PedalPi - PluginsManager Documentation

Release 1

SrMouraSilva

Sep 17, 2017

Contents

1 Install **3**

2 Example **5**

3 Observer **9**

4 Changelog **11**

5 Maintenance **13**
- 5.1 Test . . . 13
- 5.2 Generate documentation . . . 13

6 API **15**
- 6.1 PedalPi - PluginsManager - Jack . . . 15
- 6.2 PedalPi - PluginsManager - ModHost . . . 15
- 6.3 PedalPi - PluginsManager - Models . . . 22
- 6.4 PedalPi - PluginsManager - Model - Lv2 . . . 33
- 6.5 PedalPi - PluginsManager - Model - System . . . 36
- 6.6 PedalPi - PluginsManager - Observers . . . 37
- 6.7 PedalPi - PluginsManager - Util . . . 44

Pythonic management of LV2 audio plugins with mod-host.

Documentation: http://pedalpi-pluginsmanager.readthedocs.io/

Code: https://github.com/PedalPi/PluginsManager

Python Package Index: https://pypi.org/project/PedalPi-PluginsManager

License: Apache License 2.0

CHAPTER 1

Install

Plugin Manager has dependencies that must be installed before installing the library. Among the dependencies are lv2ls to check the installed audio plugins and PortAudio for information on the audio interfaces through PyAudio.

On Debian-based systems, run:

```
sudo apt-get install -y portaudio19-dev python-all-dev lilv-utils --no-install-
↪recommends
```

Of course, for PluginsManager to manage Lv2 audio plugins, it is necessary that they have installed audio plugins to be managed. The Guitarix and Calf Studio projects provide some audio plugins. To install them:

```
.. code-block:: bash
```

pip install PedalPi-PluginsManager

CHAPTER 2

Example

Note: Other examples are in the examples folder in the repository.

This examples uses Calf and Guitarix audio plugins.

Download and install mod-host. For more information, check the ModHost section.

Start audio process

```
# In this example, is starting a Zoom g3 series audio interface
jackd -R -P70 -t2000 -dalsa -dhw:Series -p256 -n3 -r44100 -s &
mod-host
```

Play!

```
from pluginsmanager.banks_manager import BanksManager
from pluginsmanager.observer.mod_host.mod_host import ModHost

from pluginsmanager.model.bank import Bank
from pluginsmanager.model.pedalboard import Pedalboard
from pluginsmanager.model.connection import Connection

from pluginsmanager.model.lv2.lv2_effect_builder import Lv2EffectBuilder

from pluginsmanager.model.system.system_effect import SystemEffect
```

Creating a bank

```
# BanksManager manager the banks
manager = BanksManager()

bank = Bank('Bank 1')
manager.append(bank)
```

Connecting with mod_host. Is necessary that the mod_host process already running

```
mod_host = ModHost('localhost')
mod_host.connect()
manager.register(mod_host)
```

Creating pedalboard

```
pedalboard = Pedalboard('Rocksmith')
bank.append(pedalboard)
# or
# bank.pedalboards.append(pedalboard)
```

Loads pedalboard. All changes in pedalboard are reproduced in mod_host

```
mod_host.pedalboard = pedalboard
```

Add effects in the pedalboard

```
builder = Lv2EffectBuilder()

reverb = builder.build('http://calf.sourceforge.net/plugins/Reverb')
fuzz = builder.build('http://guitarix.sourceforge.net/plugins/gx_fuzz_#fuzz_')
reverb2 = builder.build('http://calf.sourceforge.net/plugins/Reverb')

pedalboard.append(reverb)
pedalboard.append(fuzz)
pedalboard.append(reverb2)
# or
# pedalboard.effects.append(reverb2)
```

For obtains automatically the sound card inputs and outputs, use *SystemEffectBuilder*. It requires a *JackClient* instance, that uses JACK-Client.

```
from pluginsmanager.jack.jack_client import JackClient
client = JackClient()

from pluginsmanager.model.system.system_effect_builder import SystemEffectBuilder
sys_effect = SystemEffectBuilder(client)
```

For manual input and output sound card definition, use:

```
sys_effect = SystemEffect('system', ['capture_1', 'capture_2'], ['playback_1',
↪'playback_2'])
```

Note: NOT ADD sys_effect in any Pedalboard

Connecting *mode one*:

```
sys_effect.outputs[0].connect(reverb.inputs[0])

reverb.outputs[0].connect(fuzz.inputs[0])
reverb.outputs[1].connect(fuzz.inputs[0])
fuzz.outputs[0].connect(reverb2.inputs[0])
reverb.outputs[0].connect(reverb2.inputs[0])

reverb2.outputs[0].connect(sys_effect.inputs[0])
reverb2.outputs[0].connect(sys_effect.inputs[1])
```

