pyCarla

Release 0.1

Federico Simonetta

Jul 27, 2021

TABLE OF CONTENTS

A python module for synthesizing MIDI events and files from python code using any kind of audio plugin!

A python module based on `carla` and `jack`!

CHAPTER

ONE

INSTALLATION

The backbone of this project are the multiple dependencies on which it depends. Since it's difficult to provide a script to automatically install all of these dependencies, here is a little handbook about how to install them.

1.1 TLDR

1. Use Linux: it's free. For Windows and Mac, you can still install Carla and Jack by yourself; however, I refuse to support non-free software.
2. In general, use https://pkgs.org to look for the command needed in your distro.
3. Install: `jackd` 1.9
4. Make sure that it is available in your `PATH` environment variable

1.2 1. Installing pycarla

```
pip install --upgrade pip pycarla
```

1.3 2. Installing jack

1. Ubuntu/Debian based: `sudo apt-get install jackd2`
2. Arch based: `sudo pacman -Sy jack2`
3. Gentoo based: `sudo emerge -a media-sound/jack2`
4. Fedora based: `sudo dnf install jack-audio-connection-kit`

For other Os, pre-built binaries are available at https://jackaudio.org/downloads/

1.4 3. Installing Carla

After having installed the package, run `python -m pycarla.carla --download` to download the correct version of Carla.

If you're not in Linux, pre-built binaries for major OS available at https://github.com/falkTX/Carla/releases/latest

N.B. Configure Carla in ``patchbay`` mode (if you cannot use GUI, set ``ProcessMode=3`` into ``~/.config/falkTX/Carla2.conf``)

CHAPTER

TWO

DEVELOPMENT SETUP

1. Install poetry: `curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py | python`
2. Enter root directory of this project
3. `poetry update`
4. Put all the Carla configurations that you want to use in `data/carla_proj` Note that you can use the default ones, provided you have the same plugins available, otherwise you have to delete the default project files.

Used plugins are:

- Pianoteq
- SalamanderGrandPianoV3 uncompressed in `~/salamander/`
- Calf Reverb

1. Run `poetry run -m pycarla <a_midi_file.mid>` to do a little test

CHAPTER

THREE

USAGE

3.1 Carla presets

1. **Configure Carla in ``patchbay`` mode (if you cannot use GUI, set ``ProcessMode=3`` into ``~/.config/falkTX/Carla2.conf``)**
2. `python -m pycarla.carla --run` to launch Carla and prepare configurations

3.2 Initialization

```
from pycarla import Carla, MIDIPlayer, AudioRecorder, get_smf_duration
carla = Carla("carla_project.carxp", ['-R', '-d', 'alsa'], min_wait=4)
carla.start()

player = MIDIPlayer()
recorder = AudioRecorder()

# or
with MIDIPlayer() as player, AudioRecorder() as recorder:
    # [...]
    pass
```

3.3 Playing and recording one note

```
print("Playing and recording one note..")
duration = 2
pitch = 64
recorder.start(duration + FINAL_DECAY)
player.synthesize_midi_note(pitch, 64, duration, 0, sync=True)
recorder.wait()
audio = recorder.recorded
if not np.any(audio):
    print("Error, no sample != 0")
    carla.kill() # this kills both Carla and Jack
    # carla.kill_carla() # this kills Carla but not Jack
    sys.exit()
```

3.4 Playing and recording a full MIDI file

```
print("Playing and recording full file using freewheeling mode..")
duration = get_smf_duration("filename.mid")
# in the following, `condition` ensures that both the recorder and player
# start in the same cycle
recorder.start(duration + FINAL_DECAY, condition=player.is_ready)
player.synthesize_midi_file("filename.mid",
    condition=recorder.is_ready, in_fw=True, out_fw=True)
# or asynchronously:
# player.synthesize_midi_file("filename.mid", sync=False)
# in this case, use
# player.wait(in_fw=True, out_fw=True)
recorder.wait(in_fw=True, out_fw=True)
recorder.save_recorded("session.wav")
player.close()
server.close()
```

In future, there shold be a function that does this snippet for you

You can also use `AudioRecorder` and `MIDIPlayer` as context managers in a `with` block; in this case, skip the *close()* at the end:

```
with pycarla.AudioRecorder() as recorder, pycarla.MIDIPlayer() as player:
    # do your stuffs
    pass
```

3.5 Closing server

```
try:
    carla.kill()
except Exception as e:
    print("Processes already closed!")
```

CHAPTER

FOUR

CLASSES AND FUNCTIONS

4.1 Carla

class pycarla.carla.**Carla**(*proj_path: str*, *server_options: List[str] = []*, *min_wait: float = 0*, *nogui: bool = True*)

__make_carla_popen(*proj_path*)

exists(*ports=['Carla:events*', 'Carla:audio*']*)
simply checks if the Carla process is running and ports are available

***ports* is a list of string name representing Jack ports; you can use** '*', '?' etc.

Returns **bool** – running, false otherwise

Return type True if all ports in *ports* exist and the Carla process is

get_ports()

kill()
kill carla and wait for the server

kill_carla()
kill carla, but not the server

restart()
Restarts both the server and Carla!

