect()
else:
    print("Error connecting to bebop.  Retry")
```

1.7.2 Using libVLC for vision

Our second approach to vision relies on the libVLC library, which in turn relies on the VLC program. VLC is a cross-platform media player and libVLC is a python interface to the VLC libraries. This can be done entirely in memory (not writing out to disk as ffmpeg required), which means that the delay is minimized. If you have a need for the full image stream after your flight, you likely should choose the ffmpeg approach. If you simply want to use the vision, this approach may work better for you since you don't have the disk delay and you don't introduce the issues with the images subdirectory. The other advantage of this approach is that you get a real-time video stream of what the drone is seeing. However, controlling the drone after vision has started requires setting a new parameter called user_code_to_run, as shown below and in the highlighted line in the demo code.

```
mamboVision = DroneVisionGUI(mambo, is_bebop=False, buffer_size=200,
                             user_code_to_run=demo_mambo_user_vision_function, user_
↪args=(mambo, ))
```

To make this approach work, you MUST install the VLC client version 3.0.1 or greater. Installing only libvlc is not needed (the source is included with pyparrot) and it will not work. Installing the client installs extra software that the libvlc python library requires.

There are two example programs, one for the bebop and one for the mambo. Both show the window opened by this approach and the way that a user can run their own code by assigning code to the Run button.

libVLC demo code for the Mambo

This code can be downloaded from demoMamboVisionGUI.py and is repeated below.

```
"""
Demo of the Bebop vision using DroneVisionGUI that relies on libVLC.  It is a␣
↪different
multi-threaded approach than DroneVision

Author: Amy McGovern
"""
from pyparrot.Mambo import Mambo
from pyparrot.DroneVisionGUI import DroneVisionGUI
import cv2


# set this to true if you want to fly for the demo
testFlying = True

class UserVision:
    def __init__(self, vision):
        self.index = 0
        self.vision = vision

    def save_pictures(self, args):
        # print("in save pictures on image %d " % self.index)

        img = self.vision.get_latest_valid_picture()

        if (img is not None):
            filename = "test_image_%06d.png" % self.index
            # uncomment this if you want to write out images every time you get a new␣
↪one
            #cv2.imwrite(filename, img)
            self.index +=1
            #print(self.index)


def demo_mambo_user_vision_function(mamboVision, args):
    """
    Demo the user code to run with the run button for a mambo
```

(continues on next page)

(continued from previous page)

```
    :param args:
    :return:
    """
    mambo = args[0]

    if (testFlying):
        print("taking off!")
        mambo.safe_takeoff(5)

        if (mambo.sensors.flying_state != "emergency"):
            print("flying state is %s" % mambo.sensors.flying_state)
            print("Flying direct: going up")
            mambo.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=15, duration=2)

            print("flip left")
            print("flying state is %s" % mambo.sensors.flying_state)
            success = mambo.flip(direction="left")
            print("mambo flip result %s" % success)
            mambo.smart_sleep(5)

        print("landing")
        print("flying state is %s" % mambo.sensors.flying_state)
        mambo.safe_land(5)
    else:
        print("Sleeeping for 15 seconds - move the mambo around")
        mambo.smart_sleep(15)

    # done doing vision demo
    print("Ending the sleep and vision")
    mamboVision.close_video()

    mambo.smart_sleep(5)

    print("disconnecting")
    mambo.disconnect()


if __name__ == "__main__":
    # you will need to change this to the address of YOUR mambo
    mamboAddr = "e0:14:d0:63:3d:d0"

    # make my mambo object
    # remember to set True/False for the wifi depending on if you are using the wifi␣
↪or the BLE to connect
    mambo = Mambo(mamboAddr, use_wifi=True)
    print("trying to connect to mambo now")
    success = mambo.connect(num_retries=3)
    print("connected: %s" % success)

    if (success):
        # get the state information
        print("sleeping")
        mambo.smart_sleep(1)
        mambo.ask_for_state_update()
        mambo.smart_sleep(1)
```

(continues on next page)

(continued from previous page)

```
        print("Preparing to open vision")
        mamboVision = DroneVisionGUI(mambo, is_bebop=False, buffer_size=200,
                                     user_code_to_run=demo_mambo_user_vision_function,
↪ user_args=(mambo, ))
        userVision = UserVision(mamboVision)
        mamboVision.set_user_callback_function(userVision.save_pictures, user_
↪callback_args=None)
        mamboVision.open_video()
```

libVLC demo code for the Bebop

This code can be downloaded from demoBebopVision.py and is repeated below. Note that in version 1.5.18 we added the ability for the user to draw a second window. This is optional and is shown in the code below.

```
"""
"""
    Demo of the Bebop vision using DroneVisionGUI (relies on libVLC).  It is a␣
↪different
    multi-threaded approach than DroneVision

    Author: Amy McGovern
    """
    from pyparrot.Bebop import Bebop
    from pyparrot.DroneVisionGUI import DroneVisionGUI
    import threading
    import cv2
    import time
    from PyQt5.QtGui import QImage

    isAlive = False

    class UserVision:
        def __init__(self, vision):
            self.index = 0
            self.vision = vision

        def save_pictures(self, args):
            #print("saving picture")
            img = self.vision.get_latest_valid_picture()

            # limiting the pictures to the first 10 just to limit the demo from␣
↪writing out a ton of files
            if (img is not None and self.index <= 10):
                filename = "test_image_%06d.png" % self.index
                cv2.imwrite(filename, img)
                self.index +=1


    def draw_current_photo():
        """
        Quick demo of returning an image to show in the user window.  Clearly one␣
↪would want to make this a dynamic image
        """
        image = cv2.imread('test_image_000001.png')
```

(continues on next page)

(continued from previous page)

```
        if (image is not None):
            if len(image.shape) < 3 or image.shape[2] == 1:
                image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
            else:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            height, width, byteValue = image.shape
            byteValue = byteValue * width

            qimage = QImage(image, width, height, byteValue, QImage.Format_RGB888)

            return qimage
        else:
            return None

    def demo_user_code_after_vision_opened(bebopVision, args):
        bebop = args[0]

        print("Vision successfully started!")
        #removed the user call to this function (it now happens in open_video())
        #bebopVision.start_video_buffering()

        # takeoff
        # bebop.safe_takeoff(5)

        # skipping actually flying for safety purposes indoors - if you want
        # different pictures, move the bebop around by hand
        print("Fly me around by hand!")
        bebop.smart_sleep(15)

        if (bebopVision.vision_running):
            print("Moving the camera using velocity")
            bebop.pan_tilt_camera_velocity(pan_velocity=0, tilt_velocity=-2,␣
↪duration=4)
            bebop.smart_sleep(5)

            # land
            bebop.safe_land(5)

            print("Finishing demo and stopping vision")
            bebopVision.close_video()

        # disconnect nicely so we don't need a reboot
        print("disconnecting")
        bebop.disconnect()

    if __name__ == "__main__":
        # make my bebop object
        bebop = Bebop()

        # connect to the bebop
        success = bebop.connect(5)

        if (success):
            # start up the video
            bebopVision = DroneVisionGUI(bebop, is_bebop=True, user_code_to_run=demo_
↪user_code_after_vision_opened,
```

(continues on next page)

(continued from previous page)

```
                                              user_args=(bebop, ), user_draw_window_
↪fn=draw_current_photo)

            userVision = UserVision(bebopVision)
            bebopVision.set_user_callback_function(userVision.save_pictures, user_
↪callback_args=None)
            bebopVision.open_video()

        else:
            print("Error connecting to bebop.  Retry")
```

1.8 Frequently Asked Questions

Below is a list of common errors and how to fix them.

1.8.1 Vision isn't showing anything on the minidrone

The Minidrone camera puts itself into a "resting" state after not flying for several minutes. To solve this, you either need to fly again (a simple takeoff and landing will suffice) or reboot the minidrone and reconnect.

1.8.2 I'm using windows and my drone gives me lots of timeout errors

This is a windows security setting and it can be fixed. Go into your windows firewall settings (control panel, system and security, allow a program through Windows Firewall) and change the settings for python.exe to be allowed through the firewall for both home/private networks and public networks. Your sensors will suddenly be able to send data to your machine and safe_land will start working again as well as any sensors!

1.8.3 My drone does takeoff and landing but nothing else

Likely you have the remote controller on and attached! For some reason, if the remote is on, it will allow the python code to takeoff & land but no other commands will work. Turn off the remote and your code should work fine!

1.8.4 Errors connecting to the drone

There are two common errors that I see when flying. One requires the drone to reboot and one requires the computer controlling the drone to reboot.

Connection failed

If you fail to connect to the drone, you will see an error message like this:

```
connection failed: did you remember to connect your machine to the Drone's wifi␣
↪network?
```

The most likely cause is that you forgot to connect to the drone's wifi. If you tried to connect, sometimes that connection fails. Try again or let the connection sit for a minute and try your program again.

If you are on the wifi but you get connection refused errors, reboot the drone.

Address in use

The second common error is about the address being in use, as shown below.