Connecting *mode two*:

```
pedalboard.connections.append(Connection(sys_effect.outputs[0], reverb.inputs[0]))

pedalboard.connections.append(Connection(reverb.outputs[0], fuzz.inputs[0]))
pedalboard.connections.append(Connection(reverb.outputs[1], fuzz.inputs[0]))
pedalboard.connections.append(Connection(fuzz.outputs[0], reverb2.inputs[0]))
pedalboard.connections.append(Connection(reverb.outputs[0], reverb2.inputs[0]))

pedalboard.connections.append(Connection(reverb2.outputs[0], sys_effect.inputs[0]))
pedalboard.connections.append(Connection(reverb2.outputs[0], sys_effect.inputs[1]))
```

Warning: If you need connect system_output with system_input directly (for a bypass, as example), only the second mode will works:

```
pedalboard.connections.append(Connection(sys_effect.outputs[0], sys_effect.
↪inputs[0]))
```

Set effect status (enable/disable bypass) and param value

```
fuzz.toggle()
# or
# fuzz.active = not fuzz.active

fuzz.params[0].value = fuzz.params[0].minimum / fuzz.params[0].maximum

fuzz.outputs[0].disconnect(reverb2.inputs[0])
# or
# pedalboard.connections.remove(Connection(fuzz.outputs[0], reverb2.inputs[0]))
# or
# index = pedalboard.connections.index(Connection(fuzz.outputs[0], reverb2.inputs[0]))
# del pedalboard.connections[index]

reverb.toggle()
```

Removing effects and connections:

```
pedalboard.effects.remove(fuzz)

for connection in list(pedalboard.connections):
    pedalboard.connections.remove(connection)

for effect in list(pedalboard.effects):
    pedalboard.effects.remove(effect)
# or
# for index in reversed(range(len(pedalboard.effects))):
    # del pedalboard.effects[index]
```

CHAPTER 3

Observer

`ModHost` is an **observer** (see `UpdatesObserver`). It is informed about all changes that occur in some model instance (`BanksManager`, `Bank`, `Pedalboard`, `Effect`, `Param`, ...), allowing it to communicate with the `mod-host` process transparently.

It is possible to create observers! Some ideas are:

- Allow the use of other hosts (such as Carla);
- Automatically persist changes;
- Automatically update a human-machine interface (such as LEDs and displays that inform the state of the effects).

How to implement and the list of Observers implemented by this library can be accessed in the Observer section.

CHAPTER 4

Changelog

..include:: ../../CHANGES

CHAPTER 5

Maintenance

Test

It is not necessary for the mod_host process to be running

```
coverage3 run --source=pluginsmanager setup.py test

coverage3 report
coverage3 html
firefox htmlcov/index.html
```

Generate documentation

This project uses Sphinx + Read the Docs.

You can generate the documentation in your local machine:

```
pip3 install sphinx

cd docs
make html

firefox build/html/index.html
```

CHAPTER 6

API

Contents:

PedalPi - PluginsManager - Jack

pluginsmanager.jack.jack_client.JackClient

pluginsmanager.jack.jack_interface.JackInterfaces

pluginsmanager.jack.jack_interface.AudioInterface

PedalPi - PluginsManager - ModHost

About *mod-host*

mod-host is a LV2 host for Jack controllable via socket or command line. With it you can load audio plugins, connect, manage plugins.

For your use, is necessary download it

```
git clone https://github.com/moddevices/mod-host
cd mod-host
make
make install
```

Then boot the JACK process and start the *mod-host*. Details about "JACK" can be found at https://help.ubuntu.com/community/What%20is%20JACK

```
# In this example, is starting a Zoom g3 series audio interface
jackd -R -P70 -t2000 -dalsa -dhw:Series -p256 -n3 -r44100 -s &
mod-host
```

You can now connect to the mod-host through the Plugins Manager API. Create a ModHost object with the address that is running the *mod-host* process. Being in the same machine, it should be *'localhost'*

```
mod_host = ModHost('localhost')
mod_host.connect()
```

Finally, register the mod-host in your BanksManager. Changes made to the current pedalboard will be applied to *mod-host*

```
manager = BanksManager()
# ...
manager.register(mod_host)
```

To change the current pedalboard, change the *pedalboard* parameter to *mod_host*. Remember that for changes to occur in *mod-host*, the *pedalboard* must belong to some *bank* of *banks_manager*.

```
mod_host.pedalboard = my_awesome_pedalboard
```

ModHost

class `pluginsmanager.observer.mod_host.mod_host.`**`ModHost`**(*address='localhost'*, *port=5555*)

Python port for mod-host Mod-host is a LV2 host for Jack controllable via socket or command line.