restart_carla()
Only restarts Carla, not the Jack server!

start()
Start carla and Jack and wait *self.min_wait* seconds after a Carla instance is ready.

wait_exists()
Waits until a Carla instance is ready in Jack

pycarla.carla.**download**()

pycarla.carla.**run_carla**()

4.2 Jack Server

class pycarla.jackserver.**JackServer**(*options*)

kill()
Just calls *self.process.kill()* and reset this object

restart()
Wait for the duration of this *ExternalProcess*, then kill and restart. If the duration is not set, it doesn't return

start()
Starts the server if not already started

4.3 Playing MIDI

class pycarla.midiplayer.**MIDIPlayer**

MIDI_PORT = 'Carla'

activate()
Activate the MIDI player client and set the connections.

If the Carla instance is not found, this method rase a *RuntimeWarning*. To avoid it, use `Carla.exists` method. Note that `Carla.start` already does that!

clear()
clears the *_messages* list

synthesize_messages(*messages: List[mido.Message], sync=False, condition=<function MIDIPlayer.<lambda>>, **kwargs*)
Synthesize a list of messages

1. Connect the port of this jack client to Carla if not yet done
2. Send the list of messages to the Carla instance

If *sync* is True, this function waits until all messages have been processed, otherwise, it suddenly returns. You can wait by calling the *wait* method of this object.

This function is compatible with freewheeling mode. Freewheel prevents jack from waiting between return calls. This allows for the maximum allowed speed, but not output/input operation is done with system audio (i.e. you cannot listen/recording to anything while in freewheeling mode).

condition is a function checked in the playing callback. If *condition()* is False, no message is sent. The callback start playing at the cycle after the one in which *condition()* becomes True.

kwargs are passed to *wait* if *sync* is True.

Note: Mido numbers channels 0 to 15 instead of 1 to 16. This makes them easier to work with in Python but you may want to add and subtract 1 when communicating with the user.

synthesize_midi_file(*midifile: Any*, ***kwargs*) → multiprocessing.context.Process
Send midi messages contained in *filename* using *self.synthesize_messages*. All keywords from that method can be used here.

midifile can be a *mido.MidiFile* object or a string

After the playback, ports are resetted

synthesize_midi_note(*pitch: int*, *velocity: int*, *duration: float*, *sustain: int = 0*, *soft: int = 0*, *sostenuto: int = 0*, *channel: int = 0*, *program: int = 0*, ***kwargs*) → multiprocessing.context.Process

set up a list of messages representing one note and then calls *self.synthesize_messages*. All keywords from that method can be used here.

4.4 Recording Audio

class pycarla.audiorecorder.**AudioRecorder**

AUDIO_PORT = 'Carla'

activate()

Activate the recording client and set the connections. Set self.channels and create one input port per each Carla output port.

If the Carla instance is not found, this method rase a *RuntimeWarning*. To avoid it, use `Carla.exists` method. Note that `Carla.start` already does that!

clear()

Clears the *recorded* array

save_recorded(*filename*)

Save the recorded array to file. Extensions supported by `libsndfile`!

start_frame is the frame from which recorded is saved (use it to discard initial delays due to Jack setup).

start(*duration=None*, *sync=False*, *condition=<function AudioRecorder.<lambda>>*, ***kwargs*)

Record audio for `duration` seconds. Note that this function blocks if *sync* is True, otherwise, this returns suddenly and you should wait/stop by calling the *wait* method of this object which constructs the recorded array in *self.recorded*

condition is a function checked in the recording callback. If *condition()* is False, blocks are discarded. The callback start recording at the cycle after the one in which *condition()* becomes True.

This function is compatible with Jack freewheeling mode to record offline sessions.

kwargs are passed to *wait* if *sync* is True.

wait(*timeout=None*, *in_fw=False*, *out_fw=False*)

Wait until recording is finished. If *timeout* is a number, it should be the maximum number of seconds until which the recording stops. A boolean is returned representing if timeout is reached. (returns *False* if timeout is not set)

The recording stops when *timeout* or the duration passed when calling *start* is reached. In these cases, the recording client is deactivated and the callback stopped.

waits while setting freewheeling mode to *in_fw* it then set freewheeling mode to *out_fw* before exiting

CHAPTER

FIVE

WHY SO MANY EXTERNAL DEPENDENCIES?

Python has no strong real-time capabilities since it cannot run with parallel threads. This method delegates most of the realtime stuffs to external C/C++ programs, improving the performances and the accuracy against pure-Python based approaches. Namely, the synthesis and the management of plugins is delegated to Carla, while the MIDI messaging and audio recording is done in python using C Jack API.

This method is really portable and supports almost any type of plugins and virtual instruments thanks to the excellent Carla:

1. Linux VST2/VST3
2. Windows VST2/VST3
3. LV2
4. LADSPA
5. DSSI
6. AU
7. SF2/SF3
8. SFZ
9. Any other format supported by external plugins

CHAPTER

SIX

CREDITS

1. **Federico Simonetta** federico.simonetta at unimi.it

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

G

J

K

M

P

R

S

W