```
OSError: [Errno 48] Address already in use
```

There are two ways to fix this, depending on the issue. It is possible you tried to run a second program while you still had a first program running. If this is the case, make sure you stop all of your minidrone programs and then restart only one. If you are not running a second minidrone program, then the solution is to reboot. This sometimes happens due to the program crashing before it releases the socket.

1.9 Contact the pyparrot developers

1.9.1 Contribute

We welcome your contributions via bug report or pull request.

- Issue Tracker: https://github.com/amymcgovern/pyparrot/issues
- Pull requests: https://github.com/amymcgovern/pyparrot/pulls
- Source Code: https://github.com/amymcgovern/pyparrot

1.9.2 Support

If you are having issues, please let us know by reporting issues on GitHub using the issue tracker https://github.com/amymcgovern/pyparrot/issues.

1.10 About the pyparrot project

1.10.1 About pyparrot

Pyparrot was developed by Dr. Amy McGovern with the support of the University of Oklahoma and the Kiss Institute for Practical Robotics. The original goal was to teach children to program using educational programs like botball. The pyparrot project has been adopted by groups around the world and has been used in both K-12 settings and at the university level.

1.10.2 Educational Programs Using pyparrot

If you would like to be added to this list, please email dramymcgovern @ gmail.com (without the spaces).

- Botball
- University of Oklahoma School of Computer Science
- Talenteahaus
- Tech Garage
- St Eugene College
- KIPR/botball

1.11 License

1.11.1 MIT License

The project is licensed under the MIT License.

1.12 pyparrot

1.12.1 pyparrot package

Subpackages

pyparrot.commandsandsensors package

Submodules

pyparrot.commandsandsensors.DroneCommandParser module

class `pyparrot.commandsandsensors.DroneCommandParser.`**`DroneCommandParser`**

Bases: `object`

`get_command_tuple`(*project*, *myclass*, *cmd*)

Parses the command XML for the specified class name and command name

Parameters

- **`myclass`** – class name (renamed to myclass to avoid reserved name) in the xml file
- **`cmd`** – command to execute (from XML file)

Returns

`get_command_tuple_with_enum`(*project*, *myclass*, *cmd*, *enum_name*)

Parses the command XML for the specified class name and command name and checks for enum_name

Parameters

- **`myclass`** – class name (renamed to myclass to avoid reserved name) in the xml file

- **cmd** – command to execute (from XML file)

Returns

pyparrot.commandsandsensors.DroneSensorParser module

Sensor parser class: handles the XML parsing and gets the values but the actual data is stored with the drone itself since it knows what to do with it.

class pyparrot.commandsandsensors.DroneSensorParser.**DroneSensorParser**(*drone_type*)
Bases: object

extract_sensor_values(*data*)
Extract the sensor values from the data in the BLE packet :param data: BLE packet of sensor data :return: a list of tuples of (sensor name, sensor value, sensor enum, header_tuple)

pyparrot.commandsandsensors.DroneSensorParser.**get_data_format_and_size**(*data*, *data_type*)
Internal function to convert data_type to the corresponding struct.pack format string as per https://docs.python.org/2/library/struct.html#format-characters

Function contributed by awm102 on GitHub. Amy moved this to DroneSensorParser to be more general, edited a bit to fit within the drone sensor parser as well.

Parameters

- **data** – the data that will be packed. Not actually used here unless the data_type is string, then it is used to calculate the data size.
- **data_type** – a string representing the data type

Returns a tuple of a string representing the struct.pack format for the data type and an int representing the number of bytes

Module contents

pyparrot.networking package

Submodules

pyparrot.networking.bleConnection module

class pyparrot.networking.bleConnection.**BLEConnection**(*address*, *minidrone*)
Bases: object

ack_packet(*buffer_id*, *packet_id*)
Ack the packet id specified by the argument on the ACK_COMMAND channel

Parameters **packet_id** – the packet id to ack

Returns nothing

connect(*num_retries*)
Connects to the drone and re-tries in case of failure the specified number of times

Param num_retries is the number of times to retry

Returns True if it succeeds and False otherwise

disconnect()
Disconnect the BLE connection. Always call this at the end of your programs to cleanly disconnect.

Returns void

send_auto_takeoff_command(*command_tuple*)
Build the packet for auto takeoff and send it

Parameters **command_tuple** – command tuple from the parser

Returns True if the command was sent and False otherwise

send_command_packet_ack(*packet*)
Sends the actual packet on the ack channel. Internal function only.

Parameters **packet** – packet constructed according to the command rules (variable size, constructed elsewhere)

Returns True if the command was sent and False otherwise

send_enum_command_packet_ack(*command_tuple*, *enum_value*, *usb_id=None*)
Send a command on the ack channel with enum parameters as well (most likely a flip). All commandsandsensors except PCMD go on the ack channel per http://forum.developer.parrot.com/t/ble-characteristics-of-minidrones/5912/2

the id of the last command sent (for use in ack) is the send counter (which is incremented before sending)

Parameters

- **command_tuple** – 3 tuple of the command bytes. 0 padded for 4th byte
- **enum_value** – the enum index

Returns nothing

send_noparam_command_packet_ack(*command_tuple*)
Send a command on the ack channel - where all commandsandsensors except PCMD go, per http://forum.developer.parrot.com/t/ble-characteristics-of-minidrones/5912/2

the id of the last command sent (for use in ack) is the send counter (which is incremented before sending)

Ensures the packet was received or sends it again up to a maximum number of times.

Parameters **command_tuple** – 3 tuple of the command bytes. 0 padded for 4th byte

Returns True if the command was sent and False otherwise

send_param_command_packet(*command_tuple*, *param_tuple=None*, *param_type_tuple=0*, *ack=True*)
Send a command packet with parameters. Ack channel is optional for future flexibility, but currently commands are always send over the Ack channel so it defaults to True.

Contributed by awm102 on github. Edited by Amy McGovern to work for BLE commands also.

Param command_tuple: the command tuple derived from command_parser.get_command_tuple()

Param param_tuple (optional): the parameter values to be sent (can be found in the XML files)

Param param_size_tuple (optional): a tuple of strings representing the data type of the parameters

e.g. u8, float etc. (can be found in the XML files) :param: ack (optional): allows ack to be turned off if required :return:

send_pcmd_command(*command_tuple*, *roll*, *pitch*, *yaw*, *vertical_movement*, *duration*)
Send the PCMD command with the specified roll, pitch, and yaw

Parameters

- **`command_tuple`** – command tuple per the parser
- **`roll`** –
- **`pitch`** –
- **`yaw`** –
- **`vertical_movement`** –
- **`duration`** –

`send_single_pcmd_command`(*command_tuple*, *roll*, *pitch*, *yaw*, *vertical_movement*)

Send a single PCMD command with the specified roll, pitch, and yaw. Note this will not make that command run forever. Instead it sends ONCE. This can be used in a loop (in your agent) that makes more smooth control than using the duration option.

Parameters

- **`command_tuple`** – command tuple per the parser
- **`roll`** –
- **`pitch`** –
- **`yaw`** –
- **`vertical_movement`** –

`send_turn_command`(*command_tuple*, *degrees*)

Build the packet for turning and send it

Parameters

- **`command_tuple`** – command tuple from the parser
- **`degrees`** – how many degrees to turn

Returns True if the command was sent and False otherwise

`smart_sleep`(*timeout*)

Sleeps the requested number of seconds but wakes up for notifications

Note: NEVER use regular time.sleep! It is a blocking sleep and it will likely cause the BLE to disconnect due to dropped notifications. Always use smart_sleep instead!

Parameters **`timeout`** – number of seconds to sleep

Returns

pyparrot.networking.wifiConnection module

Holds all the data and commands needed to fly a Bebop or Anafi drone.

Author: Amy McGovern, dramymcgovern@gmail.com

class `pyparrot.networking.wifiConnection.`**`WifiConnection`**(*drone*, *drone_type='Bebop2'*, *ip_address=None*)

Bases: `object`

`ack_packet`(*buffer_id*, *packet_id*)

Ack the packet id specified by the argument on the ACK_COMMAND channel

Parameters **`packet_id`** – the packet id to ack

Returns nothing

`connect`(*num_retries*)
Connects to the drone

Parameters **`num_retries`** – maximum number of retries

Returns True if the connection succeeded and False otherwise

`disconnect`()
Disconnect cleanly from the sockets

`handle_data`(*data*)
Handles the data as it comes in

Parameters **`data`** – raw data packet

Returns

`handle_frame`(*packet_type*, *buffer_id*, *packet_seq_id*, *recv_data*)

`safe_send`(*packet*)

`send_camera_move_command`(*command_tuple*, *pan*, *tilt*)
Send the packet to move the camera (this is Bebop only).

Parameters

- **`command_tuple`** – command tuple per the parser
- **`pan`** –
- **`tilt`** –

`send_command_packet_ack`(*packet*, *seq_id*)
Sends the actual packet on the ack channel. Internal function only.

Parameters **`packet`** – packet constructed according to the command rules (variable size, constructed elsewhere)

Returns True if the command was sent and False otherwise

`send_command_packet_noack`(*packet*)
Sends the actual packet on the No-ack channel. Internal function only.

Parameters **`packet`** – packet constructed according to the command rules (variable size, constructed elsewhere)

Returns True if the command was sent and False otherwise

`send_enum_command_packet_ack`(*command_tuple*, *enum_value*, *usb_id=None*)
Send a command on the ack channel with enum parameters as well (most likely a flip). All commandsandsensors except PCMD go on the ack channel per http://forum.developer.parrot.com/t/ble-characteristics-of-minidrones/5912/2

the id of the last command sent (for use in ack) is the send counter (which is incremented before sending)

Parameters

- **`command_tuple`** – 3 tuple of the command bytes. 0 padded for 4th byte
- **`enum_value`** – the enum index

Returns nothing

send_fly_relative_command(*command_tuple*, *change_x*, *change_y*, *change_z*, *change_angle*)
Send the packet to fly relative (this is Bebop only).

Parameters

- **command_tuple** – command tuple per the parser
- **change_x** – change in x
- **change_y** – change in y
- **change_z** – change in z
- **change_angle** – change in angle

send_noparam_command_packet_ack(*command_tuple*)
Send a no parameter command packet on the ack channel :param command_tuple: :return:

send_noparam_high_priority_command_packet(*command_tuple*)
Send a no parameter command packet on the high priority channel :param command_tuple: :return:

send_param_command_packet(*command_tuple*, *param_tuple=None*, *param_type_tuple=0*, *ack=True*)
Send a command packet with parameters. Ack channel is optional for future flexibility, but currently commands are always send over the Ack channel so it defaults to True.

Contributed by awm102 on github

Param command_tuple: the command tuple derived from command_parser.get_command_tuple()

Param param_tuple (optional): the parameter values to be sent (can be found in the XML files)

Param param_size_tuple (optional): a tuple of strings representing the data type of the parameters

e.g. u8, float etc. (can be found in the XML files) :param: ack (optional): allows ack to be turned off if required :return:

send_pcmd_command(*command_tuple*, *roll*, *pitch*, *yaw*, *vertical_movement*, *duration*)
Send the PCMD command with the specified roll, pitch, and yaw

Parameters

- **command_tuple** – command tuple per the parser
- **roll** –
- **pitch** –
- **yaw** –
- **vertical_movement** –
- **duration** –

send_single_pcmd_command(*command_tuple*, *roll*, *pitch*, *yaw*, *vertical_movement*)
Send a single PCMD command with the specified roll, pitch, and yaw. Note this will not make that command run forever. Instead it sends ONCE. This can be used in a loop (in your agent) that makes more smooth control than using the duration option.

Parameters

- **command_tuple** – command tuple per the parser
- **roll** –
- **pitch** –

- **`yaw`** –
- **`vertical_movement`** –

`send_turn_command`(*command_tuple*, *degrees*)

Build the packet for turning and send it

Parameters

- **`command_tuple`** – command tuple from the parser
- **`degrees`** – how many degrees to turn

Returns True if the command was sent and False otherwise

`smart_sleep`(*timeout*)

Sleeps the requested number of seconds but wakes up for notifications

Note: time.sleep misbehaves for the BLE connections but seems ok for wifi. I encourage you to use smart_sleep since it handles the sleeping in a thread-safe way.

Parameters **`timeout`** – number of seconds to sleep

Returns

class `pyparrot.networking.wifiConnection.`**`mDNSListener`**(*wifi_connection*)

Bases: `object`

This is adapted from the listener code at

https://pypi.python.org/pypi/zeroconf

`add_service`(*zeroconf*, *type*, *name*)

`remove_service`(*zeroconf*, *type*, *name*)

Module contents

pyparrot.scripts package

Submodules

pyparrot.scripts.findMinidrone module

Find the BLE address for a mambo. To run this,

sudo python findMambo.py

Note that the sudo is necessary for BLE permissions on linux. It is only needed on this program and nothing else.

Author: Amy McGovern

`pyparrot.scripts.findMinidrone.`**`main`**()

Module contents

pyparrot.utils package

Submodules

pyparrot.utils.NonBlockingStreamReader module

A non-blocking stream reader (used to solve the process communciation with ffmpeg)

This code is almost directly from:

http://eyalarubas.com/python-subproc-nonblock.html

Amy McGovern (dramymcgovern@gmail.com) modified to allow the thread to end nicely and also to not throw an error if the stream ends, since our code already will know that from parsing (and the programs are not expected to run forever)

class `pyparrot.utils.NonBlockingStreamReader.`**`NonBlockingStreamReader`**(*stream*)

Bases: `object`

`finish_reader`()

`readline`(*timeout=None*)

exception `pyparrot.utils.NonBlockingStreamReader.`**`UnexpectedEndOfStream`**

Bases: `Exception`

pyparrot.utils.colorPrint module

Print messages in color

`pyparrot.utils.colorPrint.`**`color_print`**(*print_str*, *type='NONE'*)

pyparrot.utils.vlc module

This module provides bindings for the LibVLC public API, see U{http://wiki.videolan.org/LibVLC}.

You can find the documentation and a README file with some examples at U{http://www.olivieraubert.net/vlc/python-ctypes/}.

Basically, the most important class is L{Instance}, which is used to create a libvlc instance. From this instance, you then create L{MediaPlayer} and L{MediaListPlayer} instances.

Alternatively, you may create instances of the L{MediaPlayer} and L{MediaListPlayer} class directly and an instance of L{Instance} will be implicitly created. The latter can be obtained using the C{get_instance} method of L{MediaPlayer} and L{MediaListPlayer}.

class `pyparrot.utils.vlc.`**`AudioCleanupCb`**

Bases: `ctypes.c_void_p`

Callback prototype for audio playback cleanup. This is called when the media player no longer needs an audio output. @param opaque: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN].

class `pyparrot.utils.vlc.`**`AudioDrainCb`**

Bases: `ctypes.c_void_p`

Callback prototype for audio buffer drain. LibVLC may invoke this callback when the decoded audio track is ending. There will be no further decoded samples for the track, but playback should nevertheless continue until all already pending buffers are rendered. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN].

class `pyparrot.utils.vlc.`**`AudioFlushCb`**

Bases: `ctypes.c_void_p`

Callback prototype for audio buffer flush. LibVLC invokes this callback if it needs to discard all pending buffers and stop playback as soon as possible. This typically occurs when the media is stopped. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN].

class pyparrot.utils.vlc.**AudioOutput**

Bases: pyparrot.utils.vlc._Cstruct

description

Structure/Union member

name

Structure/Union member

next

Structure/Union member

class pyparrot.utils.vlc.**AudioOutputChannel**

Bases: pyparrot.utils.vlc._Enum

Audio channels.

Dolbys = pyparrot.utils.vlc.AudioOutputChannel.Dolbys

Error = pyparrot.utils.vlc.AudioOutputChannel.FIXME_(4294967295)

Left = pyparrot.utils.vlc.AudioOutputChannel.Left

RStereo = pyparrot.utils.vlc.AudioOutputChannel.RStereo

Right = pyparrot.utils.vlc.AudioOutputChannel.Right

Stereo = pyparrot.utils.vlc.AudioOutputChannel.Stereo

class pyparrot.utils.vlc.**AudioOutputDevice**

Bases: pyparrot.utils.vlc._Cstruct

description

Structure/Union member

device

Structure/Union member

next

Structure/Union member

class pyparrot.utils.vlc.**AudioOutputDeviceTypes**

Bases: pyparrot.utils.vlc._Enum

Audio device types.

Error = pyparrot.utils.vlc.AudioOutputDeviceTypes.FIXME_(4294967295)

Mono = pyparrot.utils.vlc.AudioOutputDeviceTypes.Mono

SPDIF = pyparrot.utils.vlc.AudioOutputDeviceTypes.SPDIF

Stereo = pyparrot.utils.vlc.AudioOutputDeviceTypes.Stereo

class pyparrot.utils.vlc.**AudioPauseCb**

Bases: ctypes.c_void_p

Callback prototype for audio pause. LibVLC invokes this callback to pause audio playback. @note: The pause callback is never called if the audio is already paused. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN]. @param pts: time stamp of the pause request (should be elapsed already).

class pyparrot.utils.vlc.**AudioPlayCb**

Bases: `ctypes.c_void_p`

Callback prototype for audio playback. The LibVLC media player decodes and post-processes the audio signal asynchronously (in an internal thread). Whenever audio samples are ready to be queued to the output, this callback is invoked. The number of samples provided per invocation may depend on the file format, the audio coding algorithm, the decoder plug-in, the post-processing filters and timing. Application must not assume a certain number of samples. The exact format of audio samples is determined by L{libvlc_audio_set_format}() or L{libvlc_audio_set_format_callbacks}() as is the channels layout. Note that the number of samples is per channel. For instance, if the audio track sampling rate is 48000 Hz, then 1200 samples represent 25 milliseconds of audio signal - regardless of the number of audio channels. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN]. @param samples: pointer to a table of audio samples to play back [IN]. @param count: number of audio samples to play back. @param pts: expected play time stamp (see libvlc_delay()).

class pyparrot.utils.vlc.**AudioResumeCb**

Bases: `ctypes.c_void_p`

Callback prototype for audio resumption. LibVLC invokes this callback to resume audio playback after it was previously paused. @note: The resume callback is never called if the audio is not paused. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN]. @param pts: time stamp of the resumption request (should be elapsed already).

class pyparrot.utils.vlc.**AudioSetVolumeCb**

Bases: `ctypes.c_void_p`

Callback prototype for audio volume change. @param data: data pointer as passed to L{libvlc_audio_set_callbacks}() [IN]. @param volume: software volume (1. = nominal, 0. = mute). @param mute: muted flag.

class pyparrot.utils.vlc.**AudioSetupCb**

Bases: `ctypes.c_void_p`

Callback prototype to setup the audio playback. This is called when the media player needs to create a new audio output. @param opaque: pointer to the data pointer passed to L{libvlc_audio_set_callbacks}() [IN/OUT]. @param format: 4 bytes sample format [IN/OUT]. @param rate: sample rate [IN/OUT]. @param channels: channels count [IN/OUT]. @return: 0 on success, anything else to skip audio playback.

class pyparrot.utils.vlc.**AudioTrack**

Bases: `pyparrot.utils.vlc._Cstruct`

channels

Structure/Union member

rate

Structure/Union member

class pyparrot.utils.vlc.**Callback**

Bases: `ctypes.c_void_p`

Callback function notification. @param p_event: the event triggering the callback.

class pyparrot.utils.vlc.**CallbackDecorators**

Bases: `object`

Class holding various method decorators for callback functions.

AudioCleanupCb

alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioDrainCb

alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioFlushCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioPauseCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioPlayCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioResumeCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioSetVolumeCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

AudioSetupCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

Callback
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

LogCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

MediaCloseCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

MediaOpenCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

MediaReadCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

MediaSeekCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

VideoCleanupCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

VideoDisplayCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

VideoFormatCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

VideoLockCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

VideoUnlockCb
alias of `ctypes.CFUNCTYPE.<locals>.CFunctionType`

class `pyparrot.utils.vlc.`**ChapterDescription**
Bases: `pyparrot.utils.vlc._Cstruct`

class `pyparrot.utils.vlc.`**DialogQuestionType**
Bases: `pyparrot.utils.vlc._Enum`

@defgroup libvlc_dialog libvlc dialog @ingroup libvlc @{ @file libvlc dialog external api.

CRITICAL = pyparrot.utils.vlc.DialogQuestionType.CRITICAL

NORMAL = pyparrot.utils.vlc.DialogQuestionType.NORMAL

WARNING = pyparrot.utils.vlc.DialogQuestionType.WARNING

class pyparrot.utils.vlc.**Event**

Bases: pyparrot.utils.vlc._Cstruct

object

Structure/Union member

type

Structure/Union member

u

Structure/Union member

class pyparrot.utils.vlc.**EventManager**

Bases: pyparrot.utils.vlc._Ctype

Create an event manager with callback handler.

This class interposes the registration and handling of event notifications in order to (a) remove the need for decorating each callback functions with the decorator '@callbackmethod', (b) allow any number of positional and/or keyword arguments to the callback (in addition to the Event instance) and (c) to preserve the Python objects such that the callback and argument objects remain alive (i.e. are not garbage collected) until B{after} the notification has been unregistered.

@note: Only a single notification can be registered for each event type in an EventManager instance.

event_attach(*eventtype*, *callback*, **args*, ***kwds*)

Register an event notification.

@param eventtype: the desired event type to be notified about. @param callback: the function to call when the event occurs. @param args: optional positional arguments for the callback. @param kwds: optional keyword arguments for the callback. @return: 0 on success, ENOMEM on error.

@note: The callback function must have at least one argument, an Event instance. Any other, optional positional and keyword arguments are in B{addition} to the first one.

event_detach(*eventtype*)

Unregister an event notification.

@param eventtype: the event type notification to be removed.

class pyparrot.utils.vlc.**EventType**

Bases: pyparrot.utils.vlc._Enum

Event types.

MediaDiscovererEnded = pyparrot.utils.vlc.EventType.MediaDiscovererEnded

MediaDiscovererStarted = pyparrot.utils.vlc.EventType.MediaDiscovererStarted

MediaDurationChanged = pyparrot.utils.vlc.EventType.MediaDurationChanged

MediaFreed = pyparrot.utils.vlc.EventType.MediaFreed

MediaListEndReached = pyparrot.utils.vlc.EventType.MediaListEndReached

MediaListItemAdded = pyparrot.utils.vlc.EventType.MediaListItemAdded

MediaListItemDeleted = pyparrot.utils.vlc.EventType.MediaListItemDeleted

MediaListPlayerNextItemSet = pyparrot.utils.vlc.EventType.MediaListPlayerNextItemSet

MediaListPlayerPlayed = pyparrot.utils.vlc.EventType.MediaListPlayerPlayed

MediaListPlayerStopped = pyparrot.utils.vlc.EventType.MediaListPlayerStopped

MediaListViewItemAdded = pyparrot.utils.vlc.EventType.MediaListViewItemAdded

MediaListViewItemDeleted = pyparrot.utils.vlc.EventType.MediaListViewItemDeleted

MediaListViewWillAddItem = pyparrot.utils.vlc.EventType.MediaListViewWillAddItem

MediaListViewWillDeleteItem = pyparrot.utils.vlc.EventType.MediaListViewWillDeleteItem

MediaListWillAddItem = pyparrot.utils.vlc.EventType.MediaListWillAddItem

MediaListWillDeleteItem = pyparrot.utils.vlc.EventType.MediaListWillDeleteItem

MediaMetaChanged = pyparrot.utils.vlc.EventType.MediaMetaChanged

MediaParsedChanged = pyparrot.utils.vlc.EventType.MediaParsedChanged

MediaPlayerAudioDevice = pyparrot.utils.vlc.EventType.MediaPlayerAudioDevice

MediaPlayerAudioVolume = pyparrot.utils.vlc.EventType.MediaPlayerAudioVolume

MediaPlayerBackward = pyparrot.utils.vlc.EventType.MediaPlayerBackward

MediaPlayerBuffering = pyparrot.utils.vlc.EventType.MediaPlayerBuffering

MediaPlayerChapterChanged = pyparrot.utils.vlc.EventType.MediaPlayerChapterChanged

MediaPlayerCorked = pyparrot.utils.vlc.EventType.MediaPlayerCorked

MediaPlayerESAdded = pyparrot.utils.vlc.EventType.MediaPlayerESAdded

MediaPlayerESDeleted = pyparrot.utils.vlc.EventType.MediaPlayerESDeleted

MediaPlayerESSelected = pyparrot.utils.vlc.EventType.MediaPlayerESSelected

MediaPlayerEncounteredError = pyparrot.utils.vlc.EventType.MediaPlayerEncounteredError

MediaPlayerEndReached = pyparrot.utils.vlc.EventType.MediaPlayerEndReached

MediaPlayerForward = pyparrot.utils.vlc.EventType.MediaPlayerForward

MediaPlayerLengthChanged = pyparrot.utils.vlc.EventType.MediaPlayerLengthChanged

MediaPlayerMediaChanged = pyparrot.utils.vlc.EventType.MediaPlayerMediaChanged

MediaPlayerMuted = pyparrot.utils.vlc.EventType.MediaPlayerMuted

MediaPlayerNothingSpecial = pyparrot.utils.vlc.EventType.MediaPlayerNothingSpecial

MediaPlayerOpening = pyparrot.utils.vlc.EventType.MediaPlayerOpening

MediaPlayerPausableChanged = pyparrot.utils.vlc.EventType.MediaPlayerPausableChanged

MediaPlayerPaused = pyparrot.utils.vlc.EventType.MediaPlayerPaused

MediaPlayerPlaying = pyparrot.utils.vlc.EventType.MediaPlayerPlaying

MediaPlayerPositionChanged = pyparrot.utils.vlc.EventType.MediaPlayerPositionChanged

MediaPlayerScrambledChanged = pyparrot.utils.vlc.EventType.MediaPlayerScrambledChanged

MediaPlayerSeekableChanged = pyparrot.utils.vlc.EventType.MediaPlayerSeekableChanged

MediaPlayerSnapshotTaken = pyparrot.utils.vlc.EventType.MediaPlayerSnapshotTaken

MediaPlayerStopped = pyparrot.utils.vlc.EventType.MediaPlayerStopped

MediaPlayerTimeChanged = pyparrot.utils.vlc.EventType.MediaPlayerTimeChanged

MediaPlayerTitleChanged = pyparrot.utils.vlc.EventType.MediaPlayerTitleChanged

MediaPlayerUncorked = pyparrot.utils.vlc.EventType.MediaPlayerUncorked

MediaPlayerUnmuted = pyparrot.utils.vlc.EventType.MediaPlayerUnmuted

MediaPlayerVout = pyparrot.utils.vlc.EventType.MediaPlayerVout

MediaStateChanged = pyparrot.utils.vlc.EventType.MediaStateChanged

MediaSubItemAdded = pyparrot.utils.vlc.EventType.MediaSubItemAdded

MediaSubItemTreeAdded = pyparrot.utils.vlc.EventType.MediaSubItemTreeAdded

RendererDiscovererItemAdded = pyparrot.utils.vlc.EventType.RendererDiscovererItemAdded

RendererDiscovererItemDeleted = pyparrot.utils.vlc.EventType.RendererDiscovererItemDel

VlmMediaAdded = pyparrot.utils.vlc.EventType.VlmMediaAdded

VlmMediaChanged = pyparrot.utils.vlc.EventType.VlmMediaChanged

VlmMediaInstanceStarted = pyparrot.utils.vlc.EventType.VlmMediaInstanceStarted

VlmMediaInstanceStatusEnd = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusEnd

VlmMediaInstanceStatusError = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusError

VlmMediaInstanceStatusInit = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusInit

VlmMediaInstanceStatusOpening = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusOpe

VlmMediaInstanceStatusPause = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusPause

VlmMediaInstanceStatusPlaying = pyparrot.utils.vlc.EventType.VlmMediaInstanceStatusPla

VlmMediaInstanceStopped = pyparrot.utils.vlc.EventType.VlmMediaInstanceStopped

VlmMediaRemoved = pyparrot.utils.vlc.EventType.VlmMediaRemoved

class pyparrot.utils.vlc.**EventUnion**

Bases: `_ctypes.Union`

filename
Structure/Union member

media
Structure/Union member

media_event
Structure/Union member

meta_type
Structure/Union member

new_cache
Structure/Union member

new_child
Structure/Union member

new_count
Structure/Union member

new_duration
Structure/Union member

new_length
Structure/Union member

new_pausable
Structure/Union member

new_position
Structure/Union member

new_scrambled
Structure/Union member

new_seekable
Structure/Union member

new_state
Structure/Union member

new_status
Structure/Union member

new_time
Structure/Union member

new_title
Structure/Union member

class pyparrot.utils.vlc.**FILE**
Bases: `_ctypes.Structure`

pyparrot.utils.vlc.**FILE_ptr**
alias of `pyparrot.utils.vlc.LP_FILE`

class pyparrot.utils.vlc.**Instance**
Bases: `pyparrot.utils.vlc._Ctype`

Create a new Instance instance.

It may take as parameter either:

- a string
- a list of strings as first parameters
- the parameters given as the constructor parameters (must be strings)

add_intf(*name*)
Try to start a user interface for the libvlc instance. @param name: interface name, or None for default. @return: 0 on success, -1 on error.

audio_filter_list_get()
Returns a list of available audio filters.

audio_output_device_count(*psz_audio_output*)
Backward compatibility stub. Do not use in new code. deprecated Use L{audio_output_device_list_get}() instead. @return: always 0.

audio_output_device_id(*psz_audio_output*, *i_device*)
Backward compatibility stub. Do not use in new code. deprecated Use L{audio_output_device_list_get}() instead. @return: always None.

audio_output_device_list_get(*aout*)
Gets a list of audio output devices for a given audio output module, See L{audio_output_device_set}(). @note: Not all audio outputs support this. In particular, an empty (None) list of devices does B{not} imply that the specified audio output does not work. @note: The list might not be exhaustive. @warning: Some audio output devices in the list might not actually work in some circumstances. By default, it is recommended to not specify any explicit audio device. @param aout: audio output name (as returned by L{audio_output_list_get}()). @return: A None-terminated linked list of potential audio output devices. It must be freed with L{audio_output_device_list_release}(). @version: LibVLC 2.1.0 or later.

audio_output_device_longname(*psz_output*, *i_device*)
Backward compatibility stub. Do not use in new code. deprecated Use L{audio_output_device_list_get}() instead. @return: always None.

audio_output_enumerate_devices()
Enumerate the defined audio output devices.

@return: list of dicts {name:, description:, devices:}

audio_output_list_get()
Gets the list of available audio output modules. @return: list of available audio outputs. It must be freed with In case of error, None is returned.

get_log_verbosity()
Always returns minus one. This function is only provided for backward compatibility. @return: always -1.

log_open()
This function does nothing useful. It is only provided for backward compatibility. @return: an unique pointer or None on error.

log_set(*cb*, *data*)
Sets the logging callback for a LibVLC instance. This function is thread-safe: it will wait for any pending callbacks invocation to complete. @param data: opaque data pointer for the callback function @note Some log messages (especially debug) are emitted by LibVLC while is being initialized. These messages cannot be captured with this interface. @warning A deadlock may occur if this function is called from the callback. @param p_instance: libvlc instance. @version: LibVLC 2.1.0 or later.

log_set_file(*stream*)
Sets up logging to a file. @param stream: FILE pointer opened for writing (the FILE pointer must remain valid until L{log_unset}()). @version: LibVLC 2.1.0 or later.

log_unset()
Unsets the logging callback. This function deregisters the logging callback for a LibVLC instance. This is rarely needed as the callback is implicitly unset when the instance is destroyed. @note: This function will wait for any pending callbacks invocation to complete (causing a deadlock if called from within the callback). @version: LibVLC 2.1.0 or later.

media_discoverer_list_get(*i_cat*, *ppp_services*)
Get media discoverer services by category. @param i_cat: category of services to fetch. @param ppp_services: address to store an allocated array of media discoverer services (must be freed with L{media_discoverer_list_release}() by the caller) [OUT]. @return: the number of media discoverer services (0 on error). @version: LibVLC 3.0.0 and later.

media_discoverer_new(*psz_name*)
Create a media discoverer object by name. After this object is created, you should attach to media_list events in order to be notified of new items discovered. You need to call L{media_discoverer_start}() in order to start the discovery. See L{media_discoverer_media_list} See L{media_discoverer_event_manager} See L{media_discoverer_start}. @param psz_name: service name; use L{media_discoverer_list_get}() to get a list of the discoverer names available in this libVLC instance. @return: media discover object or None in case of error. @version: LibVLC 3.0.0 or later.

media_discoverer_new_from_name(*psz_name*)
deprecated Use L{media_discoverer_new}() and L{media_discoverer_start}().

media_library_new()
Create an new Media Library object. @return: a new object or None on error.

media_list_new(*mrls=None*)
Create a new MediaList instance. @param mrls: optional list of MRL strings