This class offers the mod-host control in a python API:

```
# Create a mod-host, connect and register it in banks_manager
mod_host = ModHost('localhost')
mod_host.connect()
banks_manager.register(mod_host)

# Set the mod_host pedalboard for a pedalboard that the bank
# has added in banks_manager
mod_host.pedalboard = my_awesome_pedalboard
```

The changes in current pedalboard (`pedalboard` attribute of *mod_host*) will also result in mod-host:

```
driver = my_awesome_pedalboard.effects[0]
driver.active = False
```

Note: For use, is necessary that the mod-host is running, for use, access

- Install dependencies
- Building mod-host
- Running mod-host

For more JACK information, access Demystifying JACK – A Beginners Guide to Getting Started with JACK

Example:

In this example, is starting a Zoom G3 series audio interface. Others interfaces maybe needs others configurations.

```
# Starting jackdump process via console
jackd -R -P70 -t2000 -dalsa -dhw:Series -p256 -n3 -r44100 -s &
# Starting mod-host
mod-host &
```

Parameters

- **address** (*string*) – Computer mod-host process address (IP). If the process is running on the same computer that is running the python code uses *localhost*.
- **port** (*int*) – Socket port on which mod-host should be running. Default is *5555*

__del__()

Calls `close()` method for remove the audio plugins loaded and closes connection with mod-host.

```
>>> mod_host = ModHost()
>>> del mod_host
```

Note: If the mod-host process has been created with `start()` method, it will be finished.

close()

Remove the audio plugins loaded and closes connection with mod-host.

Note: If the mod-host process has been created with `start()` method, it will be finished.

connect()

Connect the object with mod-host with the _address_ parameter informed in the constructor method (`__init__()`)

pedalboard

Currently managed pedalboard (current pedalboard)

Getter Current pedalboard - Pedalboard loaded by mod-host

Setter Set the pedalboard that will be loaded by mod-host

Type Pedalboard

start()

Invokes the mod-host process.

mod-host requires JACK to be running. mod-host does not startup JACK automatically, so you need to start it before running mod-host.

Note: This function is experimental. There is no guarantee that the process will actually be initiated.

ModHost internal

The classes below are for internal use of mod-host

Connection

class pluginsmanager.observer.mod_host.connection.**Connection**(*socket_port=5555*, *address='localhost'*)

Class responsible for managing an API connection to the mod-host process via socket

close()

Closes socket connection

send(*message*)

Sends message to *mod-host*.

Note: Uses *ProtocolParser* for a high-level management. As example, view *Host*

Parameters **message** (*string*) – Message that will be sent for *mod-host*

Host

class pluginsmanager.observer.mod_host.host.**Host**(*address='localhost'*, *port=5555*)

Bridge between *mod-host* API and *mod-host* process

add(*effect*)

Add an LV2 plugin encapsulated as a jack client

Parameters **effect** (Lv2Effect) – Effect that will be loaded as LV2 plugin encapsulated

close()

Quit the connection with mod-host

connect(*connection*)

Connect two effect audio ports

Parameters **connection** (pluginsmanager.model.connection.Connection) – Connection with the two effect audio ports (output and input)

disconnect(*connection*)

Disconnect two effect audio ports

Parameters **connection** (pluginsmanager.model.connection.Connection) – Connection with the two effect audio ports (output and input)

quit()

Quit the connection with mod-host and stop the mod-host process

remove(*effect*)

Remove an LV2 plugin instance (and also the jack client)

Parameters **effect** (Lv2Effect) – Effect that your jack client encapsulated will removed

set_param_value(*param*)

Set a value to given control

Parameters **param** (Lv2Param) – Param that the value will be updated

set_status(*effect*)

Toggle effect processing

Parameters **effect** (Lv2Effect) – Effect with the status updated

ProtocolParser

class pluginsmanager.observer.mod_host.protocol_parser.**ProtocolParser**

Prepare the objects to mod-host string command

static add(*effect*)

add <lv2_uri> <instance_number>

add a LV2 plugin encapsulated as a jack client

e.g.:

```
add http://lv2plug.in/plugins/eg-amp 0
```

instance_number must be any value between 0 ~ 9999, inclusively

Parameters **effect** (Lv2Effect) – Effect will be added

static bypass(*effect*)

bypass <instance_number> <bypass_value>

toggle plugin processing

e.g.:

```
bypass 0 1
```

- if bypass_value = 1 bypass plugin
- if bypass_value = 0 process plugin

Parameters **effect** (Lv2Effect) – Effect that will be active the bypass or disable the bypass

static connect(*connection*)

connect <origin_port> <destination_port>

connect two plugin audio ports

e.g.:

```
connect system:capture_1 plugin_0:in
```

Parameters **connection** (pluginsmanager.model.connection.Connection) – Connection with a valid *Output* and *Input*

static disconnect(*connection*)

disconnect <origin_port> <destination_port>

disconnect two plugin audio ports

e.g.:

```
disconnect system:capture_1 plugin_0:in
```

Parameters **connection** (pluginsmanager.model.connection.Connection) – Connection with a valid *Output* and *Input*

static help()
`help`

show a help message

static load(*filename*)
`load <file_name>`

load a history command file dummy way to save/load workspace state

e.g.:

```
load my_setup
```

Note: Not implemented yet

static midi_learn(*plugin*, *param*)
`midi_learn <instance_number> <param_symbol>`

This command maps starts MIDI learn for a parameter

e.g.:

```
midi_learn 0 gain
```

Note: Not implemented yet

static midi_map(*plugin*, *param*, *midi_chanel*, *midi_cc*)
`midi_map <instance_number> <param_symbol> <midi_channel> <midi_cc>`