media_list_player_new()
Create a new MediaListPlayer instance.

media_new(*mrl*, **options*)
Create a new Media instance.

If mrl contains a colon (:) preceded by more than 1 letter, it will be treated as a URL. Else, it will be considered as a local path. If you need more control, directly use media_new_location/media_new_path methods.

Options can be specified as supplementary string parameters, but note that many options cannot be set at the media level, and rather at the Instance level. For instance, the marquee filter must be specified when creating the vlc.Instance or vlc.MediaPlayer.

Alternatively, options can be added to the media using the Media.add_options method (with the same limitation).

@param options: optional media option=value strings

media_new_as_node(*psz_name*)
Create a media as an empty node with a given name. See L{media_release}. @param psz_name: the name of the node. @return: the new empty media or None on error.

media_new_callbacks(*open_cb*, *read_cb*, *seek_cb*, *close_cb*, *opaque*)
Create a media with custom callbacks to read the data from. @param open_cb: callback to open the custom bitstream input media. @param read_cb: callback to read data (must not be None). @param seek_cb: callback to seek, or None if seeking is not supported. @param close_cb: callback to close the media, or None if unnecessary. @param opaque: data pointer for the open callback. @return: the newly created media or None on error @note If open_cb is None, the opaque pointer will be passed to read_cb, seek_cb and close_cb, and the stream size will be treated as unknown. @note The callbacks may be called asynchronously (from another thread). A single stream instance need not be reentrant. However the open_cb needs to be reentrant if the media is used by multiple player instances. @warning The callbacks may be used until all or any player instances that were supplied the media item are stopped. See L{media_release}. @version: LibVLC 3.0.0 and later.

media_new_fd(*fd*)
Create a media for an already open file descriptor. The file descriptor shall be open for reading (or reading and writing). Regular file descriptors, pipe read descriptors and character device descriptors (including TTYs) are supported on all platforms. Block device descriptors are supported where available. Directory descriptors are supported on systems that provide fdopendir(). Sockets are supported on all platforms where they are file descriptors, i.e. all except Windows. @note: This library will B{not} automatically close the file descriptor under any circumstance. Nevertheless, a file descriptor can usually only be rendered once in a media player. To render it a second time, the file descriptor should probably be rewound to the beginning with lseek(). See L{media_release}. @param fd: open file descriptor. @return: the newly created media or None on error. @version: LibVLC 1.1.5 and later.

media_new_location(*psz_mrl*)
Create a media with a certain given media resource location, for instance a valid URL. @note: To refer to a local file with this function, the file://... URI syntax B{must} be used (see IETF RFC3986). We recommend using L{media_new_path}() instead when dealing with local files. See L{media_release}. @param psz_mrl: the media location. @return: the newly created media or None on error.

media_new_path(*path*)
Create a media for a certain file path. See L{media_release}. @param path: local filesystem path. @return: the newly created media or None on error.

media_player_new(*uri=None*)
Create a new MediaPlayer instance.

@param uri: an optional URI to play in the player.

playlist_play(*i_id*, *i_options*, *ppsz_options*)
Start playing (if there is any item in the playlist). Additionnal playlist item options can be specified for addition to the item before it is played. @param i_id: the item to play. If this is a negative number, the next item will be selected. Otherwise, the item with the given ID will be played. @param i_options: the number of options to add to the item. @param ppsz_options: the options to add to the item.

release()
Decrement the reference count of a libvlc instance, and destroy it if it reaches zero.

renderer_discoverer_list_get(*ppp_services*)
Get media discoverer services See libvlc_renderer_list_release(). @param ppp_services: address to store an allocated array of renderer discoverer services (must be freed with libvlc_renderer_list_release() by the caller) [OUT]. @return: the number of media discoverer services (0 on error). @version: LibVLC 3.0.0 and later.

renderer_discoverer_new(*psz_name*)
Create a renderer discoverer object by name After this object is created, you should attach to events in order to be notified of the discoverer events. You need to call L{renderer_discoverer_start}() in order to start the discovery. See L{renderer_discoverer_event_manager}() See L{renderer_discoverer_start}(). @param psz_name: service name; use L{renderer_discoverer_list_get}() to get a list of the discoverer names available in this libVLC instance. @return: media discover object or None in case of error. @version: LibVLC 3.0.0 or later.

retain()
Increments the reference count of a libvlc instance. The initial reference count is 1 after L{new}() returns.

set_app_id(*id*, *version*, *icon*)
Sets some meta-information about the application. See also L{set_user_agent}(). @param id: Java-style application identifier, e.g. "com.acme.foobar". @param version: application version numbers, e.g. "1.2.3". @param icon: application icon name, e.g. "foobar". @version: LibVLC 2.1.0 or later.

set_log_verbosity(*level*)
This function does nothing. It is only provided for backward compatibility. @param level: ignored.

set_user_agent(*name*, *http*)
Sets the application name. LibVLC passes this as the user agent string when a protocol requires it. @param name: human-readable application name, e.g. "FooBar player 1.2.3". @param http: HTTP User Agent, e.g. "FooBar/1.2.3 Python/2.6.0". @version: LibVLC 1.1.1 or later.

video_filter_list_get()
Returns a list of available video filters.

vlm_add_broadcast(*psz_name*, *psz_input*, *psz_output*, *i_options*, *ppsz_options*, *b_enabled*, *b_loop*)
Add a broadcast, with one input. @param psz_name: the name of the new broadcast. @param psz_input: the input MRL. @param psz_output: the output MRL (the parameter to the "sout" variable). @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new broadcast. @param b_loop: Should this broadcast be played in loop ? @return: 0 on success, -1 on error.

vlm_add_input(*psz_name*, *psz_input*)
Add a media's input MRL. This will add the specified one. @param psz_name: the media to work on. @param psz_input: the input MRL. @return: 0 on success, -1 on error.

vlm_add_vod(*psz_name*, *psz_input*, *i_options*, *ppsz_options*, *b_enabled*, *psz_mux*)
Add a vod, with one input. @param psz_name: the name of the new vod media. @param psz_input: the input MRL. @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new vod. @param psz_mux: the muxer of the vod media. @return: 0 on success, -1 on error.

vlm_change_media (*psz_name*, *psz_input*, *psz_output*, *i_options*, *ppsz_options*, *b_enabled*, *b_loop*)
Edit the parameters of a media. This will delete all existing inputs and add the specified one. @param psz_name: the name of the new broadcast. @param psz_input: the input MRL. @param psz_output: the output MRL (the parameter to the "sout" variable). @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new broadcast. @param b_loop: Should this broadcast be played in loop ? @return: 0 on success, -1 on error.

vlm_del_media (*psz_name*)
Delete a media (VOD or broadcast). @param psz_name: the media to delete. @return: 0 on success, -1 on error.

vlm_get_event_manager = functools.partial(<bound method memoize_parameterless.__call__

vlm_get_media_instance_chapter (*psz_name*, *i_instance*)
Get vlm_media instance chapter number by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: chapter as number or -1 on error. @bug: will always return 0.

vlm_get_media_instance_length (*psz_name*, *i_instance*)
Get vlm_media instance length by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: length of media item or -1 on error.

vlm_get_media_instance_position (*psz_name*, *i_instance*)
Get vlm_media instance position by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: position as float or -1. on error.

vlm_get_media_instance_rate (*psz_name*, *i_instance*)
Get vlm_media instance playback rate by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: playback rate or -1 on error.

vlm_get_media_instance_seekable (*psz_name*, *i_instance*)
Is libvlc instance seekable ? @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: 1 if seekable, 0 if not, -1 if media does not exist. @bug: will always return 0.

vlm_get_media_instance_time (*psz_name*, *i_instance*)
Get vlm_media instance time by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: time as integer or -1 on error.

vlm_get_media_instance_title (*psz_name*, *i_instance*)
Get vlm_media instance title number by name or instance id. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: title as number or -1 on error. @bug: will always return 0.

vlm_pause_media (*psz_name*)
Pause the named broadcast. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

vlm_play_media (*psz_name*)
Play the named broadcast. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

vlm_release ()
Release the vlm instance related to the given L{Instance}.

vlm_seek_media (*psz_name*, *f_percentage*)
Seek in the named broadcast. @param psz_name: the name of the broadcast. @param f_percentage: the percentage to seek to. @return: 0 on success, -1 on error.

vlm_set_enabled (*psz_name*, *b_enabled*)
Enable or disable a media (VOD or broadcast). @param psz_name: the media to work on. @param b_enabled: the new status. @return: 0 on success, -1 on error.

vlm_set_input(*psz_name*, *psz_input*)
Set a media's input MRL. This will delete all existing inputs and add the specified one. @param psz_name: the media to work on. @param psz_input: the input MRL. @return: 0 on success, -1 on error.

vlm_set_loop(*psz_name*, *b_loop*)
Set a media's loop status. @param psz_name: the media to work on. @param b_loop: the new status. @return: 0 on success, -1 on error.

vlm_set_mux(*psz_name*, *psz_mux*)
Set a media's vod muxer. @param psz_name: the media to work on. @param psz_mux: the new muxer. @return: 0 on success, -1 on error.

vlm_set_output(*psz_name*, *psz_output*)
Set the output for a media. @param psz_name: the media to work on. @param psz_output: the output MRL (the parameter to the "sout" variable). @return: 0 on success, -1 on error.

vlm_show_media(*psz_name*)
Return information about the named media as a JSON string representation. This function is mainly intended for debugging use, if you want programmatic access to the state of a vlm_media_instance_t, please use the corresponding libvlc_vlm_get_media_instance_xxx -functions. Currently there are no such functions available for vlm_media_t though. @param psz_name: the name of the media, if the name is an empty string, all media is described. @return: string with information about named media, or None on error.

vlm_stop_media(*psz_name*)
Stop the named broadcast. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

wait()
Waits until an interface causes the instance to exit. You should start at least one interface first, using L{add_intf}().

class `pyparrot.utils.vlc.`**ListPOINTER**(*etype*)
Bases: `object`

Just like a POINTER but accept a list of ctype as an argument.

from_param(*param*)

class `pyparrot.utils.vlc.`**Log**
Bases: `_ctypes.Structure`

class `pyparrot.utils.vlc.`**LogCb**
Bases: `ctypes.c_void_p`

Callback prototype for LibVLC log message handler. @param data: data pointer as given to L{libvlc_log_set}(). @param level: message level (@ref libvlc_log_level). @param ctx: message context (meta-information about the message). @param fmt: printf() format string (as defined by ISO C11). @param args: variable argument list for the format @note Log message handlers B{must} be thread-safe. @warning The message context pointer, the format string parameters and the variable arguments are only valid until the callback returns.

class `pyparrot.utils.vlc.`**LogIterator**
Bases: `pyparrot.utils.vlc._Ctype`

Create a new VLC log iterator.

free()
Frees memory allocated by L{log_get_iterator}().

has_next()
Always returns zero. This function is only provided for backward compatibility. @return: always zero.

next()

class pyparrot.utils.vlc.**LogLevel**
Bases: pyparrot.utils.vlc._Enum

Logging messages level.

ote future libvlc versions may define new levels.

DEBUG = pyparrot.utils.vlc.LogLevel.DEBUG

ERROR = pyparrot.utils.vlc.LogLevel.ERROR

NOTICE = pyparrot.utils.vlc.LogLevel.NOTICE

WARNING = pyparrot.utils.vlc.LogLevel.WARNING

class pyparrot.utils.vlc.**LogMessage**
Bases: pyparrot.utils.vlc._Cstruct

header
Structure/Union member

message
Structure/Union member

name
Structure/Union member

severity
Structure/Union member

size
Structure/Union member

type
Structure/Union member

pyparrot.utils.vlc.**Log_ptr**
alias of pyparrot.utils.vlc.LP_Log

class pyparrot.utils.vlc.**Media**
Bases: pyparrot.utils.vlc._Ctype

Create a new Media instance.

Usage: Media(MRL, *options)

See vlc.Instance.media_new documentation for details.

add_option(*psz_options*)
Add an option to the media. This option will be used to determine how the media_player will read the media. This allows to use VLC's advanced reading/streaming options on a per-media basis. @note: The options are listed in 'vlc –long-help' from the command line, e.g. "-sout-all". Keep in mind that available options and their semantics vary across LibVLC versions and builds. @warning: Not all options affects L{Media} objects: Specifically, due to architectural issues most audio and video options, such as text renderer options, have no effects on an individual media. These options must be set through L{new}() instead. @param psz_options: the options (as a string).

add_option_flag(*psz_options*, *i_flags*)
Add an option to the media with configurable flags. This option will be used to determine how the media_player will read the media. This allows to use VLC's advanced reading/streaming options on a per-media basis. The options are detailed in vlc –long-help, for instance "–sout-all". Note that all options are not usable on medias: specifically, due to architectural issues, video-related options such as text renderer

options cannot be set on a single media. They must be set on the whole libvlc instance instead. @param psz_options: the options (as a string). @param i_flags: the flags for this option.

add_options(**options*)
Add a list of options to the media.

Options must be written without the double-dash. Warning: most audio and video options, such as text renderer, have no effects on an individual media. These options must be set at the vlc.Instance or vlc.MediaPlayer instanciation.

@param options: optional media option=value strings

duplicate()
Duplicate a media descriptor object.

event_manager = functools.partial(<bound method memoize_parameterless.__call__ of Get

get_duration()
Get duration (in ms) of media descriptor object item. @return: duration of media item or -1 on error.

get_instance()

get_meta(*e_meta*)
Read the meta of the media. If the media has not yet been parsed this will return None. See L{parse} See L{parse_with_options} See libvlc_MediaMetaChanged. @param e_meta: the meta to read. @return: the media's meta.

get_mrl()
Get the media resource locator (mrl) from a media descriptor object. @return: string with mrl of media descriptor object.

get_parsed_status()
Get Parsed status for media descriptor object. See libvlc_MediaParsedChanged See libvlc_media_parsed_status_t. @return: a value of the libvlc_media_parsed_status_t enum. @version: LibVLC 3.0.0 or later.

get_state()
Get current state of media descriptor object. Possible media states are libvlc_NothingSpecial=0, libvlc_Opening, libvlc_Playing, libvlc_Paused, libvlc_Stopped, libvlc_Ended, libvlc_Error. See libvlc_state_t. @return: state of media descriptor object.

get_stats(*p_stats*)
Get the current statistics about the media. @param p_stats:: structure that contain the statistics about the media (this structure must be allocated by the caller). @return: true if the statistics are available, false otherwise libvlc_return_bool.

get_tracks_info()
Get media descriptor's elementary streams description Note, you need to call L{parse}() or play the media at least once before calling this function. Not doing this will result in an empty array. deprecated Use L{tracks_get}() instead. @param tracks: address to store an allocated array of Elementary Streams descriptions (must be freed by the caller) [OUT]. @return: the number of Elementary Streams.

get_type()
Get the media type of the media descriptor object. @return: media type. @version: LibVLC 3.0.0 and later. See libvlc_media_type_t.

get_user_data()
Get media descriptor's user_data. user_data is specialized data accessed by the host application, VLC.framework uses it as a pointer to an native object that references a L{Media} pointer.

is_parsed()
Return true is the media descriptor object is parsed deprecated This can return true in case of failure.

> Use L{get_parsed_status}() instead

See libvlc_MediaParsedChanged. @return: true if media object has been parsed otherwise it returns false libvlc_return_bool.

parse()
: Parse a media. This fetches (local) art, meta data and tracks information. The method is synchronous. deprecated This function could block indefinitely.

> Use L{parse_with_options}() instead

See L{parse_with_options} See L{get_meta} See L{get_tracks_info}.

parse_async()
: Parse a media. This fetches (local) art, meta data and tracks information. The method is the asynchronous of L{parse}(). To track when this is over you can listen to libvlc_MediaParsedChanged event. However if the media was already parsed you will not receive this event. deprecated You can't be sure to receive the libvlc_MediaParsedChanged

> event (you can wait indefinitely for this event). Use L{parse_with_options}() instead

See L{parse} See libvlc_MediaParsedChanged See L{get_meta} See L{get_tracks_info}.

parse_stop()
: Stop the parsing of the media When the media parsing is stopped, the libvlc_MediaParsedChanged event will be sent with the libvlc_media_parsed_status_timeout status. See L{parse_with_options}. @version: LibVLC 3.0.0 or later.

parse_with_options(*parse_flag*, *timeout*)
: Parse the media asynchronously with options. This fetches (local or network) art, meta data and/or tracks information. This method is the extended version of L{parse_with_options}(). To track when this is over you can listen to libvlc_MediaParsedChanged event. However if this functions returns an error, you will not receive any events. It uses a flag to specify parse options (see libvlc_media_parse_flag_t). All these flags can be combined. By default, media is parsed if it's a local file. @note: Parsing can be aborted with L{parse_stop}(). See libvlc_MediaParsedChanged See L{get_meta} See L{tracks_get} See L{get_parsed_status} See libvlc_media_parse_flag_t. @param parse_flag: parse options: @param timeout: maximum time allowed to preparse the media. If -1, the default "preparse-timeout" option will be used as a timeout. If 0, it will wait indefinitely. If > 0, the timeout will be used (in milliseconds). @return: -1 in case of error, 0 otherwise. @version: LibVLC 3.0.0 or later.

player_new_from_media()
: Create a Media Player object from a Media. @return: a new media player object, or None on error.

release()
: Decrement the reference count of a media descriptor object. If the reference count is 0, then L{release}() will release the media descriptor object. It will send out an libvlc_MediaFreed event to all listeners. If the media descriptor object has been released it should not be used again.

retain()
: Retain a reference to a media descriptor object (libvlc_media_t). Use L{release}() to decrement the reference count of a media descriptor object.

save_meta()
: Save the meta previously set. @return: true if the write operation was successful.

set_meta(*e_meta*, *psz_value*)
: Set the meta of the media (this function will not save the meta, call L{save_meta} in order to save the meta). @param e_meta: the meta to write. @param psz_value: the media's meta.

set_user_data(*p_new_user_data*)
: Sets media descriptor's user_data. user_data is specialized data accessed by the host application,

VLC.framework uses it as a pointer to an native object that references a L{Media} pointer. @param p_new_user_data: pointer to user data.

slaves_add(*i_type*, *i_priority*, *psz_uri*)
Add a slave to the current media. A slave is an external input source that may contains an additional subtitle track (like a .srt) or an additional audio track (like a .ac3). @note: This function must be called before the media is parsed (via L{parse_with_options}()) or before the media is played (via L{player_play}()). @param i_type: subtitle or audio. @param i_priority: from 0 (low priority) to 4 (high priority). @param psz_uri: Uri of the slave (should contain a valid scheme). @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 and later.

slaves_clear()
Clear all slaves previously added by L{slaves_add}() or internally. @version: LibVLC 3.0.0 and later.

slaves_get(*ppp_slaves*)
Get a media descriptor's slave list The list will contain slaves parsed by VLC or previously added by L{slaves_add}(). The typical use case of this function is to save a list of slave in a database for a later use. @param ppp_slaves: address to store an allocated array of slaves (must be freed with L{slaves_release}()) [OUT]. @return: the number of slaves (zero on error). @version: LibVLC 3.0.0 and later. See L{slaves_add}.

subitems()
Get subitems of media descriptor object. This will increment the reference count of supplied media descriptor object. Use L{list_release}() to decrement the reference counting. @return: list of media descriptor subitems or None.

tracks_get()
Get media descriptor's elementary streams description Note, you need to call L{parse}() or play the media at least once before calling this function. Not doing this will result in an empty array. The result must be freed with L{tracks_release}. @version: LibVLC 2.1.0 and later.

class pyparrot.utils.vlc.**MediaCloseCb**
Bases: ctypes.c_void_p

Callback prototype to close a custom bitstream input media. @param opaque: private pointer as set by the @ref libvlc_media_open_cb callback.

class pyparrot.utils.vlc.**MediaDiscoverer**
Bases: pyparrot.utils.vlc._Ctype

N/A

event_manager = functools.partial(<bound method memoize_parameterless.__call__ of Get

is_running()
Query if media service discover object is running. @return: true if running, false if not libvlc_return_bool.

localized_name()
Get media service discover object its localized name. deprecated Useless, use L{list_get}() to get the longname of the service discovery. @return: localized name or None if the media_discoverer is not started.

media_list()
Get media service discover media list. @return: list of media items.

release()
Release media discover object. If the reference count reaches 0, then the object will be released.

start()
Start media discovery. To stop it, call L{stop}() or L{list_release}() directly. See L{stop}. @return: -1 in case of error, 0 otherwise. @version: LibVLC 3.0.0 or later.

stop()
Stop media discovery. See L{start}. @version: LibVLC 3.0.0 or later.

class pyparrot.utils.vlc.**MediaDiscovererCategory**
Bases: pyparrot.utils.vlc._Enum

Category of a media discoverer See libvlc_media_discoverer_list_get().

devices = pyparrot.utils.vlc.MediaDiscovererCategory.devices

lan = pyparrot.utils.vlc.MediaDiscovererCategory.lan

localdirs = pyparrot.utils.vlc.MediaDiscovererCategory.localdirs

podcasts = pyparrot.utils.vlc.MediaDiscovererCategory.podcasts

class pyparrot.utils.vlc.**MediaEvent**
Bases: pyparrot.utils.vlc._Cstruct

instance_name
Structure/Union member

media_name
Structure/Union member

class pyparrot.utils.vlc.**MediaLibrary**
Bases: pyparrot.utils.vlc._Ctype

N/A

load()
Load media library. @return: 0 on success, -1 on error.

media_list()
Get media library subitems. @return: media list subitems.

release()
Release media library object. This functions decrements the reference count of the media library object. If it reaches 0, then the object will be released.

retain()
Retain a reference to a media library object. This function will increment the reference counting for this object. Use L{release}() to decrement the reference count.

class pyparrot.utils.vlc.**MediaList**
Bases: pyparrot.utils.vlc._Ctype

Create a new MediaList instance.

Usage: MediaList(list_of_MRLs)

See vlc.Instance.media_list_new documentation for details.

add_media(*mrl*)
Add media instance to media list.

The L{lock} should be held upon entering this function. @param mrl: a media instance or a MRL. @return: 0 on success, -1 if the media list is read-only.

count()
Get count on media list items The L{lock} should be held upon entering this function. @return: number of items in media list.

event_manager = functools.partial(<bound method memoize_parameterless.__call__ of Get

get_instance()

index_of_item(*p_md*)
Find index position of List media instance in media list. Warning: the function will return the first matched position. The L{lock} should be held upon entering this function. @param p_md: media instance. @return: position of media instance or -1 if media not found.

insert_media(*p_md*, *i_pos*)
Insert media instance in media list on a position The L{lock} should be held upon entering this function. @param p_md: a media instance. @param i_pos: position in array where to insert. @return: 0 on success, -1 if the media list is read-only.

is_readonly()
This indicates if this media list is read-only from a user point of view. @return: 1 on readonly, 0 on readwrite libvlc_return_bool.

item_at_index(*i_pos*)
List media instance in media list at a position The L{lock} should be held upon entering this function. @param i_pos: position in array where to insert. @return: media instance at position i_pos, or None if not found. In case of success, L{media_retain}() is called to increase the refcount on the media.

lock()
Get lock on media list items.

media()
Get media instance from this media list instance. This action will increase the refcount on the media instance. The L{lock} should NOT be held upon entering this function. @return: media instance.

release()
Release media list created with L{new}().

remove_index(*i_pos*)
Remove media instance from media list on a position The L{lock} should be held upon entering this function. @param i_pos: position in array where to insert. @return: 0 on success, -1 if the list is read-only or the item was not found.

retain()
Retain reference to a media list.

set_media(*p_md*)
Associate media instance with this media list instance. If another media instance was present it will be released. The L{lock} should NOT be held upon entering this function. @param p_md: media instance to add.

unlock()
Release lock on media list items The L{lock} should be held upon entering this function.

class pyparrot.utils.vlc.**MediaListPlayer**
Bases: `pyparrot.utils.vlc._Ctype`

Create a new MediaListPlayer instance.

It may take as parameter either:

- a vlc.Instance
- nothing

event_manager = functools.partial(<bound method memoize_parameterless.__call__ of Retu

get_instance()
Return the associated Instance.

get_media_player()
Get media player of the media_list_player instance. @return: media player instance @note the caller is responsible for releasing the returned instance.

get_state()
Get current libvlc_state of media list player. @return: libvlc_state_t for media list player.

is_playing()
Is media list playing? @return: true for playing and false for not playing libvlc_return_bool.

next()
Play next item from media list. @return: 0 upon success -1 if there is no next item.

pause()
Toggle pause (or resume) media list.

play()
Play media list.

play_item(*p_md*)
Play the given media item. @param p_md: the media instance. @return: 0 upon success, -1 if the media is not part of the media list.

play_item_at_index(*i_index*)
Play media list item at position index. @param i_index: index in media list to play. @return: 0 upon success -1 if the item wasn't found.

previous()
Play previous item from media list. @return: 0 upon success -1 if there is no previous item.

release()
Release a media_list_player after use Decrement the reference count of a media player object. If the reference count is 0, then L{release}() will release the media player object. If the media player object has been released, then it should not be used again.

retain()
Retain a reference to a media player list object. Use L{release}() to decrement reference count.

set_media_list(*p_mlist*)
Set the media list associated with the player. @param p_mlist: list of media.

set_media_player(*p_mi*)
Replace media player in media_list_player with this instance. @param p_mi: media player instance.

set_pause(*do_pause*)
Pause or resume media list. @param do_pause: play/resume if zero, pause if non-zero. @version: LibVLC 3.0.0 or later.

set_playback_mode(*e_mode*)
Sets the playback mode for the playlist. @param e_mode: playback mode specification.

stop()
Stop playing media list.

class pyparrot.utils.vlc.**MediaOpenCb**
Bases: `ctypes.c_void_p`

Callback prototype to open a custom bitstream input media. The same media item can be opened multiple times. Each time, this callback is invoked. It should allocate and initialize any instance-specific resources, then store them in *datap. The instance resources can be freed in the @ref libvlc_media_close_cb callback. @param opaque: private pointer as passed to L{libvlc_media_new_callbacks}(). @return: datap storage space for a private data pointer, sizep byte length of the bitstream or UINT64_MAX if unknown.

class pyparrot.utils.vlc.**MediaParseFlag**

Bases: pyparrot.utils.vlc._Enum

Parse flags used by libvlc_media_parse_with_options() See libvlc_media_parse_with_options.

interact = pyparrot.utils.vlc.MediaParseFlag.interact

local = pyparrot.utils.vlc.MediaParseFlag.local

network = pyparrot.utils.vlc.MediaParseFlag.network

class pyparrot.utils.vlc.**MediaParsedStatus**

Bases: pyparrot.utils.vlc._Enum

Parse status used sent by libvlc_media_parse_with_options() or returned by libvlc_media_get_parsed_status() See libvlc_media_parse_with_options See libvlc_media_get_parsed_status.

done = pyparrot.utils.vlc.MediaParsedStatus.done

failed = pyparrot.utils.vlc.MediaParsedStatus.failed

skipped = pyparrot.utils.vlc.MediaParsedStatus.skipped

timeout = pyparrot.utils.vlc.MediaParsedStatus.timeout

class pyparrot.utils.vlc.**MediaPlayer**

Bases: pyparrot.utils.vlc._Ctype

Create a new MediaPlayer instance.

It may take as parameter either:

- a string (media URI), options… In this case, a vlc.Instance will be created.
- a vlc.Instance, a string (media URI), options…

add_slave(*i_type*, *psz_uri*, *b_select*)

Add a slave to the current media player. @note: If the player is playing, the slave will be added directly. This call will also update the slave list of the attached L{Media}. @param i_type: subtitle or audio. @param psz_uri: Uri of the slave (should contain a valid scheme). @param b_select: True if this slave should be selected when it's loaded. @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 and later. See L{media_slaves_add}.

audio_get_channel()

Get current audio channel. @return: the audio channel See libvlc_audio_output_channel_t.

audio_get_delay()

Get current audio delay. @return: the audio delay (microseconds). @version: LibVLC 1.1.1 or later.

audio_get_mute()

Get current mute status. @return: the mute status (boolean) if defined, -1 if undefined/unapplicable.

audio_get_track()

Get current audio track. @return: the audio track ID or -1 if no active input.

audio_get_track_count()

Get number of available audio tracks. @return: the number of available audio tracks (int), or -1 if unavailable.

audio_get_track_description()

Get the description of available audio tracks.

audio_get_volume()

Get current software audio volume. @return: the software volume in percents (0 = mute, 100 = nominal / 0dB).

audio_output_device_enum()
Gets a list of potential audio output devices, See L{audio_output_device_set}(). @note: Not all audio outputs support enumerating devices. The audio output may be functional even if the list is empty (None). @note: The list may not be exhaustive. @warning: Some audio output devices in the list might not actually work in some circumstances. By default, it is recommended to not specify any explicit audio device. @return: A None-terminated linked list of potential audio output devices. It must be freed with L{audio_output_device_list_release}(). @version: LibVLC 2.2.0 or later.

audio_output_device_get()
Get the current audio output device identifier. This complements L{audio_output_device_set}(). @warning: The initial value for the current audio output device identifier may not be set or may be some unknown value. A LibVLC application should compare this value against the known device identifiers (e.g. those that were previously retrieved by a call to L{audio_output_device_enum} or L{audio_output_device_list_get}) to find the current audio output device. It is possible that the selected audio output device changes (an external change) without a call to L{audio_output_device_set}. That may make this method unsuitable to use if a LibVLC application is attempting to track dynamic audio device changes as they happen. @return: the current audio output device identifier None if no device is selected or in case of error (the result must be released with free() or L{free}()). @version: LibVLC 3.0.0 or later.

audio_output_device_set(*module*, *device_id*)
Configures an explicit audio output device. If the module paramater is None, audio output will be moved to the device specified by the device identifier string immediately. This is the recommended usage. A list of adequate potential device strings can be obtained with L{audio_output_device_enum}(). However passing None is supported in LibVLC version 2.2.0 and later only; in earlier versions, this function would have no effects when the module parameter was None. If the module parameter is not None, the device parameter of the corresponding audio output, if it exists, will be set to the specified string. Note that some audio output modules do not have such a parameter (notably MMDevice and PulseAudio). A list of adequate potential device strings can be obtained with L{audio_output_device_list_get}(). @note: This function does not select the specified audio output plugin. L{audio_output_set}() is used for that purpose. @warning: The syntax for the device parameter depends on the audio output. Some audio output modules require further parameters (e.g. a channels map in the case of ALSA). @param module: If None, current audio output module. if non-None, name of audio output module. @param device_id: device identifier string. @return: Nothing. Errors are ignored (this is a design bug).

audio_output_set(*psz_name*)
Selects an audio output module. @note: Any change will take be effect only after playback is stopped and restarted. Audio output cannot be changed while playing. @param psz_name: name of audio output, use psz_name of See L{AudioOutput}. @return: 0 if function succeeded, -1 on error.

audio_set_callbacks(*play*, *pause*, *resume*, *flush*, *drain*, *opaque*)
Sets callbacks and private data for decoded audio. Use L{audio_set_format}() or L{audio_set_format_callbacks}() to configure the decoded audio format. @note: The audio callbacks override any other audio output mechanism. If the callbacks are set, LibVLC will B{not} output audio in any way. @param play: callback to play audio samples (must not be None). @param pause: callback to pause playback (or None to ignore). @param resume: callback to resume playback (or None to ignore). @param flush: callback to flush audio buffers (or None to ignore). @param drain: callback to drain audio buffers (or None to ignore). @param opaque: private pointer for the audio callbacks (as first parameter). @version: LibVLC 2.0.0 or later.

audio_set_channel(*channel*)
Set current audio channel. @param channel: the audio channel, See libvlc_audio_output_channel_t. @return: 0 on success, -1 on error.

audio_set_delay(*i_delay*)
Set current audio delay. The audio delay will be reset to zero each time the media changes. @param i_delay: the audio delay (microseconds). @return: 0 on success, -1 on error. @version: LibVLC 1.1.1 or later.

audio_set_format (*format*, *rate*, *channels*)
Sets a fixed decoded audio format. This only works in combination with L{audio_set_callbacks}(), and is mutually exclusive with L{audio_set_format_callbacks}(). @param format: a four-characters string identifying the sample format (e.g. "S16N" or "FL32"). @param rate: sample rate (expressed in Hz). @param channels: channels count. @version: LibVLC 2.0.0 or later.

audio_set_format_callbacks (*setup*, *cleanup*)
Sets decoded audio format via callbacks. This only works in combination with L{audio_set_callbacks}(). @param setup: callback to select the audio format (cannot be None). @param cleanup: callback to release any allocated resources (or None). @version: LibVLC 2.0.0 or later.

audio_set_mute (*status*)
Set mute status. @param status: If status is true then mute, otherwise unmute @warning This function does not always work. If there are no active audio playback stream, the mute status might not be available. If digital pass-through (S/PDIF, HDMI. . .) is in use, muting may be unapplicable. Also some audio output plugins do not support muting at all. @note To force silent playback, disable all audio tracks. This is more efficient and reliable than mute.

audio_set_track (*i_track*)
Set current audio track. @param i_track: the track ID (i_id field from track description). @return: 0 on success, -1 on error.

audio_set_volume (*i_volume*)
Set current software audio volume. @param i_volume: the volume in percents (0 = mute, 100 = 0dB). @return: 0 if the volume was set, -1 if it was out of range.

audio_set_volume_callback (*set_volume*)
Set callbacks and private data for decoded audio. This only works in combination with L{audio_set_callbacks}(). Use L{audio_set_format}() or L{audio_set_format_callbacks}() to configure the decoded audio format. @param set_volume: callback to apply audio volume, or None to apply volume in software. @version: LibVLC 2.0.0 or later.

audio_toggle_mute ()
Toggle mute status.

can_pause ()
Can this media player be paused? @return: true if the media player can pause libvlc_return_bool.

event_manager = functools.partial(<bound method memoize_parameterless.__call__ of Get

get_agl ()
deprecated Use L{get_nsobject}() instead.

get_chapter ()
Get movie chapter. @return: chapter number currently playing, or -1 if there is no media.

get_chapter_count ()
Get movie chapter count. @return: number of chapters in movie, or -1.

get_chapter_count_for_title (*i_title*)
Get title chapter count. @param i_title: title. @return: number of chapters in title, or -1.

get_fps ()
Get movie fps rate This function is provided for backward compatibility. It cannot deal with multiple video tracks. In LibVLC versions prior to 3.0, it would also fail if the file format did not convey the frame rate explicitly. deprecated Consider using L{media_tracks_get}() instead. @return: frames per second (fps) for this playing movie, or 0 if unspecified.

get_full_chapter_descriptions (*i_chapters_of_title*)
Get the full description of available chapters. @param i_chapters_of_title: index of the title to query for chapters (uses current title if set to -1). @return: the chapters list @version: LibVLC 3.0.0 and later.

get_full_title_descriptions()
Get the full description of available titles. @return: the titles list @version: LibVLC 3.0.0 and later.

get_fullscreen()
Get current fullscreen status. @return: the fullscreen status (boolean) libvlc_return_bool.

get_hwnd()
Get the Windows API window handle (HWND) previously set with L{set_hwnd}(). The handle will be returned even if LibVLC is not currently outputting any video to it. @return: a window handle or None if there are none.

get_instance()
Return the associated Instance.

get_length()
Get the current movie length (in ms). @return: the movie length (in ms), or -1 if there is no media.

get_media()
Get the media used by the media_player. @return: the media associated with p_mi, or None if no media is associated.

get_nsobject()
Get the NSView handler previously set with L{set_nsobject}(). @return: the NSView handler or 0 if none where set.

get_position()
Get movie position as percentage between 0.0 and 1.0. @return: movie position, or -1. in case of error.

get_rate()
Get the requested movie play rate. @warning: Depending on the underlying media, the requested rate may be different from the real playback rate. @return: movie play rate.

get_role()

Gets the media role. @return: the media player role (

ef libvlc_media_player_role_t). @version: LibVLC 3.0.0 and later.

get_state()
Get current movie state. @return: the current state of the media player (playing, paused, . . .) See libvlc_state_t.

get_time()
Get the current movie time (in ms). @return: the movie time (in ms), or -1 if there is no media.

get_title()
Get movie title. @return: title number currently playing, or -1.

get_title_count()
Get movie title count. @return: title number count, or -1.

get_xwindow()
Get the X Window System window identifier previously set with L{set_xwindow}(). Note that this will return the identifier even if VLC is not currently using it (for instance if it is playing an audio-only input). @return: an X window ID, or 0 if none where set.

has_vout()
How many video outputs does this media player have? @return: the number of video outputs.

is_playing()
is_playing. @return: 1 if the media player is playing, 0 otherwise libvlc_return_bool.

is_seekable()
Is this media player seekable? @return: true if the media player can seek libvlc_return_bool.

navigate(*navigate*)
Navigate through DVD Menu. @param navigate: the Navigation mode. @version: libVLC 2.0.0 or later.

next_chapter()
Set next chapter (if applicable).

next_frame()
Display the next frame (if supported).

pause()
Toggle pause (no effect if there is no media).

play()
Play. @return: 0 if playback started (and was already started), or -1 on error.

previous_chapter()
Set previous chapter (if applicable).

program_scrambled()
Check if the current program is scrambled. @return: true if the current program is scrambled libvlc_return_bool. @version: LibVLC 2.2.0 or later.

release()
Release a media_player after use Decrement the reference count of a media player object. If the reference count is 0, then L{release}() will release the media player object. If the media player object has been released, then it should not be used again.

retain()
Retain a reference to a media player object. Use L{release}() to decrement reference count.

set_agl(*drawable*)
deprecated Use L{set_nsobject}() instead.

set_android_context(*p_awindow_handler*)
Set the android context. @param p_awindow_handler: org.videolan.libvlc.AWindow jobject owned by the org.videolan.libvlc.MediaPlayer class from the libvlc-android project. @version: LibVLC 3.0.0 and later.

set_chapter(*i_chapter*)
Set movie chapter (if applicable). @param i_chapter: chapter number to play.

set_equalizer(*p_equalizer*)
Apply new equalizer settings to a media player. The equalizer is first created by invoking L{audio_equalizer_new}() or L{audio_equalizer_new_from_preset}(). It is possible to apply new equalizer settings to a media player whether the media player is currently playing media or not. Invoking this method will immediately apply the new equalizer settings to the audio output of the currently playing media if there is any. If there is no currently playing media, the new equalizer settings will be applied later if and when new media is played. Equalizer settings will automatically be applied to subsequently played media. To disable the equalizer for a media player invoke this method passing None for the p_equalizer parameter. The media player does not keep a reference to the supplied equalizer so it is safe for an application to release the equalizer reference any time after this method returns. @param p_equalizer: opaque equalizer handle, or None to disable the equalizer for this media player. @return: zero on success, -1 on error. @version: LibVLC 2.2.0 or later.

set_evas_object(*p_evas_object*)
Set the EFL Evas Object. @param p_evas_object: a valid EFL Evas Object (Evas_Object). @return: -1 if an error was detected, 0 otherwise. @version: LibVLC 3.0.0 and later.

set_fullscreen(*b_fullscreen*)
Enable or disable fullscreen. @warning: With most window managers, only a top-level windows can be in full-screen mode. Hence, this function will not operate properly if L{set_xwindow}() was used to embed the video in a non-top-level window. In that case, the embedding window must be reparented to the root window B{before} fullscreen mode is enabled. You will want to reparent it back to its normal parent when disabling fullscreen. @param b_fullscreen: boolean for fullscreen status.

set_hwnd(*drawable*)
Set a Win32/Win64 API window handle (HWND).

Specify where the media player should render its video output. If LibVLC was built without Win32/Win64 API output support, then this has no effects.

@param drawable: windows handle of the drawable.

set_media(*p_md*)
Set the media that will be used by the media_player. If any, previous md will be released. @param p_md: the Media. Afterwards the p_md can be safely destroyed.

set_mrl(*mrl*, **options*)
Set the MRL to play.

Warning: most audio and video options, such as text renderer, have no effects on an individual media. These options must be set at the vlc.Instance or vlc.MediaPlayer instanciation.

@param mrl: The MRL @param options: optional media option=value strings @return: the Media object

set_nsobject(*drawable*)
Set the NSView handler where the media player should render its video output. Use the vout called "macosx". The drawable is an NSObject that follow the VLCOpenGLVideoViewEmbedding protocol: @code.m @protocol VLCOpenGLVideoViewEmbedding <NSObject> - (void)addVoutSubview:(NSView *)view; - (void)removeVoutSubview:(NSView *)view; @end @endcode Or it can be an NSView object. If you want to use it along with Qt see the QMacCocoaViewContainer. Then the following code should work: @code.mm

> NSView *video = [[NSView alloc] init]; QMacCocoaViewContainer *container = new QMacCocoaViewContainer(video, parent); L{set_nsobject}(mp, video); [video release];

@endcode You can find a live example in VLCVideoView in VLCKit.framework. @param drawable: the drawable that is either an NSView or an object following the VLCOpenGLVideoViewEmbedding protocol.

set_pause(*do_pause*)
Pause or resume (no effect if there is no media). @param do_pause: play/resume if zero, pause if non-zero. @version: LibVLC 1.1.1 or later.

set_position(*f_pos*)
Set movie position as percentage between 0.0 and 1.0. This has no effect if playback is not enabled. This might not work depending on the underlying input format and protocol. @param f_pos: the position.

set_rate(*rate*)
Set movie play rate. @param rate: movie play rate to set. @return: -1 if an error was detected, 0 otherwise (but even then, it might not actually work depending on the underlying media protocol).

set_renderer(*p_item*)
Set a renderer to the media player @note: must be called before the first call of L{play}() to take effect. See L{renderer_discoverer_new}. @param p_item: an item discovered by L{renderer_discoverer_start}(). @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 or later.

set_role(*role*)

Sets the media role. @param role: the media player role (

ef libvlc_media_player_role_t). @return: 0 on success, -1 on error.

`set_time`(*i_time*)
Set the movie time (in ms). This has no effect if no media is being played. Not all formats and protocols support this. @param i_time: the movie time (in ms).

`set_title`(*i_title*)
Set movie title. @param i_title: title number to play.

`set_video_title_display`(*position*, *timeout*)
Set if, and how, the video title will be shown when media is played. @param position: position at which to display the title, or libvlc_position_disable to prevent the title from being displayed. @param timeout: title display timeout in milliseconds (ignored if libvlc_position_disable). @version: libVLC 2.1.0 or later.

`set_xwindow`(*drawable*)
Set an X Window System drawable where the media player should render its video output. The call takes effect when the playback starts. If it is already started, it might need to be stopped before changes apply. If LibVLC was built without X11 output support, then this function has no effects. By default, LibVLC will capture input events on the video rendering area. Use L{video_set_mouse_input}() and L{video_set_key_input}() to disable that and deliver events to the parent window / to the application instead. By design, the X11 protocol delivers input events to only one recipient. @warning The application must call the XInitThreads() function from Xlib before L{new}(), and before any call to XOpenDisplay() directly or via any other library. Failure to call XInitThreads() will seriously impede LibVLC performance. Calling XOpenDisplay() before XInitThreads() will eventually crash the process. That is a limitation of Xlib. @param drawable: X11 window ID @note The specified identifier must correspond to an existing Input/Output class X11 window. Pixmaps are B{not} currently supported. The default X11 server is assumed, i.e. that specified in the DISPLAY environment variable. @warning LibVLC can deal with invalid X11 handle errors, however some display drivers (EGL, GLX, VA and/or VDPAU) can unfortunately not. Thus the window handle must remain valid until playback is stopped, otherwise the process may abort or crash. @bug No more than one window handle per media player instance can be specified. If the media has multiple simultaneously active video tracks, extra tracks will be rendered into external windows beyond the control of the application.

`stop`()
Stop (no effect if there is no media).

`toggle_fullscreen`()
Toggle fullscreen status on non-embedded video outputs. @warning: The same limitations applies to this function as to L{set_fullscreen}().

`toggle_teletext`()
Toggle teletext transparent status on video output. deprecated use L{video_set_teletext}() instead.

`video_get_adjust_float`(*option*)
Get float adjust option. @param option: adjust option to get, values of libvlc_video_adjust_option_t. @version: LibVLC 1.1.1 and later.

`video_get_adjust_int`(*option*)
Get integer adjust option. @param option: adjust option to get, values of libvlc_video_adjust_option_t. @version: LibVLC 1.1.1 and later.

`video_get_aspect_ratio`()
Get current video aspect ratio. @return: the video aspect ratio or None if unspecified (the result must be released with free() or L{free}()).

`video_get_chapter_description`(*title*)
Get the description of available chapters for specific title.

@param title: selected title (int)

video_get_crop_geometry()
Get current crop filter geometry. @return: the crop filter geometry or None if unset.

video_get_cursor(*num=0*)
Get the mouse pointer coordinates over a video as 2-tuple (x, y).

Coordinates are expressed in terms of the decoded video resolution, B{not} in terms of pixels on the screen/viewport. To get the latter, you must query your windowing system directly.

Either coordinate may be negative or larger than the corresponding size of the video, if the cursor is outside the rendering area.

@warning: The coordinates may be out-of-date if the pointer is not located on the video rendering area. LibVLC does not track the mouse pointer if the latter is outside the video widget.

@note: LibVLC does not support multiple mouse pointers (but does support multiple input devices sharing the same pointer).

@param num: video number (default 0).

video_get_height(*num=0*)
Get the height of a video in pixels.

@param num: video number (default 0).

video_get_logo_int(*option*)
Get integer logo option. @param option: logo option to get, values of libvlc_video_logo_option_t.

video_get_marquee_int(*option*)
Get an integer marquee option value. @param option: marq option to get See libvlc_video_marquee_int_option_t.

video_get_marquee_string(*option*)
Get a string marquee option value. @param option: marq option to get See libvlc_video_marquee_string_option_t.

video_get_scale()
Get the current video scaling factor. See also L{video_set_scale}(). @return: the currently configured zoom factor, or 0. if the video is set to fit to the output window/drawable automatically.

video_get_size(*num=0*)
Get the video size in pixels as 2-tuple (width, height).

@param num: video number (default 0).

video_get_spu()
Get current video subtitle. @return: the video subtitle selected, or -1 if none.

video_get_spu_count()
Get the number of available video subtitles. @return: the number of available video subtitles.

video_get_spu_delay()
Get the current subtitle delay. Positive values means subtitles are being displayed later, negative values earlier. @return: time (in microseconds) the display of subtitles is being delayed. @version: LibVLC 2.0.0 or later.

video_get_spu_description()
Get the description of available video subtitles.

video_get_teletext()
Get current teletext page requested or 0 if it's disabled. Teletext is disabled by default, call L{video_set_teletext}() to enable it. @return: the current teletext page requested.

`video_get_title_description``()`
Get the description of available titles.

`video_get_track``()`
Get current video track. @return: the video track ID (int) or -1 if no active input.

`video_get_track_count``()`
Get number of available video tracks. @return: the number of available video tracks (int).

`video_get_track_description``()`
Get the description of available video tracks.

`video_get_width``(`*num=0*`)`
Get the width of a video in pixels.

@param num: video number (default 0).

`video_set_adjust_float``(`*option*, *value*`)`
Set adjust option as float. Options that take a different type value are ignored. @param option: adust option to set, values of libvlc_video_adjust_option_t. @param value: adjust option value. @version: LibVLC 1.1.1 and later.

`video_set_adjust_int``(`*option*, *value*`)`
Set adjust option as integer. Options that take a different type value are ignored. Passing libvlc_adjust_enable as option value has the side effect of starting (arg !0) or stopping (arg 0) the adjust filter. @param option: adust option to set, values of libvlc_video_adjust_option_t. @param value: adjust option value. @version: LibVLC 1.1.1 and later.

`video_set_aspect_ratio``(`*psz_aspect*`)`
Set new video aspect ratio. @param psz_aspect: new video aspect-ratio or None to reset to default @note Invalid aspect ratios are ignored.

`video_set_callbacks``(`*lock*, *unlock*, *display*, *opaque*`)`
Set callbacks and private data to render decoded video to a custom area in memory. Use L{video_set_format}() or L{video_set_format_callbacks}() to configure the decoded format. @warning: Rendering video into custom memory buffers is considerably less efficient than rendering in a custom window as normal. For optimal perfomances, VLC media player renders into a custom window, and does not use this function and associated callbacks. It is B{highly recommended} that other LibVLC-based application do likewise. To embed video in a window, use libvlc_media_player_set_xid() or equivalent depending on the operating system. If window embedding does not fit the application use case, then a custom LibVLC video output display plugin is required to maintain optimal video rendering performances. The following limitations affect performance: - Hardware video decoding acceleration will either be disabled completely,

> or require (relatively slow) copy from video/DSP memory to main memory.

- Sub-pictures (subtitles, on-screen display, etc.) must be blent into the main picture by the CPU instead of the GPU.
- Depending on the video format, pixel format conversion, picture scaling, cropping and/or picture re-orientation, must be performed by the CPU instead of the GPU.
- Memory copying is required between LibVLC reference picture buffers and application buffers (between lock and unlock callbacks).

@param lock: callback to lock video memory (must not be None). @param unlock: callback to unlock video memory (or None if not needed). @param display: callback to display video (or None if not needed). @param opaque: private pointer for the three callbacks (as first parameter). @version: LibVLC 1.1.1 or later.

video_set_crop_geometry (*psz_geometry*)
Set new crop filter geometry. @param psz_geometry: new crop filter geometry (None to unset).

video_set_deinterlace (*psz_mode*)
Enable or disable deinterlace filter. @param psz_mode: type of deinterlace filter, None to disable.

video_set_format (*chroma*, *width*, *height*, *pitch*)
Set decoded video chroma and dimensions. This only works in combination with L{video_set_callbacks}(), and is mutually exclusive with L{video_set_format_callbacks}(). @param chroma: a four-characters string identifying the chroma (e.g. "RV32" or "YUYV"). @param width: pixel width. @param height: pixel height. @param pitch: line pitch (in bytes). @version: LibVLC 1.1.1 or later. @bug: All pixel planes are expected to have the same pitch. To use the YCbCr color space with chrominance subsampling, consider using L{video_set_format_callbacks}() instead.

video_set_format_callbacks (*setup*, *cleanup*)
Set decoded video chroma and dimensions. This only works in combination with L{video_set_callbacks}(). @param setup: callback to select the video format (cannot be None). @param cleanup: callback to release any allocated resources (or None). @version: LibVLC 2.0.0 or later.

video_set_key_input (*on*)
Enable or disable key press events handling, according to the LibVLC hotkeys configuration. By default and for historical reasons, keyboard events are handled by the LibVLC video widget. @note: On X11, there can be only one subscriber for key press and mouse click events per window. If your application has subscribed to those events for the X window ID of the video widget, then LibVLC will not be able to handle key presses and mouse clicks in any case. @warning: This function is only implemented for X11 and Win32 at the moment. @param on: true to handle key press events, false to ignore them.

video_set_logo_int (*option*, *value*)
Set logo option as integer. Options that take a different type value are ignored. Passing libvlc_logo_enable as option value has the side effect of starting (arg !0) or stopping (arg 0) the logo filter. @param option: logo option to set, values of libvlc_video_logo_option_t. @param value: logo option value.

video_set_logo_string (*option*, *psz_value*)
Set logo option as string. Options that take a different type value are ignored. @param option: logo option to set, values of libvlc_video_logo_option_t. @param psz_value: logo option value.

video_set_marquee_int (*option*, *i_val*)
Enable, disable or set an integer marquee option Setting libvlc_marquee_Enable has the side effect of enabling (arg !0) or disabling (arg 0) the marq filter. @param option: marq option to set See libvlc_video_marquee_int_option_t. @param i_val: marq option value.

video_set_marquee_string (*option*, *psz_text*)
Set a marquee string option. @param option: marq option to set See libvlc_video_marquee_string_option_t. @param psz_text: marq option value.

video_set_mouse_input (*on*)
Enable or disable mouse click events handling. By default, those events are handled. This is needed for DVD menus to work, as well as a few video filters such as "puzzle". See L{video_set_key_input}(). @warning: This function is only implemented for X11 and Win32 at the moment. @param on: true to handle mouse click events, false to ignore them.

video_set_scale (*f_factor*)
Set the video scaling factor. That is the ratio of the number of pixels on screen to the number of pixels in the original decoded video in each dimension. Zero is a special value; it will adjust the video to the output window/drawable (in windowed mode) or the entire screen. Note that not all video outputs support scaling. @param f_factor: the scaling factor, or zero.

video_set_spu (*i_spu*)
Set new video subtitle. @param i_spu: video subtitle track to select (i_id from track description). @return:

0 on success, -1 if out of range.

video_set_spu_delay(*i_delay*)
Set the subtitle delay. This affects the timing of when the subtitle will be displayed. Positive values result in subtitles being displayed later, while negative values will result in subtitles being displayed earlier. The subtitle delay will be reset to zero each time the media changes. @param i_delay: time (in microseconds) the display of subtitles should be delayed. @return: 0 on success, -1 on error. @version: LibVLC 2.0.0 or later.

video_set_subtitle_file(*psz_subtitle*)
Set new video subtitle file. deprecated Use L{add_slave}() instead. @param psz_subtitle: new video subtitle file. @return: the success status (boolean).

video_set_teletext(*i_page*)

Set new teletext page to retrieve. This function can also be used to send a teletext key. @param i_page:
teletex page number requested. This value can be 0 to disable teletext, a number in the range]0;1000[to show the requested page, or a

ef libvlc_teletext_key_t. 100 is the default teletext page.

video_set_track(*i_track*)
Set video track. @param i_track: the track ID (i_id field from track description). @return: 0 on success, -1 if out of range.

video_take_snapshot(*num*, *psz_filepath*, *i_width*, *i_height*)
Take a snapshot of the current video window. If i_width AND i_height is 0, original size is used. If i_width XOR i_height is 0, original aspect-ratio is preserved. @param num: number of video output (typically 0 for the first/only one). @param psz_filepath: the path of a file or a folder to save the screenshot into. @param i_width: the snapshot's width. @param i_height: the snapshot's height. @return: 0 on success, -1 if the video was not found.

video_update_viewpoint(*p_viewpoint*, *b_absolute*)
Update the video viewpoint information. @note: It is safe to call this function before the media player is started. @param p_viewpoint: video viewpoint allocated via L{video_new_viewpoint}(). @param b_absolute: if true replace the old viewpoint with the new one. If false, increase/decrease it. @return: -1 in case of error, 0 otherwise @note the values are set asynchronously, it will be used by the next frame displayed. @version: LibVLC 3.0.0 and later.

will_play()
Is the player able to play. @return: boolean libvlc_return_bool.

class pyparrot.utils.vlc.**MediaPlayerRole**
Bases: `pyparrot.utils.vlc._Enum`

Media player roles.

ersion libvlc 3.0.0 and later. see ef libvlc_media_player_set_role().

Accessibility = pyparrot.utils.vlc.MediaPlayerRole.Accessibility

Animation = pyparrot.utils.vlc.MediaPlayerRole.Animation

Communication = pyparrot.utils.vlc.MediaPlayerRole.Communication

Game = pyparrot.utils.vlc.MediaPlayerRole.Game

Music = pyparrot.utils.vlc.MediaPlayerRole.Music

Notification = pyparrot.utils.vlc.MediaPlayerRole.Notification

Production = pyparrot.utils.vlc.MediaPlayerRole.Production

Test = pyparrot.utils.vlc.MediaPlayerRole.Test

Video = pyparrot.utils.vlc.MediaPlayerRole.Video

class pyparrot.utils.vlc.**MediaReadCb**
Bases: ctypes.c_void_p

Callback prototype to read data from a custom bitstream input media. @param opaque: private pointer as set by the @ref libvlc_media_open_cb callback. @param buf: start address of the buffer to read data into. @param len: bytes length of the buffer. @return: strictly positive number of bytes read, 0 on end-of-stream, or -1 on non-recoverable error @note If no data is immediately available, then the callback should sleep. @warning The application is responsible for avoiding deadlock situations. In particular, the callback should return an error if playback is stopped; if it does not return, then L{libvlc_media_player_stop}() will never return.

class pyparrot.utils.vlc.**MediaSeekCb**
Bases: ctypes.c_void_p

Callback prototype to seek a custom bitstream input media. @param opaque: private pointer as set by the @ref libvlc_media_open_cb callback. @param offset: absolute byte offset to seek to. @return: 0 on success, -1 on error.

class pyparrot.utils.vlc.**MediaSlave**
Bases: pyparrot.utils.vlc._Cstruct

class pyparrot.utils.vlc.**MediaSlaveType**
Bases: pyparrot.utils.vlc._Enum

Type of a media slave: subtitle or audio.

audio = pyparrot.utils.vlc.MediaSlaveType.audio

subtitle = pyparrot.utils.vlc.MediaSlaveType.subtitle

class pyparrot.utils.vlc.**MediaStats**
Bases: pyparrot.utils.vlc._Cstruct

decoded_audio
Structure/Union member

decoded_video
Structure/Union member

demux_bitrate
Structure/Union member

demux_corrupted
Structure/Union member

demux_discontinuity
Structure/Union member

demux_read_bytes
Structure/Union member

displayed_pictures
Structure/Union member

input_bitrate
Structure/Union member

lost_abuffers
Structure/Union member

lost_pictures
Structure/Union member

played_abuffers
Structure/Union member

read_bytes
Structure/Union member

send_bitrate
Structure/Union member

sent_bytes
Structure/Union member

sent_packets
Structure/Union member

class `pyparrot.utils.vlc.`**MediaTrack**
Bases: `pyparrot.utils.vlc._Cstruct`

audio
Structure/Union member

bitrate
Structure/Union member

codec
Structure/Union member

description
Structure/Union member

id
Structure/Union member

language
Structure/Union member

level
Structure/Union member

original_fourcc
Structure/Union member

profile
Structure/Union member

subtitle
Structure/Union member

type
Structure/Union member

u
Structure/Union member

video
Structure/Union member

class `pyparrot.utils.vlc.`**MediaTrackInfo**
Bases: `pyparrot.utils.vlc._Cstruct`

channels_or_height
Structure/Union member

codec
Structure/Union member

id
Structure/Union member

level
Structure/Union member

profile
Structure/Union member

rate_or_width
Structure/Union member

type
Structure/Union member

class pyparrot.utils.vlc.**MediaTrackTracks**
Bases: `_ctypes.Union`

audio
Structure/Union member

subtitle
Structure/Union member

video
Structure/Union member

class pyparrot.utils.vlc.**MediaType**
Bases: `pyparrot.utils.vlc._Enum`

Media type See libvlc_media_get_type.

directory = pyparrot.utils.vlc.MediaType.directory

disc = pyparrot.utils.vlc.MediaType.disc

file = pyparrot.utils.vlc.MediaType.file

playlist = pyparrot.utils.vlc.MediaType.playlist

stream = pyparrot.utils.vlc.MediaType.stream

unknown = pyparrot.utils.vlc.MediaType.unknown

class pyparrot.utils.vlc.**Meta**
Bases: `pyparrot.utils.vlc._Enum`

Meta data types.

Actors = pyparrot.utils.vlc.Meta.Actors

Album = pyparrot.utils.vlc.Meta.Album

AlbumArtist = pyparrot.utils.vlc.Meta.AlbumArtist

Artist = pyparrot.utils.vlc.Meta.Artist

ArtworkURL = pyparrot.utils.vlc.Meta.ArtworkURL

Copyright = pyparrot.utils.vlc.Meta.Copyright

Date = pyparrot.utils.vlc.Meta.Date

Description = pyparrot.utils.vlc.Meta.Description

Director = pyparrot.utils.vlc.Meta.Director

DiscNumber = pyparrot.utils.vlc.Meta.DiscNumber

DiscTotal = pyparrot.utils.vlc.Meta.DiscTotal

EncodedBy = pyparrot.utils.vlc.Meta.EncodedBy

Episode = pyparrot.utils.vlc.Meta.Episode

Genre = pyparrot.utils.vlc.Meta.Genre

Language = pyparrot.utils.vlc.Meta.Language

NowPlaying = pyparrot.utils.vlc.Meta.NowPlaying

Publisher = pyparrot.utils.vlc.Meta.Publisher

Rating = pyparrot.utils.vlc.Meta.Rating

Season = pyparrot.utils.vlc.Meta.Season

Setting = pyparrot.utils.vlc.Meta.Setting

ShowName = pyparrot.utils.vlc.Meta.ShowName

Title = pyparrot.utils.vlc.Meta.Title

TrackID = pyparrot.utils.vlc.Meta.TrackID

TrackNumber = pyparrot.utils.vlc.Meta.TrackNumber