This command maps a MIDI controller to a parameter

e.g.:

```
midi_map 0 gain 0 7
```

Note: Not implemented yet

static midi_unmap(*plugin*, *param*)
`midi_unmap <instance_number> <param_symbol>`

This command unmaps the MIDI controller from a parameter

e.g.:

```
unmap 0 gain
```

Note: Not implemented yet

static monitor()
`monitor <addr> <port> <status>`

open a socket port to monitoring parameters

e.g.:

```
monitor localhost 12345 1
```

- if `status = 1` start monitoring
- if `status = 0` stop monitoring

Note: Not implemented yet

static `param_get` (*param*)

`param_get <instance_number> <param_symbol>`

get the value of the request control

e.g.:

```
param_get 0 gain
```

Parameters **`param`** (`Lv2Param`) – Parameter that will be get your current value

static `param_monitor`()

`param_monitor <instance_number> <param_symbol> <cond_op> <value>`

do monitoring a plugin instance control port according given condition

e.g.:

```
param_monitor 0 gain > 2.50
```

Note: Not implemented yet

static `param_set` (*param*)

`param_set <instance_number> <param_symbol> <param_value>`

set a value to given control

e.g.:

```
param_set 0 gain 2.50
```

Parameters **`param`** (`Lv2Param`) – Parameter that will be updated your value

static `preset_load`()

`preset_load <instance_number> <preset_uri>`

load a preset state to given plugin instance

e.g.:

```
preset_load 0 "http://drobilla.net/plugins/mda/presets#JX10-moogcury-lite"
```

Note: Not implemented yet

static preset_save()

preset_save <instance_number> <preset_name> <dir> <file_name>

save a preset state from given plugin instance

e.g.:

```
preset_save 0 "My Preset" /home/user/.lv2/my-presets.lv2 mypreset.ttl
```

Note: Not implemented yet

static preset_show()

preset_show <instance_number> <preset_uri>

show the preset information of requested instance / URI

e.g.:

```
preset_show 0 http://drobilla.net/plugins/mda/presets#EPiano-bright
```

Note: Not implemented yet

static quit()

quit

bye!

static remove(*effect*)

remove <instance_number>

remove a LV2 plugin instance (and also the jack client)

e.g.:

```
remove 0
```

Parameters **effect** (Lv2Effect) – Effect will be removed

static save(*filename*)

save <file_name>

saves the history of typed commands dummy way to save/load workspace state

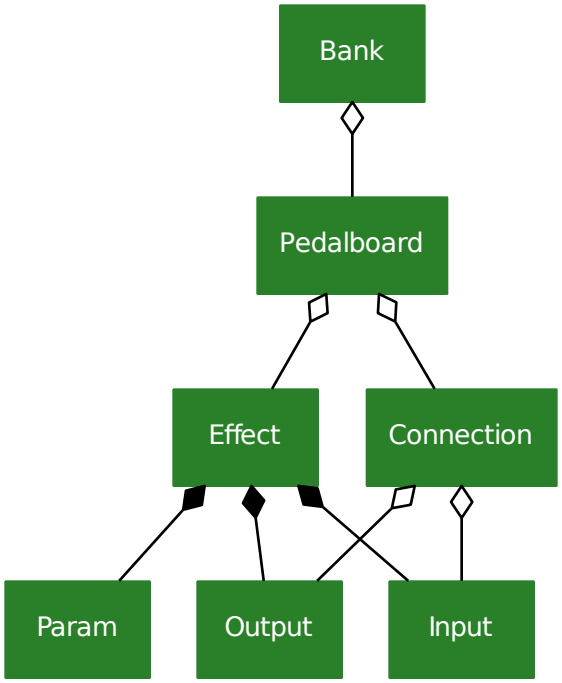
e.g.:

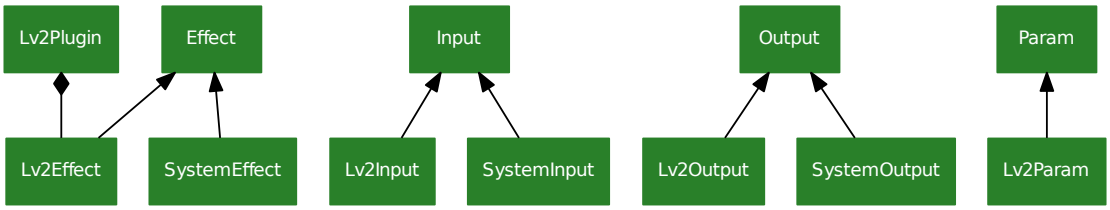
```
save my_setup
```

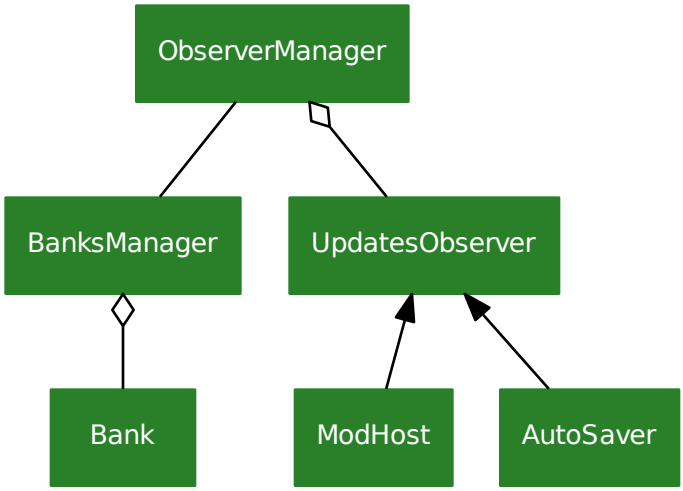
Note: Not implemented yet

PedalPi - PluginsManager - Models

This page contains the model classes.


Bank
Pedalboard
Effect
Connection
Param
Output
Input


Lv2Plugin
Effect
Input
Output
Param
Lv2Effect
SystemEffect
Lv2Input
SystemInput
Lv2Output
SystemOutput
Lv2Param



BanksManager

class pluginsmanager.banks_manager.**BanksManager**(*banks=None*)

BanksManager manager the banks. In these is possible add banks, obtains the banks and register observers for will be notified when occurs changes (like added new pedalboard, rename bank, set effect param value or state)

For use details, view Readme.rst example documentation.

Parameters **banks** (*list[Bank]*) – Banks that will be added in this. Useful for loads banks previously loaded, like banks persisted and recovered.

__iter__()

Iterates banks of the banksmanager:

```
>>> for index, bank in enumerate(banks_manager):
>>>     print(index, '-', bank)
```

Returns Iterator for banks list

append(*bank*)

Append the bank in banks manager. It will be monitored, changes in this will be notified for the notifiers.

Parameters **bank** (Bank) – Bank that will be added in this

enter_scope(*observer*)

Informs that changes occurs by the observer and isn't necessary informs the changes for observer

Parameters **observer** (UpdatesObserver) – Observer that causes changes

exit_scope()

Closes the last observer scope added

observers

Returns Observers registered in BanksManager instance

register(*observer*)
Register an observer for it be notified when occurs changes.

For more details, see *UpdatesObserver*

Parameters **observer** (UpdatesObserver) – Observer that will be notified then occurs changes

unregister(*observer*)
Remove the observers of the observers list. It will not receive any more notifications when occurs changes.

Parameters **observer** (UpdatesObserver) – Observer you will not receive any more notifications then occurs changes.

Bank

class pluginsmanager.model.bank.**Bank**(*name*)
Bank is a data structure that contains *Pedalboard*. It's useful for group common pedalboards, like "Pedalboards will be used in the Sunday show"

A fast bank overview:

```
>>> bank = Bank('RHCP')
>>> californication = Pedalboard('Californication')
```

```
>>> # Add pedalboard in bank - mode A
>>> bank.append(californication)
>>> californication.bank == bank
True
```

```
>>> bank.pedalboards[0] == californication
True
```

```
>>> # Add pedalboard in bank - mode B
>>> bank.pedalboards.append(Pedalboard('Dark Necessities'))
>>> bank.pedalboards[1].bank == bank
True
```

```
>>> # If you needs change pedalboards order (swap), use pythonic mode
>>> bank.pedalboards[1], bank.pedalboards[0] = bank.pedalboards[0], bank.
↪pedalboards[1]
>>> bank.pedalboards[1] == californication
True
```

```
>>> # Set pedalboard
>>> bank.pedalboards[0] = Pedalboard("Can't Stop")
>>> bank.pedalboards[0].bank == bank
True
```

```
>>> del bank.pedalboards[0]
>>> bank.pedalboards[0] == californication # Pedalboard Can't stop rermoved,␣
↪first is now the californication
True
```

You can also toggle pedalboards into different banks:

```
>>> bank1.pedalboards[0], bank2.pedalboards[2] = bank2.pedalboards[0], bank1.
↪pedalboards[2]
```

Parameters **name** (*string*) – Bank name

append(*pedalboard*)
Add a *Pedalboard* in this bank

This works same as:

```
>>> bank.pedalboards.append(pedalboard)
```

or:

```
>>> bank.pedalboards.insert(len(bank.pedalboards), pedalboard)
```

Parameters **pedalboard** (Pedalboard) – Pedalboard that will be added

index
Returns the first occurrence of the bank in your PluginsManager

json
Get a json decodable representation of this bank

Return dict json representation

Connection

class pluginsmanager.model.connection.**Connection**(*effect_output*, *effect_input*)
pluginsmanager.model.connection.Connection represents a connection between two distinct effects by your ports (effect *Output* with effect *Input*):

```
>>> californication = Pedalboard('Californication')
>>> californication.append(driver)
>>> californication.append(reverb)
```

```
>>> guitar_output = sys_effect.outputs[0]
>>> driver_input = driver.inputs[0]
>>> driver_output = driver.outputs[0]
>>> reverb_input = reverb.inputs[0]
>>> reverb_output = reverb.outputs[0]
>>> amp_input = sys_effect.inputs[0]
```

```
>>> # Guitar -> driver -> reverb -> amp
>>> californication.connections.append(Connection(guitar_output, driver_input))
>>> californication.connections.append(Connection(driver_output, reverb_input))
>>> californication.connections.append(Connection(reverb_output, amp_input))
```