TrackTotal = pyparrot.utils.vlc.Meta.TrackTotal

URL = pyparrot.utils.vlc.Meta.URL

class `pyparrot.utils.vlc.`**ModuleDescription**

Bases: `pyparrot.utils.vlc._Cstruct`

help

Structure/Union member

longname

Structure/Union member

name

Structure/Union member

next

Structure/Union member

shortname

Structure/Union member

class `pyparrot.utils.vlc.`**NavigateMode**

Bases: `pyparrot.utils.vlc._Enum`

Navigation mode.

activate = pyparrot.utils.vlc.NavigateMode.activate

down = pyparrot.utils.vlc.NavigateMode.down

left = pyparrot.utils.vlc.NavigateMode.left

popup = pyparrot.utils.vlc.NavigateMode.popup

right = pyparrot.utils.vlc.NavigateMode.right

up = pyparrot.utils.vlc.NavigateMode.up

class pyparrot.utils.vlc.**PlaybackMode**
Bases: `pyparrot.utils.vlc._Enum`

Defines playback modes for playlist.

default = pyparrot.utils.vlc.PlaybackMode.default

loop = pyparrot.utils.vlc.PlaybackMode.loop

repeat = pyparrot.utils.vlc.PlaybackMode.repeat

class pyparrot.utils.vlc.**PlaylistItem**
Bases: `pyparrot.utils.vlc._Cstruct`

id
Structure/Union member

name
Structure/Union member

uri
Structure/Union member

class pyparrot.utils.vlc.**Position**(**unused*)
Bases: `object`

Enum-like, immutable window position constants.

See e.g. VideoMarqueeOption.Position.

Bottom = 8

BottomCenter = 8

BottomLeft = 9

BottomRight = 10

Center = 0

CenterLeft = 1

CenterRight = 2

Left = 1

Right = 2

Top = 4

TopCenter = 4

TopLeft = 5

TopRight = 6

class pyparrot.utils.vlc.**RDDescription**
Bases: `pyparrot.utils.vlc._Cstruct`

class pyparrot.utils.vlc.**Rectangle**
Bases: `pyparrot.utils.vlc._Cstruct`

bottom
Structure/Union member

left
Structure/Union member

right
Structure/Union member

top
Structure/Union member

class `pyparrot.utils.vlc.`**State**
Bases: `pyparrot.utils.vlc._Enum`

Note the order of libvlc_state_t enum must match exactly the order of See mediacontrol_playerstatus, See input_state_e enums, and videolan.libvlc.state (at bindings/cil/src/media.cs). expected states by web plugins are: idle/close=0, opening=1, playing=3, paused=4, stopping=5, ended=6, error=7.

Buffering = pyparrot.utils.vlc.State.Buffering

Ended = pyparrot.utils.vlc.State.Ended

Error = pyparrot.utils.vlc.State.Error

NothingSpecial = pyparrot.utils.vlc.State.NothingSpecial

Opening = pyparrot.utils.vlc.State.Opening

Paused = pyparrot.utils.vlc.State.Paused

Playing = pyparrot.utils.vlc.State.Playing

Stopped = pyparrot.utils.vlc.State.Stopped

class `pyparrot.utils.vlc.`**SubtitleTrack**
Bases: `pyparrot.utils.vlc._Cstruct`

encoding
Structure/Union member

class `pyparrot.utils.vlc.`**TeletextKey**
Bases: `pyparrot.utils.vlc._Enum`

Enumeration of teletext keys than can be passed via libvlc_video_set_teletext().

blue = pyparrot.utils.vlc.TeletextKey.blue

green = pyparrot.utils.vlc.TeletextKey.green

index = pyparrot.utils.vlc.TeletextKey.index

red = pyparrot.utils.vlc.TeletextKey.red

yellow = pyparrot.utils.vlc.TeletextKey.yellow

class `pyparrot.utils.vlc.`**TitleDescription**
Bases: `pyparrot.utils.vlc._Cstruct`

class `pyparrot.utils.vlc.`**TrackDescription**
Bases: `pyparrot.utils.vlc._Cstruct`

id
Structure/Union member

name
Structure/Union member

next
Structure/Union member

class pyparrot.utils.vlc.**TrackType**
Bases: pyparrot.utils.vlc._Enum

N/A

audio = pyparrot.utils.vlc.TrackType.audio

text = pyparrot.utils.vlc.TrackType.text

unknown = pyparrot.utils.vlc.TrackType.FIXME_(4294967295)

video = pyparrot.utils.vlc.TrackType.video

exception pyparrot.utils.vlc.**VLCException**
Bases: Exception

Exception raised by libvlc methods.

class pyparrot.utils.vlc.**VideoAdjustOption**
Bases: pyparrot.utils.vlc._Enum

Option values for libvlc_video_{get,set}_adjust_{int,float,bool}.

Brightness = pyparrot.utils.vlc.VideoAdjustOption.Brightness

Contrast = pyparrot.utils.vlc.VideoAdjustOption.Contrast

Enable = pyparrot.utils.vlc.VideoAdjustOption.Enable

Gamma = pyparrot.utils.vlc.VideoAdjustOption.Gamma

Hue = pyparrot.utils.vlc.VideoAdjustOption.Hue

Saturation = pyparrot.utils.vlc.VideoAdjustOption.Saturation

class pyparrot.utils.vlc.**VideoCleanupCb**
Bases: ctypes.c_void_p

Callback prototype to configure picture buffers format. @param opaque: private pointer as passed to L{libvlc_video_set_callbacks}() (and possibly modified by @ref libvlc_video_format_cb) [IN].

class pyparrot.utils.vlc.**VideoDisplayCb**
Bases: ctypes.c_void_p

Callback prototype to display a picture. When the video frame needs to be shown, as determined by the media playback clock, the display callback is invoked. @param opaque: private pointer as passed to L{libvlc_video_set_callbacks}() [IN]. @param picture: private pointer returned from the @ref libvlc_video_lock_cb callback [IN].

class pyparrot.utils.vlc.**VideoFormatCb**
Bases: ctypes.c_void_p

Callback prototype to configure picture buffers format. This callback gets the format of the video as output by the video decoder and the chain of video filters (if any). It can opt to change any parameter as it needs. In that case, LibVLC will attempt to convert the video format (rescaling and chroma conversion) but these operations can be CPU intensive. @param opaque: pointer to the private pointer passed to L{libvlc_video_set_callbacks}() [IN/OUT]. @param chroma: pointer to the 4 bytes video format identifier [IN/OUT]. @param width: pointer to the pixel width [IN/OUT]. @param height: pointer to the pixel height [IN/OUT]. @param pitches: table of scanline pitches in bytes for each pixel plane (the table is allocated by LibVLC) [OUT]. @return: lines table of scanlines count for each plane.

class pyparrot.utils.vlc.**VideoLockCb**
Bases: ctypes.c_void_p

Callback prototype to allocate and lock a picture buffer. Whenever a new video frame needs to be decoded, the lock callback is invoked. Depending on the video chroma, one or three pixel planes of adequate dimensions must be returned via the second parameter. Those planes must be aligned on 32-bytes boundaries. @param opaque: private pointer as passed to L{libvlc_video_set_callbacks}() [IN]. @param planes: start address of the pixel planes (LibVLC allocates the array of void pointers, this callback must initialize the array) [OUT]. @return: a private pointer for the display and unlock callbacks to identify the picture buffers.

class `pyparrot.utils.vlc.`**`VideoLogoOption`**

Bases: `pyparrot.utils.vlc._Enum`

Option values for libvlc_video_{get,set}_logo_{int,string}.

`delay = pyparrot.utils.vlc.VideoLogoOption.delay`

`enable = pyparrot.utils.vlc.VideoLogoOption.enable`

`file = pyparrot.utils.vlc.VideoLogoOption.file`

`logo_x = pyparrot.utils.vlc.VideoLogoOption.logo_x`

`logo_y = pyparrot.utils.vlc.VideoLogoOption.logo_y`

`opacity = pyparrot.utils.vlc.VideoLogoOption.opacity`

`position = pyparrot.utils.vlc.VideoLogoOption.position`

`repeat = pyparrot.utils.vlc.VideoLogoOption.repeat`

class `pyparrot.utils.vlc.`**`VideoMarqueeOption`**

Bases: `pyparrot.utils.vlc._Enum`

Marq options definition.

`Color = pyparrot.utils.vlc.VideoMarqueeOption.Color`

`Enable = pyparrot.utils.vlc.VideoMarqueeOption.Enable`

`Opacity = pyparrot.utils.vlc.VideoMarqueeOption.Opacity`

`Position = pyparrot.utils.vlc.VideoMarqueeOption.Position`

`Refresh = pyparrot.utils.vlc.VideoMarqueeOption.Refresh`

`Size = pyparrot.utils.vlc.VideoMarqueeOption.Size`

`Text = pyparrot.utils.vlc.VideoMarqueeOption.Text`

`Timeout = pyparrot.utils.vlc.VideoMarqueeOption.Timeout`

`marquee_X = pyparrot.utils.vlc.VideoMarqueeOption.marquee_X`

`marquee_Y = pyparrot.utils.vlc.VideoMarqueeOption.marquee_Y`

class `pyparrot.utils.vlc.`**`VideoOrient`**

Bases: `pyparrot.utils.vlc._Enum`

N/A

`bottom = pyparrot.utils.vlc.VideoOrient.bottom`

`left = pyparrot.utils.vlc.VideoOrient.left`

`right = pyparrot.utils.vlc.VideoOrient.right`

`top = pyparrot.utils.vlc.VideoOrient.top`

class pyparrot.utils.vlc.**VideoProjection**

Bases: pyparrot.utils.vlc._Enum

N/A

equirectangular = pyparrot.utils.vlc.VideoProjection.equirectangular

rectangular = pyparrot.utils.vlc.VideoProjection.rectangular

standard = pyparrot.utils.vlc.VideoProjection.standard

class pyparrot.utils.vlc.**VideoTrack**

Bases: pyparrot.utils.vlc._Cstruct

frame_rate_den
Structure/Union member

frame_rate_num
Structure/Union member

height
Structure/Union member

sar_den
Structure/Union member

sar_num
Structure/Union member

width
Structure/Union member

class pyparrot.utils.vlc.**VideoUnlockCb**

Bases: ctypes.c_void_p

Callback prototype to unlock a picture buffer. When the video frame decoding is complete, the unlock callback is invoked. This callback might not be needed at all. It is only an indication that the application can now read the pixel values if it needs to. @note: A picture buffer is unlocked after the picture is decoded, but before the picture is displayed. @param opaque: private pointer as passed to L{libvlc_video_set_callbacks}() [IN]. @param picture: private pointer returned from the @ref libvlc_video_lock_cb callback [IN]. @param planes: pixel planes as defined by the @ref libvlc_video_lock_cb callback (this parameter is only for convenience) [IN].

class pyparrot.utils.vlc.**VideoViewpoint**

Bases: pyparrot.utils.vlc._Cstruct

pyparrot.utils.vlc.**bytes_to_str**(*b*)
Translate bytes to string.

pyparrot.utils.vlc.**callbackmethod**(*callback*)
Now obsolete @callbackmethod decorator.

pyparrot.utils.vlc.**cb**
alias of *pyparrot.utils.vlc.CallbackDecorators*

pyparrot.utils.vlc.**class_result**(*classname*)
Errcheck function. Returns a function that creates the specified class.

pyparrot.utils.vlc.**debug_callback**(*event*, **args*, ***kwds*)
Example callback, useful for debugging.

pyparrot.utils.vlc.**find_lib**()

pyparrot.utils.vlc.**get_default_instance**()
Return the default VLC.Instance.

pyparrot.utils.vlc.**hex_version**()
Return the version of these bindings in hex or 0 if unavailable.

pyparrot.utils.vlc.**libvlc_add_intf**(*p_instance*, *name*)
Try to start a user interface for the libvlc instance. @param p_instance: the instance. @param name: interface name, or None for default. @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_amp_at_index**(*p_equalizer*, *u_band*)
Get the amplification value for a particular equalizer frequency band. @param p_equalizer: valid equalizer handle, must not be None. @param u_band: index, counting from zero, of the frequency band to get. @return: amplification value (Hz); NaN if there is no such frequency band. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_band_count**()
Get the number of distinct frequency bands for an equalizer. @return: number of frequency bands. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_band_frequency**(*u_index*)
Get a particular equalizer band frequency. This value can be used, for example, to create a label for an equalizer band control in a user interface. @param u_index: index of the band, counting from zero. @return: equalizer band frequency (Hz), or -1 if there is no such band. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_preamp**(*p_equalizer*)
Get the current pre-amplification value from an equalizer. @param p_equalizer: valid equalizer handle, must not be None. @return: preamp value (Hz). @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_preset_count**()
Get the number of equalizer presets. @return: number of presets. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_get_preset_name**(*u_index*)
Get the name of a particular equalizer preset. This name can be used, for example, to prepare a preset label or menu in a user interface. @param u_index: index of the preset, counting from zero. @return: preset name, or None if there is no such preset. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_new**()
Create a new default equalizer, with all frequency values zeroed. The new equalizer can subsequently be applied to a media player by invoking L{libvlc_media_player_set_equalizer}(). The returned handle should be freed via L{libvlc_audio_equalizer_release}() when it is no longer needed. @return: opaque equalizer handle, or None on error. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_new_from_preset**(*u_index*)
Create a new equalizer, with initial frequency values copied from an existing preset. The new equalizer can subsequently be applied to a media player by invoking L{libvlc_media_player_set_equalizer}(). The returned handle should be freed via L{libvlc_audio_equalizer_release}() when it is no longer needed. @param u_index: index of the preset, counting from zero. @return: opaque equalizer handle, or None on error. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_release**(*p_equalizer*)
Release a previously created equalizer instance. The equalizer was previously created by using L{libvlc_audio_equalizer_new}() or L{libvlc_audio_equalizer_new_from_preset}(). It is safe to invoke this method with a None p_equalizer parameter for no effect. @param p_equalizer: opaque equalizer handle, or None. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_audio_equalizer_set_amp_at_index**(*p_equalizer*, *f_amp*, *u_band*)
Set a new amplification value for a particular equalizer frequency band. The new equalizer settings are subsequently applied to a media player by invoking L{libvlc_media_player_set_equalizer}(). The supplied amplification value will be clamped to the -20.0 to +20.0 range. @param p_equalizer: valid equalizer handle, must not be None. @param f_amp: amplification value (-20.0 to 20.0 Hz). @param u_band: index, counting from zero, of the frequency band to set. @return: zero on success, -1 on error. @version: LibVLC 2.2.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_equalizer_set_preamp`**(*p_equalizer*, *f_preamp*)
Set a new pre-amplification value for an equalizer. The new equalizer settings are subsequently applied to a media player by invoking L{libvlc_media_player_set_equalizer}(). The supplied amplification value will be clamped to the -20.0 to +20.0 range. @param p_equalizer: valid equalizer handle, must not be None. @param f_preamp: preamp value (-20.0 to 20.0 Hz). @return: zero on success, -1 on error. @version: LibVLC 2.2.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_filter_list_get`**(*p_instance*)
Returns a list of audio filters that are available. @param p_instance: libvlc instance. @return: a list of module descriptions. It should be freed with L{libvlc_module_description_list_release}(). In case of an error, None is returned. See L{ModuleDescription} See L{libvlc_module_description_list_release}.

`pyparrot.utils.vlc.`**`libvlc_audio_get_channel`**(*p_mi*)
Get current audio channel. @param p_mi: media player. @return: the audio channel See libvlc_audio_output_channel_t.

`pyparrot.utils.vlc.`**`libvlc_audio_get_delay`**(*p_mi*)
Get current audio delay. @param p_mi: media player. @return: the audio delay (microseconds). @version: LibVLC 1.1.1 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_get_mute`**(*p_mi*)
Get current mute status. @param p_mi: media player. @return: the mute status (boolean) if defined, -1 if undefined/unapplicable.

`pyparrot.utils.vlc.`**`libvlc_audio_get_track`**(*p_mi*)
Get current audio track. @param p_mi: media player. @return: the audio track ID or -1 if no active input.

`pyparrot.utils.vlc.`**`libvlc_audio_get_track_count`**(*p_mi*)
Get number of available audio tracks. @param p_mi: media player. @return: the number of available audio tracks (int), or -1 if unavailable.

`pyparrot.utils.vlc.`**`libvlc_audio_get_track_description`**(*p_mi*)
Get the description of available audio tracks. @param p_mi: media player. @return: list with description of available audio tracks, or None. It must be freed with L{libvlc_track_description_list_release}().

`pyparrot.utils.vlc.`**`libvlc_audio_get_volume`**(*p_mi*)
Get current software audio volume. @param p_mi: media player. @return: the software volume in percents (0 = mute, 100 = nominal / 0dB).

`pyparrot.utils.vlc.`**`libvlc_audio_output_device_count`**(*p_instance*, *psz_audio_output*)
Backward compatibility stub. Do not use in new code. deprecated Use L{libvlc_audio_output_device_list_get}() instead. @return: always 0.

`pyparrot.utils.vlc.`**`libvlc_audio_output_device_enum`**(*mp*)
Gets a list of potential audio output devices, See L{libvlc_audio_output_device_set}(). @note: Not all audio outputs support enumerating devices. The audio output may be functional even if the list is empty (None). @note: The list may not be exhaustive. @warning: Some audio output devices in the list might not actually work in some circumstances. By default, it is recommended to not specify any explicit audio device. @param mp: media player. @return: A None-terminated linked list of potential audio output devices. It must be freed with L{libvlc_audio_output_device_list_release}(). @version: LibVLC 2.2.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_output_device_get`**(*mp*)
Get the current audio output device identifier. This complements L{libvlc_audio_output_device_set}(). @warning: The initial value for the current audio output device identifier may not be set or may be some unknown value. A LibVLC application should compare this value against the known device identifiers (e.g. those that were previously retrieved by a call to L{libvlc_audio_output_device_enum} or L{libvlc_audio_output_device_list_get}) to find the current audio output device. It is possible that the selected audio output device changes (an external change) without a call to L{libvlc_audio_output_device_set}. That may make this method unsuitable to use if a LibVLC application is attempting to track dynamic audio device changes as they happen. @param mp: media

player. @return: the current audio output device identifier None if no device is selected or in case of error (the result must be released with free() or L{libvlc_free}()). @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_audio_output_device_id**(*p_instance*, *psz_audio_output*, *i_device*)
Backward compatibility stub. Do not use in new code. deprecated Use L{libvlc_audio_output_device_list_get}() instead. @return: always None.

pyparrot.utils.vlc.**libvlc_audio_output_device_list_get**(*p_instance*, *aout*)
Gets a list of audio output devices for a given audio output module, See L{libvlc_audio_output_device_set}(). @note: Not all audio outputs support this. In particular, an empty (None) list of devices does B{not} imply that the specified audio output does not work. @note: The list might not be exhaustive. @warning: Some audio output devices in the list might not actually work in some circumstances. By default, it is recommended to not specify any explicit audio device. @param p_instance: libvlc instance. @param aout: audio output name (as returned by L{libvlc_audio_output_list_get}()). @return: A None-terminated linked list of potential audio output devices. It must be freed with L{libvlc_audio_output_device_list_release}(). @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_audio_output_device_list_release**(*p_list*)
Frees a list of available audio output devices. @param p_list: list with audio outputs for release. @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_audio_output_device_longname**(*p_instance*, *psz_output*, *i_device*)
Backward compatibility stub. Do not use in new code. deprecated Use L{libvlc_audio_output_device_list_get}() instead. @return: always None.

pyparrot.utils.vlc.**libvlc_audio_output_device_set**(*mp*, *module*, *device_id*)
Configures an explicit audio output device. If the module paramater is None, audio output will be moved to the device specified by the device identifier string immediately. This is the recommended usage. A list of adequate potential device strings can be obtained with L{libvlc_audio_output_device_enum}(). However passing None is supported in LibVLC version 2.2.0 and later only; in earlier versions, this function would have no effects when the module parameter was None. If the module parameter is not None, the device parameter of the corresponding audio output, if it exists, will be set to the specified string. Note that some audio output modules do not have such a parameter (notably MMDevice and PulseAudio). A list of adequate potential device strings can be obtained with L{libvlc_audio_output_device_list_get}(). @note: This function does not select the specified audio output plugin. L{libvlc_audio_output_set}() is used for that purpose. @warning: The syntax for the device parameter depends on the audio output. Some audio output modules require further parameters (e.g. a channels map in the case of ALSA). @param mp: media player. @param module: If None, current audio output module. if non-None, name of audio output module. @param device_id: device identifier string. @return: Nothing. Errors are ignored (this is a design bug).

pyparrot.utils.vlc.**libvlc_audio_output_list_get**(*p_instance*)
Gets the list of available audio output modules. @param p_instance: libvlc instance. @return: list of available audio outputs. It must be freed with In case of error, None is returned.

pyparrot.utils.vlc.**libvlc_audio_output_list_release**(*p_list*)
Frees the list of available audio output modules. @param p_list: list with audio outputs for release.

pyparrot.utils.vlc.**libvlc_audio_output_set**(*p_mi*, *psz_name*)
Selects an audio output module. @note: Any change will take be effect only after playback is stopped and restarted. Audio output cannot be changed while playing. @param p_mi: media player. @param psz_name: name of audio output, use psz_name of See L{AudioOutput}. @return: 0 if function succeeded, -1 on error.

pyparrot.utils.vlc.**libvlc_audio_set_callbacks**(*mp*, *play*, *pause*, *resume*, *flush*, *drain*, *opaque*)
Sets callbacks and private data for decoded audio. Use L{libvlc_audio_set_format}() or L{libvlc_audio_set_format_callbacks}() to configure the decoded audio format. @note: The audio callbacks override any other audio output mechanism. If the callbacks are set, LibVLC will B{not} output audio

in any way. @param mp: the media player. @param play: callback to play audio samples (must not be None). @param pause: callback to pause playback (or None to ignore). @param resume: callback to resume playback (or None to ignore). @param flush: callback to flush audio buffers (or None to ignore). @param drain: callback to drain audio buffers (or None to ignore). @param opaque: private pointer for the audio callbacks (as first parameter). @version: LibVLC 2.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_set_channel`**(*p_mi*, *channel*)
Set current audio channel. @param p_mi: media player. @param channel: the audio channel, See libvlc_audio_output_channel_t. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_audio_set_delay`**(*p_mi*, *i_delay*)
Set current audio delay. The audio delay will be reset to zero each time the media changes. @param p_mi: media player. @param i_delay: the audio delay (microseconds). @return: 0 on success, -1 on error. @version: LibVLC 1.1.1 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_set_format`**(*mp*, *format*, *rate*, *channels*)
Sets a fixed decoded audio format. This only works in combination with L{libvlc_audio_set_callbacks}(), and is mutually exclusive with L{libvlc_audio_set_format_callbacks}(). @param mp: the media player. @param format: a four-characters string identifying the sample format (e.g. "S16N" or "FL32"). @param rate: sample rate (expressed in Hz). @param channels: channels count. @version: LibVLC 2.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_set_format_callbacks`**(*mp*, *setup*, *cleanup*)
Sets decoded audio format via callbacks. This only works in combination with L{libvlc_audio_set_callbacks}(). @param mp: the media player. @param setup: callback to select the audio format (cannot be None). @param cleanup: callback to release any allocated resources (or None). @version: LibVLC 2.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_set_mute`**(*p_mi*, *status*)
Set mute status. @param p_mi: media player. @param status: If status is true then mute, otherwise unmute @warning This function does not always work. If there are no active audio playback stream, the mute status might not be available. If digital pass-through (S/PDIF, HDMI. . .) is in use, muting may be unapplicable. Also some audio output plugins do not support muting at all. @note To force silent playback, disable all audio tracks. This is more efficient and reliable than mute.

`pyparrot.utils.vlc.`**`libvlc_audio_set_track`**(*p_mi*, *i_track*)
Set current audio track. @param p_mi: media player. @param i_track: the track ID (i_id field from track description). @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_audio_set_volume`**(*p_mi*, *i_volume*)
Set current software audio volume. @param p_mi: media player. @param i_volume: the volume in percents (0 = mute, 100 = 0dB). @return: 0 if the volume was set, -1 if it was out of range.

`pyparrot.utils.vlc.`**`libvlc_audio_set_volume_callback`**(*mp*, *set_volume*)
Set callbacks and private data for decoded audio. This only works in combination with L{libvlc_audio_set_callbacks}(). Use L{libvlc_audio_set_format}() or L{libvlc_audio_set_format_callbacks}() to configure the decoded audio format. @param mp: the media player. @param set_volume: callback to apply audio volume, or None to apply volume in software. @version: LibVLC 2.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_audio_toggle_mute`**(*p_mi*)
Toggle mute status. @param p_mi: media player @warning Toggling mute atomically is not always possible: On some platforms, other processes can mute the VLC audio playback stream asynchronously. Thus, there is a small race condition where toggling will not work. See also the limitations of L{libvlc_audio_set_mute}().

`pyparrot.utils.vlc.`**`libvlc_chapter_descriptions_release`**(*p_chapters*, *i_count*)
Release a chapter description. @param p_chapters: chapter description array to release. @param i_count: number of chapter descriptions to release. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_clearerr`**()

Clears the LibVLC error status for the current thread. This is optional. By default, the error status is automatically overridden when a new error occurs, and destroyed when the thread exits.

pyparrot.utils.vlc.**libvlc_clock**()
Return the current time as defined by LibVLC. The unit is the microsecond. Time increases monotonically (regardless of time zone changes and RTC adjustements). The origin is arbitrary but consistent across the whole system (e.g. the system uptim, the time since the system was booted). @note: On systems that support it, the POSIX monotonic clock is used.

pyparrot.utils.vlc.**libvlc_dialog_dismiss**(*p_id*)
Dismiss a dialog After this call, p_id won't be valid anymore See libvlc_dialog_cbs.pf_cancel. @param p_id: id of the dialog. @return: 0 on success, or -1 on error. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_dialog_get_context**(*p_id*)
Return the opaque pointer associated with the dialog id. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_dialog_post_action**(*p_id*, *i_action*)
Post a question answer After this call, p_id won't be valid anymore See libvlc_dialog_cbs.pf_display_question. @param p_id: id of the dialog. @param i_action: 1 for action1, 2 for action2. @return: 0 on success, or -1 on error. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_dialog_post_login**(*p_id*, *psz_username*, *psz_password*, *b_store*)
Post a login answer After this call, p_id won't be valid anymore See libvlc_dialog_cbs.pf_display_login. @param p_id: id of the dialog. @param psz_username: valid and non empty string. @param psz_password: valid string (can be empty). @param b_store: if true, store the credentials. @return: 0 on success, or -1 on error. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_dialog_set_context**(*p_id*, *p_context*)
Associate an opaque pointer with the dialog id. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_event_attach**(*p_event_manager*, *i_event_type*, *f_callback*, *user_data*)
Register for an event notification. @param p_event_manager: the event manager to which you want to attach to. Generally it is obtained by vlc_my_object_event_manager() where my_object is the object you want to listen to. @param i_event_type: the desired event to which we want to listen. @param f_callback: the function to call when i_event_type occurs. @param user_data: user provided data to carry with the event. @return: 0 on success, ENOMEM on error.

pyparrot.utils.vlc.**libvlc_event_detach**(*p_event_manager*, *i_event_type*, *f_callback*, *p_user_data*)
Unregister an event notification. @param p_event_manager: the event manager. @param i_event_type: the desired event to which we want to unregister. @param f_callback: the function to call when i_event_type occurs. @param p_user_data: user provided data to carry with the event.

pyparrot.utils.vlc.**libvlc_event_type_name**(*event_type*)
Get an event's type name. @param event_type: the desired event.

pyparrot.utils.vlc.**libvlc_get_changeset**()
Retrieve libvlc changeset. Example: "aa9bce0bc4". @return: a string containing the libvlc changeset.

pyparrot.utils.vlc.**libvlc_get_compiler**()
Retrieve libvlc compiler version. Example: "gcc version 4.2.3 (Ubuntu 4.2.3-2ubuntu6)". @return: a string containing the libvlc compiler version.

pyparrot.utils.vlc.**libvlc_get_fullscreen**(*p_mi*)
Get current fullscreen status. @param p_mi: the media player. @return: the fullscreen status (boolean) libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_get_log_verbosity**(*p_instance*)
Always returns minus one. This function is only provided for backward compatibility. @param p_instance: ignored. @return: always -1.

`pyparrot.utils.vlc.`**`libvlc_get_version`**`()`
Retrieve libvlc version. Example: "1.1.0-git The Luggage". @return: a string containing the libvlc version.

`pyparrot.utils.vlc.`**`libvlc_hex_version`**`()`
Return the libvlc version in hex or 0 if unavailable.

`pyparrot.utils.vlc.`**`libvlc_log_clear`**`(`*p_log*`)`
This function does nothing. It is only provided for backward compatibility. @param p_log: ignored.

`pyparrot.utils.vlc.`**`libvlc_log_close`**`(`*p_log*`)`
Frees memory allocated by L{libvlc_log_open}(). @param p_log: libvlc log instance or None.

`pyparrot.utils.vlc.`**`libvlc_log_count`**`(`*p_log*`)`
Always returns zero. This function is only provided for backward compatibility. @param p_log: ignored. @return: always zero.

`pyparrot.utils.vlc.`**`libvlc_log_get_context`**`(`*ctx*`)`
Gets log message debug infos. This function retrieves self-debug information about a log message: - the name of the VLC module emitting the message, - the name of the source code module (i.e. file) and - the line number within the source code module. The returned module name and file name will be None if unknown. The returned line number will similarly be zero if unknown. @param ctx: message context (as passed to the @ref libvlc_log_cb callback). @return: module module name storage (or None), file source code file name storage (or None), line source code file line number storage (or None). @version: LibVLC 2.1.0 or later.

`pyparrot.utils.vlc.`**`libvlc_log_get_iterator`**`(`*p_log*`)`
This function does nothing useful. It is only provided for backward compatibility. @param p_log: ignored. @return: an unique pointer or None on error or if the parameter was None.

`pyparrot.utils.vlc.`**`libvlc_log_get_object`**`(`*ctx*, *id*`)`
Gets log message info. This function retrieves meta-information about a log message: - the type name of the VLC object emitting the message, - the object header if any, and - a temporaly-unique object identifier. This information is mainly meant for B{manual} troubleshooting. The returned type name may be "generic" if unknown, but it cannot be None. The returned header will be None if unset; in current versions, the header is used to distinguish for VLM inputs. The returned object ID will be zero if the message is not associated with any VLC object. @param ctx: message context (as passed to the @ref libvlc_log_cb callback). @return: name object name storage (or None), header object header (or None), line source code file line number storage (or None). @version: LibVLC 2.1.0 or later.

`pyparrot.utils.vlc.`**`libvlc_log_iterator_free`**`(`*p_iter*`)`
Frees memory allocated by L{libvlc_log_get_iterator}(). @param p_iter: libvlc log iterator or None.

`pyparrot.utils.vlc.`**`libvlc_log_iterator_has_next`**`(`*p_iter*`)`
Always returns zero. This function is only provided for backward compatibility. @param p_iter: ignored. @return: always zero.

`pyparrot.utils.vlc.`**`libvlc_log_iterator_next`**`(`*p_iter*, *p_buf*`)`
Always returns None. This function is only provided for backward compatibility. @param p_iter: libvlc log iterator or None. @param p_buf: ignored. @return: always None.

`pyparrot.utils.vlc.`**`libvlc_log_open`**`(`*p_instance*`)`
This function does nothing useful. It is only provided for backward compatibility. @param p_instance: libvlc instance. @return: an unique pointer or None on error.

`pyparrot.utils.vlc.`**`libvlc_log_set`**`(`*p_instance*, *cb*, *data*`)`
Sets the logging callback for a LibVLC instance. This function is thread-safe: it will wait for any pending callbacks invocation to complete. @param cb: callback function pointer. @param data: opaque data pointer for the callback function @note Some log messages (especially debug) are emitted by LibVLC while is being initialized. These messages cannot be captured with this interface. @warning A deadlock may occur if this function is called from the callback. @param p_instance: libvlc instance. @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_log_set_file**(*p_instance*, *stream*)
Sets up logging to a file. @param p_instance: libvlc instance. @param stream: FILE pointer opened for writing (the FILE pointer must remain valid until L{libvlc_log_unset}()). @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_log_unset**(*p_instance*)
Unsets the logging callback. This function deregisters the logging callback for a LibVLC instance. This is rarely needed as the callback is implicitly unset when the instance is destroyed. @note: This function will wait for any pending callbacks invocation to complete (causing a deadlock if called from within the callback). @param p_instance: libvlc instance. @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_media_add_option**(*p_md*, *psz_options*)
Add an option to the media. This option will be used to determine how the media_player will read the media. This allows to use VLC's advanced reading/streaming options on a per-media basis. @note: The options are listed in 'vlc –long-help' from the command line, e.g. "-sout-all". Keep in mind that available options and their semantics vary across LibVLC versions and builds. @warning: Not all options affects L{Media} objects: Specifically, due to architectural issues most audio and video options, such as text renderer options, have no effects on an individual media. These options must be set through L{libvlc_new}() instead. @param p_md: the media descriptor. @param psz_options: the options (as a string).

pyparrot.utils.vlc.**libvlc_media_add_option_flag**(*p_md*, *psz_options*, *i_flags*)
Add an option to the media with configurable flags. This option will be used to determine how the media_player will read the media. This allows to use VLC's advanced reading/streaming options on a per-media basis. The options are detailed in vlc –long-help, for instance "–sout-all". Note that all options are not usable on medias: specifically, due to architectural issues, video-related options such as text renderer options cannot be set on a single media. They must be set on the whole libvlc instance instead. @param p_md: the media descriptor. @param psz_options: the options (as a string). @param i_flags: the flags for this option.

pyparrot.utils.vlc.**libvlc_media_discoverer_event_manager**(*p_mdis*)
Get event manager from media service discover object. deprecated Useless, media_discoverer events are only triggered when calling L{libvlc_media_discoverer_start}() and L{libvlc_media_discoverer_stop}(). @param p_mdis: media service discover object. @return: event manager object.

pyparrot.utils.vlc.**libvlc_media_discoverer_is_running**(*p_mdis*)
Query if media service discover object is running. @param p_mdis: media service discover object. @return: true if running, false if not libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_discoverer_list_get**(*p_inst*, *i_cat*, *ppp_services*)
Get media discoverer services by category. @param p_inst: libvlc instance. @param i_cat: category of services to fetch. @param ppp_services: address to store an allocated array of media discoverer services (must be freed with L{libvlc_media_discoverer_list_release}() by the caller) [OUT]. @return: the number of media discoverer services (0 on error). @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_media_discoverer_list_release**(*pp_services*, *i_count*)
Release an array of media discoverer services. @param pp_services: array to release. @param i_count: number of elements in the array. @version: LibVLC 3.0.0 and later. See L{libvlc_media_discoverer_list_get}().

pyparrot.utils.vlc.**libvlc_media_discoverer_localized_name**(*p_mdis*)
Get media service discover object its localized name. deprecated Useless, use L{libvlc_media_discoverer_list_get}() to get the longname of the service discovery. @param p_mdis: media discover object. @return: localized name or None if the media_discoverer is not started.

pyparrot.utils.vlc.**libvlc_media_discoverer_media_list**(*p_mdis*)
Get media service discover media list. @param p_mdis: media service discover object. @return: list of media items.

pyparrot.utils.vlc.**libvlc_media_discoverer_new**(*p_inst*, *psz_name*)
Create a media discoverer object by name. After this object is created, you should attach to media_list events in order to be notified of new items discovered. You need to call L{libvlc_media_discoverer_start}() in order to start the discovery. See L{libvlc_media_discoverer_media_list}

See L{libvlc_media_discoverer_event_manager} See L{libvlc_media_discoverer_start}. @param p_inst: libvlc instance. @param psz_name: service name; use L{libvlc_media_discoverer_list_get}() to get a list of the discoverer names available in this libVLC instance. @return: media discover object or None in case of error. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_discoverer_new_from_name**(*p_inst*, *psz_name*)
deprecated Use L{libvlc_media_discoverer_new}() and L{libvlc_media_discoverer_start}().

pyparrot.utils.vlc.**libvlc_media_discoverer_release**(*p_mdis*)
Release media discover object. If the reference count reaches 0, then the object will be released. @param p_mdis: media service discover object.

pyparrot.utils.vlc.**libvlc_media_discoverer_start**(*p_mdis*)
Start media discovery. To stop it, call L{libvlc_media_discoverer_stop}() or L{libvlc_media_discoverer_list_release}() directly. See L{libvlc_media_discoverer_stop}. @param p_mdis: media discover object. @return: -1 in case of error, 0 otherwise. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_discoverer_stop**(*p_mdis*)
Stop media discovery. See L{libvlc_media_discoverer_start}. @param p_mdis: media discover object. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_duplicate**(*p_md*)
Duplicate a media descriptor object. @param p_md: a media descriptor object.

pyparrot.utils.vlc.**libvlc_media_event_manager**(*p_md*)
Get event manager from media descriptor object. NOTE: this function doesn't increment reference counting. @param p_md: a media descriptor object. @return: event manager object.

pyparrot.utils.vlc.**libvlc_media_get_codec_description**(*i_type*, *i_codec*)
Get codec description from media elementary stream. @param i_type: i_type from L{MediaTrack}. @param i_codec: i_codec or i_original_fourcc from L{MediaTrack}. @return: codec description. @version: LibVLC 3.0.0 and later. See L{MediaTrack}.

pyparrot.utils.vlc.**libvlc_media_get_duration**(*p_md*)
Get duration (in ms) of media descriptor object item. @param p_md: media descriptor object. @return: duration of media item or -1 on error.

pyparrot.utils.vlc.**libvlc_media_get_meta**(*p_md*, *e_meta*)
Read the meta of the media. If the media has not yet been parsed this will return None. See L{libvlc_media_parse} See L{libvlc_media_parse_with_options} See libvlc_MediaMetaChanged. @param p_md: the media descriptor. @param e_meta: the meta to read. @return: the media's meta.

pyparrot.utils.vlc.**libvlc_media_get_mrl**(*p_md*)
Get the media resource locator (mrl) from a media descriptor object. @param p_md: a media descriptor object. @return: string with mrl of media descriptor object.

pyparrot.utils.vlc.**libvlc_media_get_parsed_status**(*p_md*)
Get Parsed status for media descriptor object. See libvlc_MediaParsedChanged See libvlc_media_parsed_status_t. @param p_md: media descriptor object. @return: a value of the libvlc_media_parsed_status_t enum. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_get_state**(*p_md*)
Get current state of media descriptor object. Possible media states are libvlc_NothingSpecial=0, libvlc_Opening, libvlc_Playing, libvlc_Paused, libvlc_Stopped, libvlc_Ended, libvlc_Error. See libvlc_state_t. @param p_md: a media descriptor object. @return: state of media descriptor object.

pyparrot.utils.vlc.**libvlc_media_get_stats**(*p_md*, *p_stats*)
Get the current statistics about the media. @param p_md:: media descriptor object. @param p_stats:: structure that contain the statistics about the media (this structure must be allocated by the caller). @return: true if the statistics are available, false otherwise libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_get_tracks_info**(*p_md*)
Get media descriptor's elementary streams description Note, you need to call L{libvlc_media_parse}() or play the media at least once before calling this function. Not doing this will result in an empty array. deprecated Use L{libvlc_media_tracks_get}() instead. @param p_md: media descriptor object. @param tracks: address to store an allocated array of Elementary Streams descriptions (must be freed by the caller) [OUT]. @return: the number of Elementary Streams.

pyparrot.utils.vlc.**libvlc_media_get_type**(*p_md*)
Get the media type of the media descriptor object. @param p_md: media descriptor object. @return: media type. @version: LibVLC 3.0.0 and later. See libvlc_media_type_t.

pyparrot.utils.vlc.**libvlc_media_get_user_data**(*p_md*)
Get media descriptor's user_data. user_data is specialized data accessed by the host application, VLC.framework uses it as a pointer to an native object that references a L{Media} pointer. @param p_md: media descriptor object.

pyparrot.utils.vlc.**libvlc_media_is_parsed**(*p_md*)
Return true is the media descriptor object is parsed deprecated This can return true in case of failure.

Use L{libvlc_media_get_parsed_status}() instead

See libvlc_MediaParsedChanged. @param p_md: media descriptor object. @return: true if media object has been parsed otherwise it returns false libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_library_load**(*p_mlib*)
Load media library. @param p_mlib: media library object. @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_media_library_media_list**(*p_mlib*)
Get media library subitems. @param p_mlib: media library object. @return: media list subitems.

pyparrot.utils.vlc.**libvlc_media_library_new**(*p_instance*)
Create an new Media Library object. @param p_instance: the libvlc instance. @return: a new object or None on error.

pyparrot.utils.vlc.**libvlc_media_library_release**(*p_mlib*)
Release media library object. This functions decrements the reference count of the media library object. If it reaches 0, then the object will be released. @param p_mlib: media library object.

pyparrot.utils.vlc.**libvlc_media_library_retain**(*p_mlib*)
Retain a reference to a media library object. This function will increment the reference counting for this object. Use L{libvlc_media_library_release}() to decrement the reference count. @param p_mlib: media library object.

pyparrot.utils.vlc.**libvlc_media_list_add_media**(*p_ml*, *p_md*)
Add media instance to media list The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @param p_md: a media instance. @return: 0 on success, -1 if the media list is read-only.

pyparrot.utils.vlc.**libvlc_media_list_count**(*p_ml*)
Get count on media list items The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @return: number of items in media list.

pyparrot.utils.vlc.**libvlc_media_list_event_manager**(*p_ml*)
Get libvlc_event_manager from this media list instance. The p_event_manager is immutable, so you don't have to hold the lock. @param p_ml: a media list instance. @return: libvlc_event_manager.

pyparrot.utils.vlc.**libvlc_media_list_index_of_item**(*p_ml*, *p_md*)
Find index position of List media instance in media list. Warning: the function will return the first matched position. The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @param p_md: media instance. @return: position of media instance or -1 if media not found.

pyparrot.utils.vlc.**libvlc_media_list_insert_media**(*p_ml*, *p_md*, *i_pos*)
Insert media instance in media list on a position The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @param p_md: a media instance. @param i_pos: position in array where to insert. @return: 0 on success, -1 if the media list is read-only.

pyparrot.utils.vlc.**libvlc_media_list_is_readonly**(*p_ml*)
This indicates if this media list is read-only from a user point of view. @param p_ml: media list instance. @return: 1 on readonly, 0 on readwrite libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_list_item_at_index**(*p_ml*, *i_pos*)
List media instance in media list at a position The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @param i_pos: position in array where to insert. @return: media instance at position i_pos, or None if not found. In case of success, L{libvlc_media_retain}() is called to increase the refcount on the media.

pyparrot.utils.vlc.**libvlc_media_list_lock**(*p_ml*)
Get lock on media list items. @param p_ml: a media list instance.

pyparrot.utils.vlc.**libvlc_media_list_media**(*p_ml*)
Get media instance from this media list instance. This action will increase the refcount on the media instance. The L{libvlc_media_list_lock} should NOT be held upon entering this function. @param p_ml: a media list instance. @return: media instance.

pyparrot.utils.vlc.**libvlc_media_list_new**(*p_instance*)
Create an empty media list. @param p_instance: libvlc instance. @return: empty media list, or None on error.

pyparrot.utils.vlc.**libvlc_media_list_player_event_manager**(*p_mlp*)
Return the event manager of this media_list_player. @param p_mlp: media list player instance. @return: the event manager.

pyparrot.utils.vlc.**libvlc_media_list_player_get_media_player**(*p_mlp*)
Get media player of the media_list_player instance. @param p_mlp: media list player instance. @return: media player instance @note the caller is responsible for releasing the returned instance.

pyparrot.utils.vlc.**libvlc_media_list_player_get_state**(*p_mlp*)
Get current libvlc_state of media list player. @param p_mlp: media list player instance. @return: libvlc_state_t for media list player.

pyparrot.utils.vlc.**libvlc_media_list_player_is_playing**(*p_mlp*)
Is media list playing? @param p_mlp: media list player instance. @return: true for playing and false for not playing libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_list_player_new**(*p_instance*)
Create new media_list_player. @param p_instance: libvlc instance. @return: media list player instance or None on error.

pyparrot.utils.vlc.**libvlc_media_list_player_next**(*p_mlp*)
Play next item from media list. @param p_mlp: media list player instance. @return: 0 upon success -1 if there is no next item.

pyparrot.utils.vlc.**libvlc_media_list_player_pause**(*p_mlp*)
Toggle pause (or resume) media list. @param p_mlp: media list player instance.

pyparrot.utils.vlc.**libvlc_media_list_player_play**(*p_mlp*)
Play media list. @param p_mlp: media list player instance.

pyparrot.utils.vlc.**libvlc_media_list_player_play_item**(*p_mlp*, *p_md*)
Play the given media item. @param p_mlp: media list player instance. @param p_md: the media instance. @return: 0 upon success, -1 if the media is not part of the media list.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_play_item_at_index`**(*p_mlp*, *i_index*)
Play media list item at position index. @param p_mlp: media list player instance. @param i_index: index in media list to play. @return: 0 upon success -1 if the item wasn't found.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_previous`**(*p_mlp*)
Play previous item from media list. @param p_mlp: media list player instance. @return: 0 upon success -1 if there is no previous item.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_release`**(*p_mlp*)
Release a media_list_player after use Decrement the reference count of a media player object. If the reference count is 0, then L{libvlc_media_list_player_release}() will release the media player object. If the media player object has been released, then it should not be used again. @param p_mlp: media list player instance.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_retain`**(*p_mlp*)
Retain a reference to a media player list object. Use L{libvlc_media_list_player_release}() to decrement reference count. @param p_mlp: media player list object.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_set_media_list`**(*p_mlp*, *p_mlist*)
Set the media list associated with the player. @param p_mlp: media list player instance. @param p_mlist: list of media.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_set_media_player`**(*p_mlp*, *p_mi*)
Replace media player in media_list_player with this instance. @param p_mlp: media list player instance. @param p_mi: media player instance.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_set_pause`**(*p_mlp*, *do_pause*)
Pause or resume media list. @param p_mlp: media list player instance. @param do_pause: play/resume if zero, pause if non-zero. @version: LibVLC 3.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_set_playback_mode`**(*p_mlp*, *e_mode*)
Sets the playback mode for the playlist. @param p_mlp: media list player instance. @param e_mode: playback mode specification.

`pyparrot.utils.vlc.`**`libvlc_media_list_player_stop`**(*p_mlp*)
Stop playing media list. @param p_mlp: media list player instance.

`pyparrot.utils.vlc.`**`libvlc_media_list_release`**(*p_ml*)
Release media list created with L{libvlc_media_list_new}(). @param p_ml: a media list created with L{libvlc_media_list_new}().

`pyparrot.utils.vlc.`**`libvlc_media_list_remove_index`**(*p_ml*, *i_pos*)
Remove media instance from media list on a position The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance. @param i_pos: position in array where to insert. @return: 0 on success, -1 if the list is read-only or the item was not found.

`pyparrot.utils.vlc.`**`libvlc_media_list_retain`**(*p_ml*)
Retain reference to a media list. @param p_ml: a media list created with L{libvlc_media_list_new}().

`pyparrot.utils.vlc.`**`libvlc_media_list_set_media`**(*p_ml*, *p_md*)
Associate media instance with this media list instance. If another media instance was present it will be released. The L{libvlc_media_list_lock} should NOT be held upon entering this function. @param p_ml: a media list instance. @param p_md: media instance to add.

`pyparrot.utils.vlc.`**`libvlc_media_list_unlock`**(*p_ml*)
Release lock on media list items The L{libvlc_media_list_lock} should be held upon entering this function. @param p_ml: a media list instance.

`pyparrot.utils.vlc.`**`libvlc_media_new_as_node`**(*p_instance*, *psz_name*)
Create a media as an empty node with a given name. See L{libvlc_media_release}. @param p_instance: the instance. @param psz_name: the name of the node. @return: the new empty media or None on error.

`pyparrot.utils.vlc.`**`libvlc_media_new_callbacks`**(*instance*, *open_cb*, *read_cb*, *seek_cb*, *close_cb*, *opaque*)

Create a media with custom callbacks to read the data from. @param instance: LibVLC instance. @param open_cb: callback to open the custom bitstream input media. @param read_cb: callback to read data (must not be None). @param seek_cb: callback to seek, or None if seeking is not supported. @param close_cb: callback to close the media, or None if unnecessary. @param opaque: data pointer for the open callback. @return: the newly created media or None on error @note If open_cb is None, the opaque pointer will be passed to read_cb, seek_cb and close_cb, and the stream size will be treated as unknown. @note The callbacks may be called asynchronously (from another thread). A single stream instance need not be reentrant. However the open_cb needs to be reentrant if the media is used by multiple player instances. @warning The callbacks may be used until all or any player instances that were supplied the media item are stopped. See L{libvlc_media_release}. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_new_fd`**(*p_instance*, *fd*)

Create a media for an already open file descriptor. The file descriptor shall be open for reading (or reading and writing). Regular file descriptors, pipe read descriptors and character device descriptors (including TTYs) are supported on all platforms. Block device descriptors are supported where available. Directory descriptors are supported on systems that provide fdopendir(). Sockets are supported on all platforms where they are file descriptors, i.e. all except Windows. @note: This library will B{not} automatically close the file descriptor under any circumstance. Nevertheless, a file descriptor can usually only be rendered once in a media player. To render it a second time, the file descriptor should probably be rewound to the beginning with lseek(). See L{libvlc_media_release}. @param p_instance: the instance. @param fd: open file descriptor. @return: the newly created media or None on error. @version: LibVLC 1.1.5 and later.

`pyparrot.utils.vlc.`**`libvlc_media_new_location`**(*p_instance*, *psz_mrl*)

Create a media with a certain given media resource location, for instance a valid URL. @note: To refer to a local file with this function, the file://... URI syntax B{must} be used (see IETF RFC3986). We recommend using L{libvlc_media_new_path}() instead when dealing with local files. See L{libvlc_media_release}. @param p_instance: the instance. @param psz_mrl: the media location. @return: the newly created media or None on error.

`pyparrot.utils.vlc.`**`libvlc_media_new_path`**(*p_instance*, *path*)

Create a media for a certain file path. See L{libvlc_media_release}. @param p_instance: the instance. @param path: local filesystem path. @return: the newly created media or None on error.

`pyparrot.utils.vlc.`**`libvlc_media_parse`**(*p_md*)

Parse a media. This fetches (local) art, meta data and tracks information. The method is synchronous. deprecated This function could block indefinitely.

> Use L{libvlc_media_parse_with_options}() instead

See L{libvlc_media_parse_with_options} See L{libvlc_media_get_meta} See L{libvlc_media_get_tracks_info}. @param p_md: media descriptor object.

`pyparrot.utils.vlc.`**`libvlc_media_parse_async`**(*p_md*)

Parse a media. This fetches (local) art, meta data and tracks information. The method is the asynchronous of L{libvlc_media_parse}(). To track when this is over you can listen to libvlc_MediaParsedChanged event. However if the media was already parsed you will not receive this event. deprecated You can't be sure to receive the libvlc_MediaParsedChanged

> event (you can wait indefinitely for this event). Use L{libvlc_media_parse_with_options}() instead

See L{libvlc_media_parse} See libvlc_MediaParsedChanged See L{libvlc_media_get_meta} See L{libvlc_media_get_tracks_info}. @param p_md: media descriptor object.

`pyparrot.utils.vlc.`**`libvlc_media_parse_stop`**(*p_md*)

Stop the parsing of the media When the media parsing is stopped, the libvlc_MediaParsedChanged event will be sent with the libvlc_media_parsed_status_timeout status. See L{libvlc_media_parse_with_options}. @param p_md: media descriptor object. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_parse_with_options**(*p_md*, *parse_flag*, *timeout*)
Parse the media asynchronously with options. This fetches (local or network) art, meta data and/or tracks information. This method is the extended version of L{libvlc_media_parse_with_options}(). To track when this is over you can listen to libvlc_MediaParsedChanged event. However if this functions returns an error, you will not receive any events. It uses a flag to specify parse options (see libvlc_media_parse_flag_t). All these flags can be combined. By default, media is parsed if it's a local file. @note: Parsing can be aborted with L{libvlc_media_parse_stop}(). See libvlc_MediaParsedChanged See L{libvlc_media_get_meta} See L{libvlc_media_tracks_get} See L{libvlc_media_get_parsed_status} See libvlc_media_parse_flag_t. @param p_md: media descriptor object. @param parse_flag: parse options: @param timeout: maximum time allowed to preparse the media. If -1, the default "preparse-timeout" option will be used as a timeout. If 0, it will wait indefinitely. If > 0, the timeout will be used (in milliseconds). @return: -1 in case of error, 0 otherwise. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_add_slave**(*p_mi*, *i_type*, *psz_uri*, *b_select*)
Add a slave to the current media player. @note: If the player is playing, the slave will be added directly. This call will also update the slave list of the attached L{Media}. @param p_mi: the media player. @param i_type: subtitle or audio. @param psz_uri: Uri of the slave (should contain a valid scheme). @param b_select: True if this slave should be selected when it's loaded. @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 and later. See L{libvlc_media_slaves_add}.

pyparrot.utils.vlc.**libvlc_media_player_can_pause**(*p_mi*)
Can this media player be paused? @param p_mi: the media player. @return: true if the media player can pause libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_player_event_manager**(*p_mi*)
Get the Event Manager from which the media player send event. @param p_mi: the Media Player. @return: the event manager associated with p_mi.

pyparrot.utils.vlc.**libvlc_media_player_get_agl**(*p_mi*)
deprecated Use L{libvlc_media_player_get_nsobject}() instead.

pyparrot.utils.vlc.**libvlc_media_player_get_chapter**(*p_mi*)
Get movie chapter. @param p_mi: the Media Player. @return: chapter number currently playing, or -1 if there is no media.

pyparrot.utils.vlc.**libvlc_media_player_get_chapter_count**(*p_mi*)
Get movie chapter count. @param p_mi: the Media Player. @return: number of chapters in movie, or -1.

pyparrot.utils.vlc.**libvlc_media_player_get_chapter_count_for_title**(*p_mi*, *i_title*)
Get title chapter count. @param p_mi: the Media Player. @param i_title: title. @return: number of chapters in title, or -1.

pyparrot.utils.vlc.**libvlc_media_player_get_fps**(*p_mi*)
Get movie fps rate This function is provided for backward compatibility. It cannot deal with multiple video tracks. In LibVLC versions prior to 3.0, it would also fail if the file format did not convey the frame rate explicitly. deprecated Consider using L{libvlc_media_tracks_get}() instead. @param p_mi: the Media Player. @return: frames per second (fps) for this playing movie, or 0 if unspecified.

pyparrot.utils.vlc.**libvlc_media_player_get_full_chapter_descriptions**(*p_mi*, *i_chapters_of_title*, *pp_chapters*)
Get the full description of available chapters. @param p_mi: the media player. @param i_chapters_of_title: index of the title to query for chapters (uses current title if set to -1). @param pp_chapters: address to store an allocated array of chapter descriptions descriptions (must be freed with L{libvlc_chapter_descriptions_release}() by the caller) [OUT]. @return: the number of chapters (-1 on error). @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_media_player_get_full_title_descriptions**(*p_mi*, *titles*)
Get the full description of available titles. @param p_mi: the media player. @param titles: address to store an

allocated array of title descriptions descriptions (must be freed with L{libvlc_title_descriptions_release}() by the caller) [OUT]. @return: the number of titles (-1 on error). @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_hwnd`**(*p_mi*)
Get the Windows API window handle (HWND) previously set with L{libvlc_media_player_set_hwnd}(). The handle will be returned even if LibVLC is not currently outputting any video to it. @param p_mi: the Media Player. @return: a window handle or None if there are none.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_length`**(*p_mi*)
Get the current movie length (in ms). @param p_mi: the Media Player. @return: the movie length (in ms), or -1 if there is no media.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_media`**(*p_mi*)
Get the media used by the media_player. @param p_mi: the Media Player. @return: the media associated with p_mi, or None if no media is associated.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_nsobject`**(*p_mi*)
Get the NSView handler previously set with L{libvlc_media_player_set_nsobject}(). @param p_mi: the Media Player. @return: the NSView handler or 0 if none where set.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_position`**(*p_mi*)
Get movie position as percentage between 0.0 and 1.0. @param p_mi: the Media Player. @return: movie position, or -1. in case of error.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_rate`**(*p_mi*)
Get the requested movie play rate. @warning: Depending on the underlying media, the requested rate may be different from the real playback rate. @param p_mi: the Media Player. @return: movie play rate.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_role`**(*p_mi*)

Gets the media role. @param p_mi: media player. @return: the media player role (

ef libvlc_media_player_role_t). @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_state`**(*p_mi*)
Get current movie state. @param p_mi: the Media Player. @return: the current state of the media player (playing, paused, . . .) See libvlc_state_t.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_time`**(*p_mi*)
Get the current movie time (in ms). @param p_mi: the Media Player. @return: the movie time (in ms), or -1 if there is no media.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_title`**(*p_mi*)
Get movie title. @param p_mi: the Media Player. @return: title number currently playing, or -1.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_title_count`**(*p_mi*)
Get movie title count. @param p_mi: the Media Player. @return: title number count, or -1.

`pyparrot.utils.vlc.`**`libvlc_media_player_get_xwindow`**(*p_mi*)
Get the X Window System window identifier previously set with L{libvlc_media_player_set_xwindow}(). Note that this will return the identifier even if VLC is not currently using it (for instance if it is playing an audio-only input). @param p_mi: the Media Player. @return: an X window ID, or 0 if none where set.

`pyparrot.utils.vlc.`**`libvlc_media_player_has_vout`**(*p_mi*)
How many video outputs does this media player have? @param p_mi: the media player. @return: the number of video outputs.

`pyparrot.utils.vlc.`**`libvlc_media_player_is_playing`**(*p_mi*)
is_playing. @param p_mi: the Media Player. @return: 1 if the media player is playing, 0 otherwise libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_player_is_seekable**(*p_mi*)
Is this media player seekable? @param p_mi: the media player. @return: true if the media player can seek libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_player_navigate**(*p_mi*, *navigate*)
Navigate through DVD Menu. @param p_mi: the Media Player. @param navigate: the Navigation mode. @version: libVLC 2.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_new**(*p_libvlc_instance*)
Create an empty Media Player object. @param p_libvlc_instance: the libvlc instance in which the Media Player should be created. @return: a new media player object, or None on error.

pyparrot.utils.vlc.**libvlc_media_player_new_from_media**(*p_md*)
Create a Media Player object from a Media. @param p_md: the media. Afterwards the p_md can be safely destroyed. @return: a new media player object, or None on error.

pyparrot.utils.vlc.**libvlc_media_player_next_chapter**(*p_mi*)
Set next chapter (if applicable). @param p_mi: the Media Player.

pyparrot.utils.vlc.**libvlc_media_player_next_frame**(*p_mi*)
Display the next frame (if supported). @param p_mi: the media player.

pyparrot.utils.vlc.**libvlc_media_player_pause**(*p_mi*)
Toggle pause (no effect if there is no media). @param p_mi: the Media Player.

pyparrot.utils.vlc.**libvlc_media_player_play**(*p_mi*)
Play. @param p_mi: the Media Player. @return: 0 if playback started (and was already started), or -1 on error.

pyparrot.utils.vlc.**libvlc_media_player_previous_chapter**(*p_mi*)
Set previous chapter (if applicable). @param p_mi: the Media Player.

pyparrot.utils.vlc.**libvlc_media_player_program_scrambled**(*p_mi*)
Check if the current program is scrambled. @param p_mi: the media player. @return: true if the current program is scrambled libvlc_return_bool. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_release**(*p_mi*)
Release a media_player after use Decrement the reference count of a media player object. If the reference count is 0, then L{libvlc_media_player_release}() will release the media player object. If the media player object has been released, then it should not be used again. @param p_mi: the Media Player to free.

pyparrot.utils.vlc.**libvlc_media_player_retain**(*p_mi*)
Retain a reference to a media player object. Use L{libvlc_media_player_release}() to decrement reference count. @param p_mi: media player object.

pyparrot.utils.vlc.**libvlc_media_player_set_agl**(*p_mi*, *drawable*)
deprecated Use L{libvlc_media_player_set_nsobject}() instead.

pyparrot.utils.vlc.**libvlc_media_player_set_android_context**(*p_mi*, *p_awindow_handler*)
Set the android context. @param p_mi: the media player. @param p_awindow_handler: org.videolan.libvlc.AWindow jobject owned by the org.videolan.libvlc.MediaPlayer class from the libvlc-android project. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_media_player_set_chapter**(*p_mi*, *i_chapter*)
Set movie chapter (if applicable). @param p_mi: the Media Player. @param i_chapter: chapter number to play.

pyparrot.utils.vlc.**libvlc_media_player_set_equalizer**(*p_mi*, *p_equalizer*)
Apply new equalizer settings to a media player. The equalizer is first created by invoking L{libvlc_audio_equalizer_new}() or L{libvlc_audio_equalizer_new_from_preset}(). It is possible to apply new equalizer settings to a media player whether the media player is currently playing media or not. Invoking this method will immediately apply the new equalizer settings to the audio output of the currently playing media if there is any. If there is no currently playing media, the new equalizer settings will be applied later if and when

new media is played. Equalizer settings will automatically be applied to subsequently played media. To disable the equalizer for a media player invoke this method passing None for the p_equalizer parameter. The media player does not keep a reference to the supplied equalizer so it is safe for an application to release the equalizer reference any time after this method returns. @param p_mi: opaque media player handle. @param p_equalizer: opaque equalizer handle, or None to disable the equalizer for this media player. @return: zero on success, -1 on error. @version: LibVLC 2.2.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_set_evas_object**(*p_mi*, *p_evas_object*)
Set the EFL Evas Object. @param p_mi: the media player. @param p_evas_object: a valid EFL Evas Object (Evas_Object). @return: -1 if an error was detected, 0 otherwise. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_media_player_set_hwnd**(*p_mi*, *drawable*)