Another way to use implicitly connections:

```
>>> guitar_output.connect(driver_input)
>>> driver_output.connect(reverb_input)
>>> reverb_output.connect(amp_input)
```

Parameters

- **effect_output** (Output) – Output port that will be connected with input port
- **effect_input** (Input) – Input port that will be connected with output port

input

Return Output Input connection port

json

Get a json decodable representation of this effect

Return dict json representation

output

Return Output Output connection port

Effect

class pluginsmanager.model.effect.**Effect**

Representation of a audio plugin instance - LV2 plugin encapsulated as a jack client.

Effect contains a *active* status (off=bypass), a list of *Param*, a list of *Input* and a list of pluginsmanager.mod_host.connection.Connection:

```
>>> reverb = builder.build('http://calf.sourceforge.net/plugins/Reverb')
>>> pedalboard.append(reverb)
>>> reverb
<Lv2Effect object as 'Calf Reverb' active at 0x7fd58d874ba8>

>>> reverb.active
True
>>> reverb.toggle()
>>> reverb.active
False
>>> reverb.active = True
>>> reverb.active
True

>>> reverb.inputs
(<Lv2Input object as In L at 0x7fd58c583208>, <Lv2Input object as In R at␣
↪0x7fd58c587320>)
>>> reverb.outputs
(<Lv2Output object as Out L at 0x7fd58c58a438>, <Lv2Output object as Out R at␣
↪0x7fd58c58d550>)
>>> reverb.params
(<Lv2Param object as value=1.5 [0.4000000059604645 - 15.0] at 0x7fd587f77908>,
↪<Lv2Param object as value=5000.0 [2000.0 - 20000.0] at 0x7fd587f7a9e8>,
↪<Lv2Param object as value=2 [0 - 5] at 0x7fd587f7cac8>, <Lv2Param object as␣
↪value=0.5 [0.0 - 1.0] at 0x7fd587f7eba8>, <Lv2Param object as value=0.25 [0.0 -␣
↪2.0] at 0x7fd58c576c88>, <Lv2Param object as value=1.0 [0.0 - 2.0] at␣
↪0x7fd58c578d68>, <Lv2Param object as value=0.0 [0.0 - 500.0] at 0x7fd58c57ae80>,
↪ <Lv2Param object as value=300.0 [20.0 - 20000.0] at 0x7fd58c57df98>, <Lv2Param␣
↪object as value=5000.0 [20.0 - 20000.0] at 0x7fd58c5810f0>)
```

Parameters **pedalboard** (Pedalboard) – Pedalboard where the effect lies.

active
Effect status: active or bypass

Getter Current effect status

Setter Set the effect Status

Type bool

connections

Return list[Connection] Connections that this effects is present (with input or output port)

index
Returns the first occurrence of the effect in your pedalboard

inputs

Return list[Input] Inputs of effect

is_possible_connect_itself
return bool: Is possible connect the with it self?

json
Get a json decodable representation of this effect

Return dict json representation

outputs

Return list[Output] Outputs of effect

params

Return list[Param] Params of effect

toggle()
Toggle the effect status: `self.active = not self.active`

Input

class pluginsmanager.model.input.**Input**(*effect*)
Input is the medium in which the audio will go into effect to be processed.

Effects usually have a one (mono) or two inputs (stereo L + stereo R). But this isn't a rule: Some have only *Output*, like audio frequency generators, others have more than two.

For obtains the inputs:

```
>>> my_awesome_effect
<Lv2Effect object as 'Calf Reverb' active at 0x7fd58d874ba8>
>>> my_awesome_effect.inputs
(<Lv2Input object as In L at 0x7fd58c583208>, <Lv2Input object as In R at␣
↪0x7fd58c587320>)

>>> effect_input = my_awesome_effect.inputs[0]
>>> effect_input
<Lv2Input object as In L at 0x7fd58c583208>

>>> symbol = effect_input.symbol
>>> symbol
'in_l'
```

```
>>> my_awesome_effect.inputs[symbol] == effect_input
True
```

For connections between effects, view `pluginsmanager.mod_host.connection.Connection`.

Parameters **effect** (Effect) – Effect of input

effect

Returns Effect of input

index
:return Input index in the your effect

json
Get a json decodable representation of this input

Return dict json representation

symbol

Returns Input identifier

Output

class `pluginsmanager.model.output.`**Output**(*effect*)
Output is the medium in which the audio processed by the effect is returned.

Effects usually have a one (mono) or two outputs (stereo L + stereo R). .