Set a Win32/Win64 API window handle (HWND) where the media player should render its video output. If LibVLC was built without Win32/Win64 API output support, then this has no effects. @param p_mi: the Media Player. @param drawable: windows handle of the drawable.

pyparrot.utils.vlc.**libvlc_media_player_set_media**(*p_mi*, *p_md*)
Set the media that will be used by the media_player. If any, previous md will be released. @param p_mi: the Media Player. @param p_md: the Media. Afterwards the p_md can be safely destroyed.

pyparrot.utils.vlc.**libvlc_media_player_set_nsobject**(*p_mi*, *drawable*)
Set the NSView handler where the media player should render its video output. Use the vout called "macosx". The drawable is an NSObject that follow the VLCOpenGLVideoViewEmbedding protocol: @code.m @protocol VLCOpenGLVideoViewEmbedding <NSObject> - (void)addVoutSubview:(NSView *)view; - (void)removeVoutSubview:(NSView *)view; @end @endcode Or it can be an NSView object. If you want to use it along with Qt see the QMacCocoaViewContainer. Then the following code should work: @code.mm

> NSView *video = [[NSView alloc] init]; QMacCocoaViewContainer *container = new QMacCocoaViewContainer(video, parent); L{libvlc_media_player_set_nsobject}(mp, video); [video release];

@endcode You can find a live example in VLCVideoView in VLCKit.framework. @param p_mi: the Media Player. @param drawable: the drawable that is either an NSView or an object following the VLCOpenGLVideoViewEmbedding protocol.

pyparrot.utils.vlc.**libvlc_media_player_set_pause**(*mp*, *do_pause*)
Pause or resume (no effect if there is no media). @param mp: the Media Player. @param do_pause: play/resume if zero, pause if non-zero. @version: LibVLC 1.1.1 or later.

pyparrot.utils.vlc.**libvlc_media_player_set_position**(*p_mi*, *f_pos*)
Set movie position as percentage between 0.0 and 1.0. This has no effect if playback is not enabled. This might not work depending on the underlying input format and protocol. @param p_mi: the Media Player. @param f_pos: the position.

pyparrot.utils.vlc.**libvlc_media_player_set_rate**(*p_mi*, *rate*)
Set movie play rate. @param p_mi: the Media Player. @param rate: movie play rate to set. @return: -1 if an error was detected, 0 otherwise (but even then, it might not actually work depending on the underlying media protocol).

pyparrot.utils.vlc.**libvlc_media_player_set_renderer**(*p_mi*, *p_item*)
Set a renderer to the media player @note: must be called before the first call of L{libvlc_media_player_play}() to take effect. See L{libvlc_renderer_discoverer_new}. @param p_mi: the Media Player. @param p_item: an item discovered by L{libvlc_renderer_discoverer_start}(). @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_set_role**(*p_mi*, *role*)

Sets the media role. @param p_mi: media player. @param role: the media player role (

ef libvlc_media_player_role_t). @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_media_player_set_time**(*p_mi*, *i_time*)
Set the movie time (in ms). This has no effect if no media is being played. Not all formats and protocols support this. @param p_mi: the Media Player. @param i_time: the movie time (in ms).

pyparrot.utils.vlc.**libvlc_media_player_set_title**(*p_mi*, *i_title*)
Set movie title. @param p_mi: the Media Player. @param i_title: title number to play.

pyparrot.utils.vlc.**libvlc_media_player_set_video_title_display**(*p_mi*, *position*, *timeout*)
Set if, and how, the video title will be shown when media is played. @param p_mi: the media player. @param position: position at which to display the title, or libvlc_position_disable to prevent the title from being displayed. @param timeout: title display timeout in milliseconds (ignored if libvlc_position_disable). @version: libVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_media_player_set_xwindow**(*p_mi*, *drawable*)
Set an X Window System drawable where the media player should render its video output. The call takes effect when the playback starts. If it is already started, it might need to be stopped before changes apply. If LibVLC was built without X11 output support, then this function has no effects. By default, LibVLC will capture input events on the video rendering area. Use L{libvlc_video_set_mouse_input}() and L{libvlc_video_set_key_input}() to disable that and deliver events to the parent window / to the application instead. By design, the X11 protocol delivers input events to only one recipient. @warning The application must call the XInitThreads() function from Xlib before L{libvlc_new}(), and before any call to XOpenDisplay() directly or via any other library. Failure to call XInitThreads() will seriously impede LibVLC performance. Calling XOpenDisplay() before XInitThreads() will eventually crash the process. That is a limitation of Xlib. @param p_mi: media player. @param drawable: X11 window ID @note The specified identifier must correspond to an existing Input/Output class X11 window. Pixmaps are B{not} currently supported. The default X11 server is assumed, i.e. that specified in the DISPLAY environment variable. @warning LibVLC can deal with invalid X11 handle errors, however some display drivers (EGL, GLX, VA and/or VDPAU) can unfortunately not. Thus the window handle must remain valid until playback is stopped, otherwise the process may abort or crash. @bug No more than one window handle per media player instance can be specified. If the media has multiple simultaneously active video tracks, extra tracks will be rendered into external windows beyond the control of the application.

pyparrot.utils.vlc.**libvlc_media_player_stop**(*p_mi*)
Stop (no effect if there is no media). @param p_mi: the Media Player.

pyparrot.utils.vlc.**libvlc_media_player_will_play**(*p_mi*)
Is the player able to play. @param p_mi: the Media Player. @return: boolean libvlc_return_bool.

pyparrot.utils.vlc.**libvlc_media_release**(*p_md*)
Decrement the reference count of a media descriptor object. If the reference count is 0, then L{libvlc_media_release}() will release the media descriptor object. It will send out an libvlc_MediaFreed event to all listeners. If the media descriptor object has been released it should not be used again. @param p_md: the media descriptor.

pyparrot.utils.vlc.**libvlc_media_retain**(*p_md*)
Retain a reference to a media descriptor object (libvlc_media_t). Use L{libvlc_media_release}() to decrement the reference count of a media descriptor object. @param p_md: the media descriptor.

pyparrot.utils.vlc.**libvlc_media_save_meta**(*p_md*)
Save the meta previously set. @param p_md: the media desriptor. @return: true if the write operation was successful.

pyparrot.utils.vlc.**libvlc_media_set_meta**(*p_md*, *e_meta*, *psz_value*)
Set the meta of the media (this function will not save the meta, call L{libvlc_media_save_meta} in order to save the meta). @param p_md: the media descriptor. @param e_meta: the meta to write. @param psz_value: the media's meta.

pyparrot.utils.vlc.**libvlc_media_set_user_data**(*p_md*, *p_new_user_data*)

Sets media descriptor's user_data. user_data is specialized data accessed by the host application, VLC.framework uses it as a pointer to an native object that references a L{Media} pointer. @param p_md: media descriptor object. @param p_new_user_data: pointer to user data.

`pyparrot.utils.vlc.`**`libvlc_media_slaves_add`**(*p_md*, *i_type*, *i_priority*, *psz_uri*)
Add a slave to the current media. A slave is an external input source that may contains an additional subtitle track (like a .srt) or an additional audio track (like a .ac3). @note: This function must be called before the media is parsed (via L{libvlc_media_parse_with_options}()) or before the media is played (via L{libvlc_media_player_play}()). @param p_md: media descriptor object. @param i_type: subtitle or audio. @param i_priority: from 0 (low priority) to 4 (high priority). @param psz_uri: Uri of the slave (should contain a valid scheme). @return: 0 on success, -1 on error. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_slaves_clear`**(*p_md*)
Clear all slaves previously added by L{libvlc_media_slaves_add}() or internally. @param p_md: media descriptor object. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_slaves_get`**(*p_md*, *ppp_slaves*)
Get a media descriptor's slave list The list will contain slaves parsed by VLC or previously added by L{libvlc_media_slaves_add}(). The typical use case of this function is to save a list of slave in a database for a later use. @param p_md: media descriptor object. @param ppp_slaves: address to store an allocated array of slaves (must be freed with L{libvlc_media_slaves_release}()) [OUT]. @return: the number of slaves (zero on error). @version: LibVLC 3.0.0 and later. See L{libvlc_media_slaves_add}.

`pyparrot.utils.vlc.`**`libvlc_media_slaves_release`**(*pp_slaves*, *i_count*)
Release a media descriptor's slave list. @param pp_slaves: slave array to release. @param i_count: number of elements in the array. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_subitems`**(*p_md*)
Get subitems of media descriptor object. This will increment the reference count of supplied media descriptor object. Use L{libvlc_media_list_release}() to decrement the reference counting. @param p_md: media descriptor object. @return: list of media descriptor subitems or None.

`pyparrot.utils.vlc.`**`libvlc_media_tracks_get`**(*p_md*, *tracks*)
Get media descriptor's elementary streams description Note, you need to call L{libvlc_media_parse}() or play the media at least once before calling this function. Not doing this will result in an empty array. @param p_md: media descriptor object. @param tracks: address to store an allocated array of Elementary Streams descriptions (must be freed with L{libvlc_media_tracks_release}. @return: the number of Elementary Streams (zero on error). @version: LibVLC 2.1.0 and later.

`pyparrot.utils.vlc.`**`libvlc_media_tracks_release`**(*p_tracks*, *i_count*)
Release media descriptor's elementary streams description array. @param p_tracks: tracks info array to release. @param i_count: number of elements in the array. @version: LibVLC 2.1.0 and later.

`pyparrot.utils.vlc.`**`libvlc_module_description_list_release`**(*p_list*)
Release a list of module descriptions. @param p_list: the list to be released.

`pyparrot.utils.vlc.`**`libvlc_new`**(*argc*, *argv*)
Create and initialize a libvlc instance. This functions accept a list of "command line" arguments similar to the main(). These arguments affect the LibVLC instance default configuration. @note LibVLC may create threads. Therefore, any thread-unsafe process initialization must be performed before calling L{libvlc_new}(). In particular and where applicable: - setlocale() and textdomain(), - setenv(), unsetenv() and putenv(), - with the X11 display system, XInitThreads()

> (see also L{libvlc_media_player_set_xwindow}()) and

- on Microsoft Windows, SetErrorMode().
- sigprocmask() shall never be invoked; pthread_sigmask() can be used.

On POSIX systems, the SIGCHLD signal B{must not} be ignored, i.e. the signal handler must set to SIG_DFL or a function pointer, not SIG_IGN. Also while LibVLC is active, the wait() function shall not be called, and any call to waitpid() shall use a strictly positive value for the first parameter (i.e. the PID). Failure to follow those rules may lead to a deadlock or a busy loop. Also on POSIX systems, it is recommended that the SIGPIPE signal be blocked, even if it is not, in principles, necessary, e.g.: @code @endcode On Microsoft Windows Vista/2008, the process error mode SEM_FAILCRITICALERRORS flag B{must} be set before using LibVLC. On later versions, that is optional and unnecessary. Also on Microsoft Windows (Vista and any later version), setting the default DLL directories to SYSTEM32 exclusively is strongly recommended for security reasons: @code @endcode. @param argc: the number of arguments (should be 0). @param argv: list of arguments (should be None). @return: the libvlc instance or None in case of error. @version Arguments are meant to be passed from the command line to LibVLC, just like VLC media player does. The list of valid arguments depends on the LibVLC version, the operating system and platform, and set of available LibVLC plugins. Invalid or unsupported arguments will cause the function to fail (i.e. return None). Also, some arguments may alter the behaviour or otherwise interfere with other LibVLC functions. @warning There is absolutely no warranty or promise of forward, backward and cross-platform compatibility with regards to L{libvlc_new}() arguments. We recommend that you do not use them, other than when debugging.

`pyparrot.utils.vlc.`**`libvlc_playlist_play`**(*p_instance*, *i_id*, *i_options*, *ppsz_options*)

Start playing (if there is any item in the playlist). Additionnal playlist item options can be specified for addition to the item before it is played. @param p_instance: the playlist instance. @param i_id: the item to play. If this is a negative number, the next item will be selected. Otherwise, the item with the given ID will be played. @param i_options: the number of options to add to the item. @param ppsz_options: the options to add to the item.

`pyparrot.utils.vlc.`**`libvlc_release`**(*p_instance*)

Decrement the reference count of a libvlc instance, and destroy it if it reaches zero. @param p_instance: the instance to destroy.

`pyparrot.utils.vlc.`**`libvlc_renderer_discoverer_event_manager`**(*p_rd*)

Get the event manager of the renderer discoverer The possible events to attach are @ref libvlc_RendererDiscovererItemAdded and @ref libvlc_RendererDiscovererItemDeleted. The @ref libvlc_renderer_item_t struct passed to event callbacks is owned by VLC, users should take care of holding/releasing this struct for their internal usage. See libvlc_event_t.u.renderer_discoverer_item_added.item See libvlc_event_t.u.renderer_discoverer_item_removed.item. @return: a valid event manager (can't fail). @version: LibVLC 3.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_renderer_discoverer_list_get`**(*p_inst*, *ppp_services*)

Get media discoverer services See libvlc_renderer_list_release(). @param p_inst: libvlc instance. @param ppp_services: address to store an allocated array of renderer discoverer services (must be freed with libvlc_renderer_list_release() by the caller) [OUT]. @return: the number of media discoverer services (0 on error). @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_renderer_discoverer_list_release`**(*pp_services*, *i_count*)

Release an array of media discoverer services See L{libvlc_renderer_discoverer_list_get}(). @param pp_services: array to release. @param i_count: number of elements in the array. @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_renderer_discoverer_new`**(*p_inst*, *psz_name*)

Create a renderer discoverer object by name After this object is created, you should attach to events in order to be notified of the discoverer events. You need to call L{libvlc_renderer_discoverer_start}() in order to start the discovery. See L{libvlc_renderer_discoverer_event_manager}() See L{libvlc_renderer_discoverer_start}(). @param p_inst: libvlc instance. @param psz_name: service name; use L{libvlc_renderer_discoverer_list_get}() to get a list of the discoverer names available in this libVLC instance. @return: media discover object or None in case of error. @version: LibVLC 3.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_renderer_discoverer_release`**(*p_rd*)

Release a renderer discoverer object. @param p_rd: renderer discoverer object. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_discoverer_start**(*p_rd*)
Start renderer discovery To stop it, call L{libvlc_renderer_discoverer_stop}() or L{libvlc_renderer_discoverer_release}() directly. See L{libvlc_renderer_discoverer_stop}(). @param p_rd: renderer discoverer object. @return: -1 in case of error, 0 otherwise. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_discoverer_stop**(*p_rd*)
Stop renderer discovery. See L{libvlc_renderer_discoverer_start}(). @param p_rd: renderer discoverer object. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_flags**(*p_item*)
Get the flags of a renderer item See LIBVLC_RENDERER_CAN_AUDIO See LIBVLC_RENDERER_CAN_VIDEO. @return: bitwise flag: capabilities of the renderer, see. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_hold**(*p_item*)
Hold a renderer item, i.e. creates a new reference This functions need to called from the libvlc_RendererDiscovererItemAdded callback if the libvlc user wants to use this item after. (for display or for passing it to the mediaplayer for example). @return: the current item. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_icon_uri**(*p_item*)
Get the icon uri of a renderer item. @return: the uri of the item's icon (can be None, must *not* be freed). @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_name**(*p_item*)
Get the human readable name of a renderer item. @return: the name of the item (can't be None, must *not* be freed). @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_release**(*p_item*)
Releases a renderer item, i.e. decrements its reference counter. @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_renderer_item_type**(*p_item*)
Get the type (not translated) of a renderer item. For now, the type can only be "chromecast" ("upnp", "airplay" may come later). @return: the type of the item (can't be None, must *not* be freed). @version: LibVLC 3.0.0 or later.

pyparrot.utils.vlc.**libvlc_retain**(*p_instance*)
Increments the reference count of a libvlc instance. The initial reference count is 1 after L{libvlc_new}() returns. @param p_instance: the instance to reference.

pyparrot.utils.vlc.**libvlc_set_app_id**(*p_instance*, *id*, *version*, *icon*)
Sets some meta-information about the application. See also L{libvlc_set_user_agent}(). @param p_instance: LibVLC instance. @param id: Java-style application identifier, e.g. "com.acme.foobar". @param version: application version numbers, e.g. "1.2.3". @param icon: application icon name, e.g. "foobar". @version: LibVLC 2.1.0 or later.

pyparrot.utils.vlc.**libvlc_set_fullscreen**(*p_mi*, *b_fullscreen*)
Enable or disable fullscreen. @warning: With most window managers, only a top-level windows can be in full-screen mode. Hence, this function will not operate properly if L{libvlc_media_player_set_xwindow}() was used to embed the video in a non-top-level window. In that case, the embedding window must be reparented to the root window B{before} fullscreen mode is enabled. You will want to reparent it back to its normal parent when disabling fullscreen. @param p_mi: the media player. @param b_fullscreen: boolean for fullscreen status.

pyparrot.utils.vlc.**libvlc_set_log_verbosity**(*p_instance*, *level*)
This function does nothing. It is only provided for backward compatibility. @param p_instance: ignored. @param level: ignored.

pyparrot.utils.vlc.**libvlc_set_user_agent**(*p_instance*, *name*, *http*)
Sets the application name. LibVLC passes this as the user agent string when a protocol requires it. @param p_instance: LibVLC instance. @param name: human-readable application name, e.g. "FooBar player 1.2.3". @param http: HTTP User Agent, e.g. "FooBar/1.2.3 Python/2.6.0". @version: LibVLC 1.1.1 or later.

pyparrot.utils.vlc.**libvlc_title_descriptions_release**(*p_titles*, *i_count*)
Release a title description. @param p_titles: title description array to release. @param i_count: number of title descriptions to release. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_toggle_fullscreen**(*p_mi*)
Toggle fullscreen status on non-embedded video outputs. @warning: The same limitations applies to this function as to L{libvlc_set_fullscreen}(). @param p_mi: the media player.

pyparrot.utils.vlc.**libvlc_toggle_teletext**(*p_mi*)
Toggle teletext transparent status on video output. deprecated use L{libvlc_video_set_teletext}() instead. @param p_mi: the media player.

pyparrot.utils.vlc.**libvlc_track_description_list_release**(*p_track_description*)
Release (free) L{TrackDescription}. @param p_track_description: the structure to release.

pyparrot.utils.vlc.**libvlc_track_description_release**(*p_track_description*)
deprecated Use L{libvlc_track_description_list_release}() instead.

pyparrot.utils.vlc.**libvlc_video_filter_list_get**(*p_instance*)
Returns a list of video filters that are available. @param p_instance: libvlc instance. @return: a list of module descriptions. It should be freed with L{libvlc_module_description_list_release}(). In case of an error, None is returned. See L{ModuleDescription} See L{libvlc_module_description_list_release}.

pyparrot.utils.vlc.**libvlc_video_get_adjust_float**(*p_mi*, *option*)
Get float adjust option. @param p_mi: libvlc media player instance. @param option: adjust option to get, values of libvlc_video_adjust_option_t. @version: LibVLC 1.1.1 and later.

pyparrot.utils.vlc.**libvlc_video_get_adjust_int**(*p_mi*, *option*)
Get integer adjust option. @param p_mi: libvlc media player instance. @param option: adjust option to get, values of libvlc_video_adjust_option_t. @version: LibVLC 1.1.1 and later.

pyparrot.utils.vlc.**libvlc_video_get_aspect_ratio**(*p_mi*)
Get current video aspect ratio. @param p_mi: the media player. @return: the video aspect ratio or None if unspecified (the result must be released with free() or L{libvlc_free}()).

pyparrot.utils.vlc.**libvlc_video_get_chapter_description**(*p_mi*, *i_title*)
Get the description of available chapters for specific title. @param p_mi: the media player. @param i_title: selected title. @return: list containing description of available chapter for title i_title. It must be freed with L{libvlc_track_description_list_release}().

pyparrot.utils.vlc.**libvlc_video_get_crop_geometry**(*p_mi*)
Get current crop filter geometry. @param p_mi: the media player. @return: the crop filter geometry or None if unset.

pyparrot.utils.vlc.**libvlc_video_get_cursor**(*p_mi*, *num*)
Get the mouse pointer coordinates over a video. Coordinates are expressed in terms of the decoded video resolution, B{not} in terms of pixels on the screen/viewport (to get the latter, you can query your windowing system directly). Either of the coordinates may be negative or larger than the corresponding dimension of the video, if the cursor is outside the rendering area. @warning: The coordinates may be out-of-date if the pointer is not located on the video rendering area. LibVLC does not track the pointer if it is outside of the video widget. @note: LibVLC does not support multiple pointers (it does of course support multiple input devices sharing the same pointer) at the moment. @param p_mi: media player. @param num: number of the video (starting from, and most commonly 0). @return: px abscissa, py ordinate.

pyparrot.utils.vlc.**libvlc_video_get_height**(*p_mi*)
: Get current video height. deprecated Use L{libvlc_video_get_size}() instead. @param p_mi: the media player. @return: the video pixel height or 0 if not applicable.

pyparrot.utils.vlc.**libvlc_video_get_logo_int**(*p_mi*, *option*)
: Get integer logo option. @param p_mi: libvlc media player instance. @param option: logo option to get, values of libvlc_video_logo_option_t.

pyparrot.utils.vlc.**libvlc_video_get_marquee_int**(*p_mi*, *option*)
: Get an integer marquee option value. @param p_mi: libvlc media player. @param option: marq option to get See libvlc_video_marquee_int_option_t.

pyparrot.utils.vlc.**libvlc_video_get_marquee_string**(*p_mi*, *option*)
: Get a string marquee option value. @param p_mi: libvlc media player. @param option: marq option to get See libvlc_video_marquee_string_option_t.

pyparrot.utils.vlc.**libvlc_video_get_scale**(*p_mi*)
: Get the current video scaling factor. See also L{libvlc_video_set_scale}(). @param p_mi: the media player. @return: the currently configured zoom factor, or 0. if the video is set to fit to the output window/drawable automatically.

pyparrot.utils.vlc.**libvlc_video_get_size**(*p_mi*, *num*)
: Get the pixel dimensions of a video. @param p_mi: media player. @param num: number of the video (starting from, and most commonly 0). @return: px pixel width, py pixel height.

pyparrot.utils.vlc.**libvlc_video_get_spu**(*p_mi*)
: Get current video subtitle. @param p_mi: the media player. @return: the video subtitle selected, or -1 if none.

pyparrot.utils.vlc.**libvlc_video_get_spu_count**(*p_mi*)
: Get the number of available video subtitles. @param p_mi: the media player. @return: the number of available video subtitles.

pyparrot.utils.vlc.**libvlc_video_get_spu_delay**(*p_mi*)
: Get the current subtitle delay. Positive values means subtitles are being displayed later, negative values earlier. @param p_mi: media player. @return: time (in microseconds) the display of subtitles is being delayed. @version: LibVLC 2.0.0 or later.

pyparrot.utils.vlc.**libvlc_video_get_spu_description**(*p_mi*)
: Get the description of available video subtitles. @param p_mi: the media player. @return: list containing description of available video subtitles. It must be freed with L{libvlc_track_description_list_release}().

pyparrot.utils.vlc.**libvlc_video_get_teletext**(*p_mi*)
: Get current teletext page requested or 0 if it's disabled. Teletext is disabled by default, call L{libvlc_video_set_teletext}() to enable it. @param p_mi: the media player. @return: the current teletext page requested.

pyparrot.utils.vlc.**libvlc_video_get_title_description**(*p_mi*)
: Get the description of available titles. @param p_mi: the media player. @return: list containing description of available titles. It must be freed with L{libvlc_track_description_list_release}().

pyparrot.utils.vlc.**libvlc_video_get_track**(*p_mi*)
: Get current video track. @param p_mi: media player. @return: the video track ID (int) or -1 if no active input.

pyparrot.utils.vlc.**libvlc_video_get_track_count**(*p_mi*)
: Get number of available video tracks. @param p_mi: media player. @return: the number of available video tracks (int).

pyparrot.utils.vlc.**libvlc_video_get_track_description**(*p_mi*)
: Get the description of available video tracks. @param p_mi: media player. @return: list with description of available video tracks, or None on error. It must be freed with L{libvlc_track_description_list_release}().

`pyparrot.utils.vlc.`**`libvlc_video_get_width`**(*p_mi*)
Get current video width. deprecated Use L{libvlc_video_get_size}() instead. @param p_mi: the media player. @return: the video pixel width or 0 if not applicable.

`pyparrot.utils.vlc.`**`libvlc_video_new_viewpoint`**()
Create a video viewpoint structure. @return: video viewpoint or None (the result must be released with free() or L{libvlc_free}()). @version: LibVLC 3.0.0 and later.

`pyparrot.utils.vlc.`**`libvlc_video_set_adjust_float`**(*p_mi*, *option*, *value*)
Set adjust option as float. Options that take a different type value are ignored. @param p_mi: libvlc media player instance. @param option: adust option to set, values of libvlc_video_adjust_option_t. @param value: adjust option value. @version: LibVLC 1.1.1 and later.

`pyparrot.utils.vlc.`**`libvlc_video_set_adjust_int`**(*p_mi*, *option*, *value*)
Set adjust option as integer. Options that take a different type value are ignored. Passing libvlc_adjust_enable as option value has the side effect of starting (arg !0) or stopping (arg 0) the adjust filter. @param p_mi: libvlc media player instance. @param option: adust option to set, values of libvlc_video_adjust_option_t. @param value: adjust option value. @version: LibVLC 1.1.1 and later.

`pyparrot.utils.vlc.`**`libvlc_video_set_aspect_ratio`**(*p_mi*, *psz_aspect*)
Set new video aspect ratio. @param p_mi: the media player. @param psz_aspect: new video aspect-ratio or None to reset to default @note Invalid aspect ratios are ignored.

`pyparrot.utils.vlc.`**`libvlc_video_set_callbacks`**(*mp*, *lock*, *unlock*, *display*, *opaque*)
Set callbacks and private data to render decoded video to a custom area in memory. Use L{libvlc_video_set_format}() or L{libvlc_video_set_format_callbacks}() to configure the decoded format. @warning: Rendering video into custom memory buffers is considerably less efficient than rendering in a custom window as normal. For optimal perfomances, VLC media player renders into a custom window, and does not use this function and associated callbacks. It is B{highly recommended} that other LibVLC-based application do likewise. To embed video in a window, use libvlc_media_player_set_xid() or equivalent depending on the operating system. If window embedding does not fit the application use case, then a custom LibVLC video output display plugin is required to maintain optimal video rendering performances. The following limitations affect performance: - Hardware video decoding acceleration will either be disabled completely,

> or require (relatively slow) copy from video/DSP memory to main memory.

- Sub-pictures (subtitles, on-screen display, etc.) must be blent into the main picture by the CPU instead of the GPU.
- Depending on the video format, pixel format conversion, picture scaling, cropping and/or picture re-orientation, must be performed by the CPU instead of the GPU.
- Memory copying is required between LibVLC reference picture buffers and application buffers (between lock and unlock callbacks).

@param mp: the media player. @param lock: callback to lock video memory (must not be None). @param unlock: callback to unlock video memory (or None if not needed). @param display: callback to display video (or None if not needed). @param opaque: private pointer for the three callbacks (as first parameter). @version: LibVLC 1.1.1 or later.

`pyparrot.utils.vlc.`**`libvlc_video_set_crop_geometry`**(*p_mi*, *psz_geometry*)
Set new crop filter geometry. @param p_mi: the media player. @param psz_geometry: new crop filter geometry (None to unset).

`pyparrot.utils.vlc.`**`libvlc_video_set_deinterlace`**(*p_mi*, *psz_mode*)
Enable or disable deinterlace filter. @param p_mi: libvlc media player. @param psz_mode: type of deinterlace filter, None to disable.

`pyparrot.utils.vlc.`**`libvlc_video_set_format`**(*mp*, *chroma*, *width*, *height*, *pitch*)
: Set decoded video chroma and dimensions. This only works in combination with L{libvlc_video_set_callbacks}(), and is mutually exclusive with L{libvlc_video_set_format_callbacks}(). @param mp: the media player. @param chroma: a four-characters string identifying the chroma (e.g. "RV32" or "YUYV"). @param width: pixel width. @param height: pixel height. @param pitch: line pitch (in bytes). @version: LibVLC 1.1.1 or later. @bug: All pixel planes are expected to have the same pitch. To use the YCbCr color space with chrominance subsampling, consider using L{libvlc_video_set_format_callbacks}() instead.

`pyparrot.utils.vlc.`**`libvlc_video_set_format_callbacks`**(*mp*, *setup*, *cleanup*)
: Set decoded video chroma and dimensions. This only works in combination with L{libvlc_video_set_callbacks}(). @param mp: the media player. @param setup: callback to select the video format (cannot be None). @param cleanup: callback to release any allocated resources (or None). @version: LibVLC 2.0.0 or later.

`pyparrot.utils.vlc.`**`libvlc_video_set_key_input`**(*p_mi*, *on*)
: Enable or disable key press events handling, according to the LibVLC hotkeys configuration. By default and for historical reasons, keyboard events are handled by the LibVLC video widget. @note: On X11, there can be only one subscriber for key press and mouse click events per window. If your application has subscribed to those events for the X window ID of the video widget, then LibVLC will not be able to handle key presses and mouse clicks in any case. @warning: This function is only implemented for X11 and Win32 at the moment. @param p_mi: the media player. @param on: true to handle key press events, false to ignore them.

`pyparrot.utils.vlc.`**`libvlc_video_set_logo_int`**(*p_mi*, *option*, *value*)
: Set logo option as integer. Options that take a different type value are ignored. Passing libvlc_logo_enable as option value has the side effect of starting (arg !0) or stopping (arg 0) the logo filter. @param p_mi: libvlc media player instance. @param option: logo option to set, values of libvlc_video_logo_option_t. @param value: logo option value.

`pyparrot.utils.vlc.`**`libvlc_video_set_logo_string`**(*p_mi*, *option*, *psz_value*)
: Set logo option as string. Options that take a different type value are ignored. @param p_mi: libvlc media player instance. @param option: logo option to set, values of libvlc_video_logo_option_t. @param psz_value: logo option value.

`pyparrot.utils.vlc.`**`libvlc_video_set_marquee_int`**(*p_mi*, *option*, *i_val*)
: Enable, disable or set an integer marquee option Setting libvlc_marquee_Enable has the side effect of enabling (arg !0) or disabling (arg 0) the marq filter. @param p_mi: libvlc media player. @param option: marq option to set See libvlc_video_marquee_int_option_t. @param i_val: marq option value.

`pyparrot.utils.vlc.`**`libvlc_video_set_marquee_string`**(*p_mi*, *option*, *psz_text*)
: Set a marquee string option. @param p_mi: libvlc media player. @param option: marq option to set See libvlc_video_marquee_string_option_t. @param psz_text: marq option value.

`pyparrot.utils.vlc.`**`libvlc_video_set_mouse_input`**(*p_mi*, *on*)
: Enable or disable mouse click events handling. By default, those events are handled. This is needed for DVD menus to work, as well as a few video filters such as "puzzle". See L{libvlc_video_set_key_input}(). @warning: This function is only implemented for X11 and Win32 at the moment. @param p_mi: the media player. @param on: true to handle mouse click events, false to ignore them.

`pyparrot.utils.vlc.`**`libvlc_video_set_scale`**(*p_mi*, *f_factor*)
: Set the video scaling factor. That is the ratio of the number of pixels on screen to the number of pixels in the original decoded video in each dimension. Zero is a special value; it will adjust the video to the output window/drawable (in windowed mode) or the entire screen. Note that not all video outputs support scaling. @param p_mi: the media player. @param f_factor: the scaling factor, or zero.

`pyparrot.utils.vlc.`**`libvlc_video_set_spu`**(*p_mi*, *i_spu*)
: Set new video subtitle. @param p_mi: the media player. @param i_spu: video subtitle track to select (i_id from track description). @return: 0 on success, -1 if out of range.

pyparrot.utils.vlc.**libvlc_video_set_spu_delay**(*p_mi*, *i_delay*)
Set the subtitle delay. This affects the timing of when the subtitle will be displayed. Positive values result in subtitles being displayed later, while negative values will result in subtitles being displayed earlier. The subtitle delay will be reset to zero each time the media changes. @param p_mi: media player. @param i_delay: time (in microseconds) the display of subtitles should be delayed. @return: 0 on success, -1 on error. @version: LibVLC 2.0.0 or later.

pyparrot.utils.vlc.**libvlc_video_set_subtitle_file**(*p_mi*, *psz_subtitle*)
Set new video subtitle file. deprecated Use L{libvlc_media_player_add_slave}() instead. @param p_mi: the media player. @param psz_subtitle: new video subtitle file. @return: the success status (boolean).

pyparrot.utils.vlc.**libvlc_video_set_teletext**(*p_mi*, *i_page*)

Set new teletext page to retrieve. This function can also be used to send a teletext key. @param p_mi: the media player. @param i_page: teletex page number requested. This value can be 0 to disable teletext, a number in the range]0;1000[to show the requested page, or a

ef libvlc_teletext_key_t. 100 is the default teletext page.

pyparrot.utils.vlc.**libvlc_video_set_track**(*p_mi*, *i_track*)
Set video track. @param p_mi: media player. @param i_track: the track ID (i_id field from track description). @return: 0 on success, -1 if out of range.

pyparrot.utils.vlc.**libvlc_video_take_snapshot**(*p_mi*, *num*, *psz_filepath*, *i_width*, *i_height*)
Take a snapshot of the current video window. If i_width AND i_height is 0, original size is used. If i_width XOR i_height is 0, original aspect-ratio is preserved. @param p_mi: media player instance. @param num: number of video output (typically 0 for the first/only one). @param psz_filepath: the path of a file or a folder to save the screenshot into. @param i_width: the snapshot's width. @param i_height: the snapshot's height. @return: 0 on success, -1 if the video was not found.

pyparrot.utils.vlc.**libvlc_video_update_viewpoint**(*p_mi*, *p_viewpoint*, *b_absolute*)
Update the video viewpoint information. @note: It is safe to call this function before the media player is started. @param p_mi: the media player. @param p_viewpoint: video viewpoint allocated via L{libvlc_video_new_viewpoint}(). @param b_absolute: if true replace the old viewpoint with the new one. If false, increase/decrease it. @return: -1 in case of error, 0 otherwise @note the values are set asynchronously, it will be used by the next frame displayed. @version: LibVLC 3.0.0 and later.

pyparrot.utils.vlc.**libvlc_vlm_add_broadcast**(*p_instance*, *psz_name*, *psz_input*, *psz_output*, *i_options*, *ppsz_options*, *b_enabled*, *b_loop*)
Add a broadcast, with one input. @param p_instance: the instance. @param psz_name: the name of the new broadcast. @param psz_input: the input MRL. @param psz_output: the output MRL (the parameter to the "sout" variable). @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new broadcast. @param b_loop: Should this broadcast be played in loop ? @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_vlm_add_input**(*p_instance*, *psz_name*, *psz_input*)
Add a media's input MRL. This will add the specified one. @param p_instance: the instance. @param psz_name: the media to work on. @param psz_input: the input MRL. @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_vlm_add_vod**(*p_instance*, *psz_name*, *psz_input*, *i_options*, *ppsz_options*, *b_enabled*, *psz_mux*)
Add a vod, with one input. @param p_instance: the instance. @param psz_name: the name of the new vod media. @param psz_input: the input MRL. @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new vod. @param psz_mux: the muxer of the vod media. @return: 0 on success, -1 on error.

pyparrot.utils.vlc.**libvlc_vlm_change_media**(*p_instance*, *psz_name*, *psz_input*, *psz_output*, *i_options*, *ppsz_options*, *b_enabled*, *b_loop*)
Edit the parameters of a media. This will delete all existing inputs and add the specified one. @param

p_instance: the instance. @param psz_name: the name of the new broadcast. @param psz_input: the input MRL. @param psz_output: the output MRL (the parameter to the "sout" variable). @param i_options: number of additional options. @param ppsz_options: additional options. @param b_enabled: boolean for enabling the new broadcast. @param b_loop: Should this broadcast be played in loop ? @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_del_media`**(*p_instance*, *psz_name*)
Delete a media (VOD or broadcast). @param p_instance: the instance. @param psz_name: the media to delete. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_event_manager`**(*p_instance*)
Get libvlc_event_manager from a vlm media. The p_event_manager is immutable, so you don't have to hold the lock. @param p_instance: a libvlc instance. @return: libvlc_event_manager.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_chapter`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance chapter number by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: chapter as number or -1 on error. @bug: will always return 0.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_length`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance length by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: length of media item or -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_position`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance position by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: position as float or -1. on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_rate`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance playback rate by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: playback rate or -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_seekable`**(*p_instance*, *psz_name*, *i_instance*)
Is libvlc instance seekable ? @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: 1 if seekable, 0 if not, -1 if media does not exist. @bug: will always return 0.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_time`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance time by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: time as integer or -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_get_media_instance_title`**(*p_instance*, *psz_name*, *i_instance*)
Get vlm_media instance title number by name or instance id. @param p_instance: a libvlc instance. @param psz_name: name of vlm media instance. @param i_instance: instance id. @return: title as number or -1 on error. @bug: will always return 0.

`pyparrot.utils.vlc.`**`libvlc_vlm_pause_media`**(*p_instance*, *psz_name*)
Pause the named broadcast. @param p_instance: the instance. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_play_media`**(*p_instance*, *psz_name*)
Play the named broadcast. @param p_instance: the instance. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_release`**(*p_instance*)
: Release the vlm instance related to the given L{Instance}. @param p_instance: the instance.

`pyparrot.utils.vlc.`**`libvlc_vlm_seek_media`**(*p_instance*, *psz_name*, *f_percentage*)
: Seek in the named broadcast. @param p_instance: the instance. @param psz_name: the name of the broadcast. @param f_percentage: the percentage to seek to. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_set_enabled`**(*p_instance*, *psz_name*, *b_enabled*)
: Enable or disable a media (VOD or broadcast). @param p_instance: the instance. @param psz_name: the media to work on. @param b_enabled: the new status. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_set_input`**(*p_instance*, *psz_name*, *psz_input*)
: Set a media's input MRL. This will delete all existing inputs and add the specified one. @param p_instance: the instance. @param psz_name: the media to work on. @param psz_input: the input MRL. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_set_loop`**(*p_instance*, *psz_name*, *b_loop*)
: Set a media's loop status. @param p_instance: the instance. @param psz_name: the media to work on. @param b_loop: the new status. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_set_mux`**(*p_instance*, *psz_name*, *psz_mux*)
: Set a media's vod muxer. @param p_instance: the instance. @param psz_name: the media to work on. @param psz_mux: the new muxer. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_set_output`**(*p_instance*, *psz_name*, *psz_output*)
: Set the output for a media. @param p_instance: the instance. @param psz_name: the media to work on. @param psz_output: the output MRL (the parameter to the "sout" variable). @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_show_media`**(*p_instance*, *psz_name*)
: Return information about the named media as a JSON string representation. This function is mainly intended for debugging use, if you want programmatic access to the state of a vlm_media_instance_t, please use the corresponding libvlc_vlm_get_media_instance_xxx -functions. Currently there are no such functions available for vlm_media_t though. @param p_instance: the instance. @param psz_name: the name of the media, if the name is an empty string, all media is described. @return: string with information about named media, or None on error.

`pyparrot.utils.vlc.`**`libvlc_vlm_stop_media`**(*p_instance*, *psz_name*)
: Stop the named broadcast. @param p_instance: the instance. @param psz_name: the name of the broadcast. @return: 0 on success, -1 on error.

`pyparrot.utils.vlc.`**`libvlc_vprinterr`**(*fmt*, *ap*)
: Sets the LibVLC error status and message for the current thread. Any previous error is overridden. @param fmt: the format string. @param ap: the arguments. @return: a nul terminated string in any case.

`pyparrot.utils.vlc.`**`libvlc_wait`**(*p_instance*)
: Waits until an interface causes the instance to exit. You should start at least one interface first, using L{libvlc_add_intf}(). @param p_instance: the instance @warning This function wastes one thread doing basically nothing. libvlc_set_exit_handler() should be used instead.

`class` `pyparrot.utils.vlc.`**`memoize_parameterless`**(*func*)
: Bases: `object`

  Decorator. Caches a parameterless method's return value each time it is called.

  If called later with the same arguments, the cached value is returned (not reevaluated). Adapted from https://wiki.python.org/moin/PythonDecoratorLibrary

`pyparrot.utils.vlc.`**`module_description_list`**(*head*)
: Convert a ModuleDescription linked list to a Python list (and release the former).

pyparrot.utils.vlc.**str_to_bytes**(*s*)
  Translate string or bytes to bytes.

pyparrot.utils.vlc.**string_result**(*result*, *func*, *arguments*)
  Errcheck function. Returns a string and frees the original pointer.

  It assumes the result is a char *.

pyparrot.utils.vlc.**track_description_list**(*head*)
  Convert a TrackDescription linked list to a Python list (and release the former).

Module contents

Submodules

pyparrot.Bebop module

Bebop class holds all of the methods needed to pilot the drone from python and to ask for sensor data back from the drone

Author: Amy McGovern, dramymcgovern@gmail.com

class pyparrot.Bebop.**Bebop**(*drone_type='Bebop2'*, *ip_address=None*)
  Bases: object

  ask_for_state_update()
    Ask for a full state update (likely this should never be used but it can be called if you want to see everything the bebop is storing)

    Returns nothing but it will eventually fill the sensors with all of the state variables as they arrive

  connect(*num_retries*)
    Connects to the drone and re-tries in case of failure the specified number of times. Seamlessly connects to either wifi or BLE depending on how you initialized it

    Param num_retries is the number of times to retry

    Returns True if it succeeds and False otherwise

  disconnect()
    Disconnect the BLE connection. Always call this at the end of your programs to cleanly disconnect.

    Returns void

  emergency_land()
    Sends the land command to the bebop on the high priority/emergency channel. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

    Returns True if the command was sent and False otherwise

  enable_geofence(*value*)
      If geofence is enabled, the drone won't fly over the given max distance.

    1 if the drone can't fly further than max distance, 0 if no limitation on the drone should be done.

    Parameters **value** –

    Returns

flat_trim(*duration=0*)
Sends the flat_trim command to the bebop. Gets the codes for it from the xml files. :param duration: if duration is greater than 0, waits for the trim command to be finished or duration to be reached

flip(*direction*)
Sends the flip command to the bebop. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times. Valid directions to flip are: front, back, right, left

Returns True if the command was sent and False otherwise

fly_direct(*roll*, *pitch*, *yaw*, *vertical_movement*, *duration*)
Direct fly commands using PCMD. Each argument ranges from -100 to 100. Numbers outside that are clipped to that range.

Note that the xml refers to gaz, which is apparently french for vertical movements: http://forum.developer.parrot.com/t/terminology-of-gaz/3146

Parameters

- **roll** –
- **pitch** –
- **yaw** –
- **vertical_movement** –

Returns

is_landed()
Returns true if it is landed or emergency and False otherwise :return:

land()
Sends the land command to the bebop. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

move_relative(*dx*, *dy*, *dz*, *dradians*)
Move relative to our current position and pause until the command is done.

Parameters

- **dx** – change in front axis (meters)
- **dy** – change in right/left (positive is right) (meters)
- **dz** – change in height (positive is DOWN) (meters)
- **dradians** – change in heading in radians

Returns nothing

pan_tilt_camera(*tilt_degrees*, *pan_degrees*)
Send the command to pan/tilt the camera by the specified number of degrees in pan/tilt

Note, this only seems to work in small increments. Use pan_tilt_velocity to get the camera to look straight downward

Parameters

- **tilt_degrees** – tilt degrees
- **pan_degrees** – pan degrees

Returns

`pan_tilt_camera_velocity`(*tilt_velocity*, *pan_velocity*, *duration=0*)
Send the command to tilt the camera by the specified number of degrees per second in pan/tilt. This function has two modes. First, if duration is 0, the initial velocity is sent and then the function returns (meaning the camera will keep moving). If duration is greater than 0, the command executes for that amount of time and then sends a stop command to the camera and then returns.

Parameters

- **`tilt_degrees`** – tile change in degrees per second
- **`pan_degrees`** – pan change in degrees per second
- **`duration`** – seconds to run the command for

Returns

`safe_land`(*timeout*)
Ensure the Bebop lands by sending the command until it shows landed on sensors

`safe_takeoff`(*timeout*)
Sends commands to takeoff until the Bebop reports it is taking off

Parameters **`timeout`** – quit trying to takeoff if it takes more than timeout seconds

`set_exposition`(*value*)
Set image exposure

Parameters **`value`** –

Returns

`set_hull_protection`(*present*)
Set the presence of hull protection - this is only needed for bebop 1 1 if present, 0 if not present

Parameters **`present`** –

Returns

`set_indoor`(*is_outdoor*)
Set bebop 1 to indoor mode (not used in bebop 2!!) 1 if outdoor, 0 if indoor

Parameters **`present`** –

Returns

`set_max_altitude`(*altitude*)
Set max altitude in meters.

Parameters **`altitude`** – altitude in meters

Returns

`set_max_distance`(*distance*)
Set max distance between the takeoff and the drone in meters.

Parameters **`distance`** – distance in meters

Returns

`set_max_rotation_speed`(*speed*)
Set max yaw rotation speed in degree/s

Parameters **`speed`** – max rotation speed for yaw in degree/s

Returns

`set_max_tilt`(*tilt*)
Set max pitch/roll in degrees

Parameters **`tilt`** – max tilt for both pitch and roll in degrees

Returns

`set_max_tilt_rotation_speed`(*speed*)
Set max pitch/roll rotation speed in degree/s

Parameters **`speed`** – max rotation speed for both pitch and roll in degree/s

Returns

`set_max_vertical_speed`(*speed*)
Set max vertical speed in m/s

Parameters **`speed`** – max vertical speed in m/s

Returns

`set_picture_format`(*format*)
Set picture format

Parameters **`format`** –

Returns

`set_saturation`(*value*)
Set image saturation

Parameters **`value`** –

Returns

`set_timelapse`(*enable*, *interval=8*)
Set timelapse mode

Parameters

- **`enable`** –
- **`interval`** –

Returns

`set_user_sensor_callback`(*function*, *args*)
Set the (optional) user callback function for sensors. Every time a sensor is updated, it calls this function.

Parameters

- **`function`** – name of the function
- **`args`** – tuple of arguments to the function

Returns nothing

`set_video_framerate`(*framerate*)
Set video framerate

Parameters **`framerate`** –

Returns

`set_video_recording`(*mode*)
Set video recording mode

Parameters **`mode`** –

Returns

`set_video_resolutions`(*type*)
Set video resolutions

Parameters **`type`** –

Returns

`set_video_stabilization`(*mode*)
Set video stabilization mode

Parameters **`mode`** –

Returns

`set_video_stream_mode`(*mode='low_latency'*)
Set the video mode for the RTP stream. :param: mode: one of 'low_latency', 'high_reliability' or 'high_reliability_low_framerate'

Returns True if the command was sent and False otherwise

`set_white_balance`(*type*)
Set white balance

Parameters **`type`** –

Returns

`smart_sleep`(*timeout*)
Don't call time.sleep directly as it will mess up BLE and miss WIFI packets! Use this which handles packets received while sleeping

Parameters **`timeout`** – number of seconds to sleep

`start_video_stream`()
Sends the start stream command to the bebop. The bebop will start streaming RTP packets on the port defined in wifiConnection.py (55004 by default). The packets can be picked up by opening an approriate SDP file in a media player such as VLC, MPlayer, FFMPEG or OpenCV.

Returns nothing

`stop_video_stream`()
Sends the stop stream command to the bebop. The bebop will stop streaming RTP packets.

Returns nothing

`takeoff`()
Sends the takeoff command to the bebop. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

`update_sensors`(*data_type*, *buffer_id*, *sequence_number*, *raw_data*, *ack*)
Update the sensors (called via the wifi or ble connection)

Parameters

- **`data`** – raw data packet that needs to be parsed
- **`ack`** – True if this packet needs to be ack'd and False otherwise

`class` `pyparrot.Bebop.`**`BebopSensors`**
Bases: `object`

set_user_callback_function(*function*, *args*)
Sets the user callback function (called everytime the sensors are updated)

Parameters

- **function** – name of the user callback function
- **args** – arguments (tuple) to the function

Returns

update(*sensor_name*, *sensor_value*, *sensor_enum*)

pyparrot.DroneVision module

DroneVision is separated from the main Mambo/Bebop class to enable the use of the drone without the FPV camera. If you want to do vision processing, you will need to create a DroneVision object to capture the video stream.

Note that this module relies on the opencv module and the ffmpeg program

Ffmpeg write the images out to the images directory and then they are read in from the user thread. The DroneVisionGUI does not save copies of the images and instead shows you the images on the screen (they are saved to memory only). While you can see the images in real-time from this program using VisionServer, if you need copies of the images, you will want to use the ffmpeg approach. If you want a smaller delay on your image data for real-time control, you likely want to use libvlc and DroneVisionGUI.

Author: Amy McGovern, dramymcgovern@gmail.com

class pyparrot.DroneVision.**DroneVision**(*drone_object*, *model*, *buffer_size=200*, *cleanup_old_images=True*)
Bases: object

close_video()
Stop the vision processing and all its helper threads

get_latest_valid_picture()
Return the latest valid image (from the buffer)

Returns last valid image received from the Mambo

open_video()
Open the video stream using ffmpeg for capturing and processing. The address for the stream is the same for all Mambos and is documented here:

http://forum.developer.parrot.com/t/streaming-address-of-mambo-fpv-for-videoprojection/6442/6

Remember that this will only work if you have connected to the wifi for your mambo!

Note that the old method tried to open the stream directly into opencv but there are known issues with rtsp streams in opencv. We bypassed opencv to use ffmpeg directly and then opencv is used to process the output of ffmpeg

:return True if the vision opened correctly and False otherwise

set_user_callback_function(*user_callback_function=None*, *user_callback_args=None*)
Set the (optional) user callback function for handling the new vision frames. This is run in a separate thread that starts when you start the vision buffering

Parameters

- **user_callback_function** – function
- **user_callback_args** – arguments to the function

Returns

pyparrot.DroneVisionGUI module

DroneVisionGUI is a new class that parallels DroneVision but with several important changes.

1) This module uses VLC instead of FFMPEG

2) This module opens a GUI window to show you the video in real-time (you could watch it in real-time previously through the VisionServer) 3) Because GUI windows are different on different OS's (and in particular OS X behaves differently than linux and windows) and because they want to run in the main program thread, the way your program runs is different. You first open the GUI and then you have the GUI spawn a thread to run your program. 4) This module can use a virtual disk in memory to save the images, thus shortening the time delay for the camera for your programs.

Author: Amy McGovern, dramymcgovern@gmail.com Some of the LIBVLC code comes from Author: Valentin Benke, valentin.benke@aon.at

class `pyparrot.DroneVisionGUI.`**`DroneVisionGUI`**(*drone_object*, *model*, *user_code_to_run*, *user_args*, *buffer_size=200*, *network_caching=200*, *fps=20*, *user_draw_window_fn=None*)

Bases: `object`

`close_exit`()
Land, close the video, and exit the GUI :return:

`close_video`()
Stop the vision processing and all its helper threads

`get_latest_valid_picture`()
Return the latest valid image (from the buffer)

Returns last valid image received from the Mambo

`land`()
Send the land command over the emergency channel when the user pushes the button

Returns

`land_close_exit`()
Called if you Quit the GUI: lands the drone, stops vision, and exits the GUI :return:

`open_video`()
Open the video stream using vlc. Note that this version is blocking meaning this function will NEVER return. If you want to run your own code and not just watch the video, be sure you set your user code in the constructor!

Remember that this will only work if you have connected to the wifi for your mambo!

:return never returns due to QT running in the main loop by requirement

`run_user_code`(*button*)
Start the thread to run the user code :return:

`set_user_callback_function`(*user_callback_function=None*, *user_callback_args=None*)
Set the (optional) user callback function for handling the new vision frames. This is run in a separate thread that starts when you start the vision buffering

Parameters

- **`user_callback_function`** – function

- **user_callback_args** – arguments to the function

Returns

pyparrot.Minidrone module

Mambo class holds all of the methods needed to pilot the drone from python and to ask for sensor data back from the drone

Author: Amy McGovern, dramymcgovern@gmail.com Author: Alexander Zach, https://github.com/alex-zach, groundcam support Author: Valentin Benke, https://github.com/Vabe7, groundcam support

class pyparrot.Minidrone.**Mambo**(*address=”*, *use_wifi=False*)

Bases: *pyparrot.Minidrone.Minidrone*

disconnect()

Disconnect the BLE connection. Always call this at the end of your programs to cleanly disconnect.

Returns void

fire_gun()

Fire the gun (assumes it is attached) - note not supposed under wifi since the camera takes the place of the gun

Returns True if the command was sent and False otherwise (can include errors or asking to do this using wifi)

class pyparrot.Minidrone.**MamboGroundcam**

Bases: object

get_groundcam_picture(*filename*, *cv2_flag*)

Downloads the specified picture from the Mambo and stores it into a tempfile.

Parameters

- **filename** – the name of the file which should be downloaded ON THE MAMBO.
- **cv2_flag** – if true this function will return a cv2 image object, if false the name of the temporary file will be returned

:return False if there was an error during download, if cv2 is True a cv2 frame or it just returns the file name of the temporary file

get_groundcam_pictures_names()

Retruns a list with the names of the pictures stored on the Mambo. :return The list as an array, if there isn’t any file, the array is empty.

class pyparrot.Minidrone.**Minidrone**(*address=”*, *use_wifi=False*)

Bases: object

ask_for_state_update()

Ask for a full state update (likely this should never be used but it can be called if you want to see everything the mambo is storing)

Returns nothing but it will eventually fill the MamboSensors with all of the state variables as they arrive

change_mode()

change_preferred_mode(*mode=’easy’*)

close_claw()

Close the claw - note not supposed under wifi since the camera takes the place of the claw

Returns True if the command was sent and False otherwise (can include errors or asking to do this using wifi)

connect(*num_retries*)
Connects to the drone and re-tries in case of failure the specified number of times. Seamlessly connects to either wifi or BLE depending on how you initialized it

Param num_retries is the number of times to retry

Returns True if it succeeds and False otherwise

emergency()
Sends the emergency command to the mambo. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

flat_trim()
Sends the flat_trim command to the mambo. Gets the codes for it from the xml files.

flip(*direction*)
Sends the flip command to the mambo. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times. Valid directions to flip are: front, back, right, left

Returns True if the command was sent and False otherwise

fly_direct(*roll*, *pitch*, *yaw*, *vertical_movement*, *duration=None*)
Direct fly commands using PCMD. Each argument ranges from -100 to 100. Numbers outside that are clipped to that range.

Note that the xml refers to gaz, which is apparently french for vertical movements: http://forum.developer.parrot.com/t/terminology-of-gaz/3146

duration is optional: if you want it to fly for a specified period of time, set this to the number of seconds (fractions are fine) or use None to send the command once. Note, if you do this, you will need an outside loop that sends lots of commands or your drone will not fly very far. The command is not repeated inside the drone. it executes once and goes back to hovering without new commands coming in. But the option of zero duration allows for smoother flying if you want to do the control loop yourself.

Parameters

- **roll** – roll speed in -100 to 100
- **pitch** – pitch speed in -100 to 100
- **yaw** – yaw speed in -100 to 100
- **vertical_movement** – vertical speed in -100 to 100
- **duration** – optional: seconds for a specified duration or None to send it once (see note above)

Returns

hover()
Sends the command execute a flat trim to the mambo. This is basically a hover command. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

is_landed()
Returns true if it is landed or emergency and False otherwise :return:

land()
Sends the land command to the mambo. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

open_claw()
Open the claw - note not supposed under wifi since the camera takes the place of the claw

Returns True if the command was sent and False otherwise (can include errors or asking to do this using wifi)

safe_emergency(*timeout*)
Sends emergency stop command until the Mambo reports it is not flying anymore

Parameters **timeout** – quit trying to emergency stop if it takes more than timeout seconds

safe_land(*timeout*)
Ensure the mambo lands by sending the command until it shows landed on sensors

safe_takeoff(*timeout*)
Sends commands to takeoff until the mambo reports it is taking off

Parameters **timeout** – quit trying to takeoff if it takes more than timeout seconds

set_max_tilt(*value*)
Sets the maximum tilt in degrees. Ensures you only set positive values.

Parameters **value** – maximum tilt in degrees

Returns True if the command was sent and False otherwise

set_max_vertical_speed(*value*)
Sets the maximum vertical speed in m/s. Unknown what the true maximum is but we do ensure you only set positive values.

Parameters **value** – maximum speed

Returns True if the command was sent and False otherwise

set_user_sensor_callback(*function*, *args*)
Set the (optional) user callback function for sensors. Every time a sensor is updated, it calls this function.

Parameters

- **function** – name of the function
- **args** – tuple of arguments to the function

Returns nothing

smart_sleep(*timeout*)
Don't call time.sleep directly as it will mess up BLE and miss WIFI packets! Use this which handles packets received while sleeping

Parameters **timeout** – number of seconds to sleep

take_picture()
Ask the drone to take a picture also checks how many frames are on there, if there are ore than 35 it deletes one If connected via Wifi it If it is connected via WiFi it also deletes all frames on the Mambo once there are more than 35, since after there are 40 the next ones are ignored :return: True if the command was sent and False otherwise

takeoff()

Sends the takeoff command to the mambo. Gets the codes for it from the xml files. Ensures the packet was received or sends it again up to a maximum number of times.

Returns True if the command was sent and False otherwise

turn_degrees(*degrees*)

Turn the mambo the specified number of degrees (-180, 180). Degrees must be an integere so it is cast to an integer here. If you send it a float, it will be rounded according to the rules of int()

This is called cap in the xml but it means degrees per http://forum.developer.parrot.com/t/what-does-cap-stand-for/6213/2

Parameters **degrees** – degrees to turn (-180 to 180)

Returns True if the command was sent and False otherwise

turn_on_auto_takeoff()

Turn on the auto take off (throw mode) :return: True if the command was sent and False otherwise

update_sensors(*data_type*, *buffer_id*, *sequence_number*, *raw_data*, *ack*)

Update the sensors (called via the wifi or ble connection)

Parameters

- **data** – raw data packet that needs to be parsed
- **ack** – True if this packet needs to be ack'd and False otherwise

class pyparrot.Minidrone.**MinidroneSensors**

Bases: `object`

Store the minidrone's last known sensor values

get_estimated_z_orientation()

Uses the quaternions to return an estimated orientation

Learn more about unit quaternions here:

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

NOTE: This is not a real compass heading. 0 degrees is where you are facing when the mambo turns on!

Returns

quaternion_to_euler_angle(*w*, *x*, *y*, *z*)

This code is directly from:

https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles

Parameters

- **x** –
- **y** –
- **z** –

Returns

set_user_callback_function(*function*, *args*)

Sets the user callback function (called everytime the sensors are updated)

Parameters

- **function** – name of the user callback function
- **args** – arguments (tuple) to the function

Returns

update(*name*, *value*, *sensor_enum*)
Update the sensor

Parameters

- **name** – name of the sensor to update
- **value** – new value for the sensor
- **sensor_enum** – enum list for the sensors that use enums so that we can translate from numbers to strings

Returns

class pyparrot.Minidrone.**Swing**(*address=''*, *use_wifi=False*)
Bases: *pyparrot.Minidrone.Minidrone*

disconnect()
Disconnect the BLE connection. Always call this at the end of your programs to cleanly disconnect.

Returns void

set_flying_mode(*mode*)
Set drone flying mode

Parameters **state** –

Returns

set_plane_gear_box(*state*)
Set plane gear box

Parameters **state** –

Returns

pyparrot.VisionServer module

This is a simple web server to let the user see the vision that is being processed by ffmpeg. It essentially replaces the role of VLC. Note that there are several user parameters that should be set to run this program.

This does not replace the vision process! This is a separate process just to run a web server that lets you see what the mambo is seeing.

orig author: Igor Maculan - n3wtron@gmail.com A Simple mjpg stream http server

Updated for python3 including png streaming and graceful error handling - Taner Davis

pyparrot.VisionServer.**HOST_NAME = '127.0.0.1'**
The port youd like to stream to locally. Unless you have a strong reason to change this, keep this as 9090.

pyparrot.VisionServer.**IMAGE_PATH = '/Users/amy/repos/pyparrot/images/image_%03d.png'**
The URL or website name you would like to stream to. Unless you have a strong reason to change this, keep this as "127.0.0.1"

pyparrot.VisionServer.**PORT_NUMBER = 9090**

Set this according to how we want to stream: Stream in color -> >0 (1, 2, 873, etc.) Stream in black and white -> 0 Stream in color with transparency -> <0 (-1, -6, -747, etc.)

pyparrot.VisionServer.**main**()
Builds an http page to see images in a stream as they are created in the IMAGE_PATH specified above.

Module contents

CHAPTER 2

Indices and tables

- modindex is a good place to start if you want to read API docs
- genindex of ALL functions (warning, this is huge and overwhelming)
- search

Python Module Index

Index