For obtains the outputs:

```
>>> my_awesome_effect
<Lv2Effect object as 'Calf Reverb' active at 0x7fd58d874ba8>
>>> my_awesome_effect.outputs
(<Lv2Output object as Out L at 0x7fd58c58a438>, <Lv2Output object as Out R at␣
↪0x7fd58c58d550>)

>>> output = my_awesome_effect.outputs[0]
>>> output
<Lv2Output object as Out L at 0x7fd58c58a438>

>>> symbol = my_awesome_effect.outputs[0].symbol
>>> symbol
'output_l'

>>> my_awesome_effect.outputs[symbol] == output
True
```

For connections between effects, view `pluginsmanager.mod_host.connection.Connection`.

Parameters **effect** (Effect) – Effect that contains the output

connect(*effect_input*)
Connect it with effect_input:

```
>>> driver_output = driver.outputs[0]
>>> reverb_input = reverb.inputs[0]
>>> Connection(driver_output, reverb_input) in driver.effect.connections
False
>>> driver_output.connect(reverb_input)
```

```
>>> Connection(driver_output, reverb_input) in driver.effect.connections
True
```

Note: This method does not work for all cases. class:*SystemOutput* can not be connected with class:*SystemInput* this way. For this case, use

```
>>> pedalboard.connections.append(Connection(system_output, system_input))
```

Parameters **effect_input** (Input) – Input that will be connected with it

disconnect(*effect_input*)

Disconnect it with effect_input

```
>>> driver_output = driver.outputs[0]
>>> reverb_input = reverb.inputs[0]
>>> Connection(driver_output, reverb_input) in driver.effect.connections
True
>>> driver_output.disconnect(reverb_input)
>>> Connection(driver_output, reverb_input) in driver.effect.connections
False
```

Note: This method does not work for all cases. class:*SystemOutput* can not be disconnected with class:*SystemInput* this way. For this case, use

```
>>> pedalboard.connections.remove(Connection(system_output, system_input))
```

Parameters **effect_input** (Input) – Input that will be disconnected with it

effect

Returns Effect of output

index

:return Output index in the your effect

json

Get a json decodable representation of this output

Return dict json representation

symbol

Returns Output identifier

Param

class pluginsmanager.model.param.**Param**(*effect*, *default*)

Param represents an Audio Plugin Parameter:

```
>>> my_awesome_effect
<Lv2Effect object as 'Calf Reverb' active at 0x7fd58d874ba8>
>>> my_awesome_effect.params
```

```
(<Lv2Param object as value=1.5 [0.4000000059604645 - 15.0] at 0x7fd587f77908>,
↪<Lv2Param object as value=5000.0 [2000.0 - 20000.0] at 0x7fd587f7a9e8>,
↪<Lv2Param object as value=2 [0 - 5] at 0x7fd587f7cac8>, <Lv2Param object as␣
↪value=0.5 [0.0 - 1.0] at 0x7fd587f7eba8>, <Lv2Param object as value=0.25 [0.0 -␣
↪2.0] at 0x7fd58c576c88>, <Lv2Param object as value=1.0 [0.0 - 2.0] at␣
↪0x7fd58c578d68>, <Lv2Param object as value=0.0 [0.0 - 500.0] at 0x7fd58c57ae80>,
↪ <Lv2Param object as value=300.0 [20.0 - 20000.0] at 0x7fd58c57df98>, <Lv2Param␣
↪object as value=5000.0 [20.0 - 20000.0] at 0x7fd58c5810f0>)

>>> param = my_awesome_effect.params[0]
>>> param
<Lv2Param object as value=1.5 [0.4000000059604645 - 15.0] at 0x7fd587f77908>

>>> param.default
1.5
>>> param.value = 14

>>> symbol = param.symbol
>>> symbol
'decay_time'
>>> param == my_awesome_effect.params[symbol]
True
```

Parameters

- **effect** (Effect) – Effect in which this parameter belongs
- **default** – Default value (initial value parameter)

default

Default parameter value. Then a effect is instanced, the value initial for a parameter is your default value.

Getter Default parameter value.

effect

Returns Effect in which this parameter belongs

json

Get a json decodable representation of this param

Return dict json representation

maximum

Returns Greater value that the parameter can assume

minimum

Returns Smaller value that the parameter can assume

symbol

Returns Param identifier

value

Parameter value

Getter Current value

Setter Set the current value

Pedalboard

class pluginsmanager.model.pedalboard.**Pedalboard**(*name*)

Pedalboard is a patch representation: your structure contains *Effect* and pluginsmanager.mod_host.connection.Connection:

```
>>> pedalboard = Pedalboard('Rocksmith')
>>> bank.append(pedalboard)

>>> builder = Lv2EffectBuilder()
>>> pedalboard.effects
ObservableList: []
>>> reverb = builder.build('http://calf.sourceforge.net/plugins/Reverb')
>>> pedalboard.append(reverb)
>>> pedalboard.effects
ObservableList: [<Lv2Effect object as 'Calf Reverb'  active at 0x7f60effb09e8>]

>>> fuzz = builder.build('http://guitarix.sourceforge.net/plugins/gx_fuzzfacefm_#_
↪fuzzfacefm_')
>>> pedalboard.effects.append(fuzz)

>>> pedalboard.connections
ObservableList: []
>>> pedalboard.connections.append(Connection(sys_effect.outputs[0], fuzz.
↪inputs[0])) # View SystemEffect for more details
>>> pedalboard.connections.append(Connection(fuzz.outputs[0], reverb.inputs[0]))
>>> # It works too
>>> reverb.outputs[1].connect(sys_effect.inputs[0])
ObservableList: [<Connection object as 'system.capture_1 -> GxFuzzFaceFullerMod.In
↪' at 0x7f60f45f3f60>, <Connection object as 'GxFuzzFaceFullerMod.Out -> Calf␣
↪Reverb.In L' at 0x7f60f45f57f0>, <Connection object as 'Calf Reverb.Out R ->␣
↪system.playback_1' at 0x7f60f45dacc0>]

>>> pedalboard.data
{}
>>> pedalboard.data = {'my-awesome-component': True}
>>> pedalboard.data
{'my-awesome-component': True}
```

For load the pedalboard for play the songs with it:

```
>>> mod_host.pedalboard = pedalboard
```

All changes[1] in the pedalboard will be reproduced in mod-host. [1] Except in data attribute, changes in this does not interfere with anything.

Parameters **name** (*string*) – Pedalboard name

append(*effect*)

Add a *Effect* in this pedalboard

This works same as:

```
>>> pedalboard.effects.append(effect)
```

or:

```
>>> pedalboard.effects.insert(len(pedalboard.effects), effect)
```

Parameters **effect** (*Effect*) – Effect that will be added

connections
Return the pedalboard connections list

Note: Because the connections is an *ObservableList*, it isn't settable. For replace, del the connections unnecessary and add the necessary connections

effects
Return the effects presents in the pedalboard

Note: Because the effects is an *ObservableList*, it isn't settable. For replace, del the effects unnecessary and add the necessary effects

index
Returns the first occurrence of the pedalboard in your bank

json
Get a json decodable representation of this pedalboard

Return dict json representation

PedalPi - PluginsManager - Model - Lv2

Lv2EffectBuilder

class pluginsmanager.model.lv2.lv2_effect_builder.**Lv2EffectBuilder**(*plugins_json=None*)
Generates lv2 audio plugins instance (as *Lv2Effect* object).

Note: In the current implementation, the data plugins are persisted in *plugins.json*.

Parameters **plugins_json** (*Path*) – Plugins json path file

build(*lv2_uri*)
Returns a new *Lv2Effect* by the valid lv2_uri

Parameters **lv2_uri** (*string*) –

Return Lv2Effect Effect created

lv2_plugins_data()
Generates a file with all plugins data info. It uses the lilvlib library.

PluginsManager can manage lv2 audio plugins through previously obtained metadata from the lv2 audio plugins descriptor files.

To speed up usage, data has been pre-generated and loaded into this piped packet. This avoids a dependency installation in order to obtain the metadata.

However, this measure makes it not possible to manage audio plugins that were not included in the list.

To work around this problem, this method - using the lilvlib library - can get the information from the audio plugins. You can use this data to generate a file containing the settings:

```
>>> builder = Lv2EffectBuilder()
>>> plugins_data = builder.lv2_plugins_data()

>>> import json
>>> with open('plugins.json', 'w') as outfile:
>>>     json.dump(plugins_data, outfile)
```

The next time you instantiate this class, you can pass the configuration file:

```
>>> builder = Lv2EffectBuilder(os.path.abspath('plugins.json'))
```

Or, if you want to load the data without having to create a new instance of this class:

```
>>> builder.reload(builder.lv2_plugins_data())
```

Warning: To use this method, it is necessary that the system has the lilv in a version equal to or greater than 0.22.0. Many linux systems currently have previous versions on their package lists, so you need to compile them manually.

In order to ease the work, Pedal Pi has compiled lilv for some versions of linux. You can get the list of .deb packages in https://github.com/PedalPi/lilvlib/releases.

```
# Example
wget https://github.com/PedalPi/lilvlib/releases/download/v1.0.0/python3-
↪lilv_0.22.1.git20160613_amd64.deb
sudo dpkg -i python3-lilv_0.22.1+git20160613_amd64.deb
```

If the architecture of your computer is not contemplated, moddevices provided a script to generate the package. Go to https://github.com/moddevices/lilvlib to get the script in its most up-to-date version.

Return list lv2 audio plugins metadata

plugins_json_file = '/home/docs/checkouts/readthedocs.org/user_builds/pedalpi-pluginsmanager/checkouts/v0.5.1/p
Informs the path of the *plugins.json* file. This file contains the lv2 plugins metadata info

reload(*metadata*)
Loads the metadata. They will be used so that it is possible to generate lv2 audio plugins.

Parameters **metadata** (*list*) – lv2 audio plugins metadata

Lv2Effect

class pluginsmanager.model.lv2.lv2_effect.**Lv2Effect**(*plugin*)
Representation of a Lv2 audio plugin instance.

For general effect use, see *Effect* class documentation.

It's possible obtains the *Lv2Plugin* information:

```
>>> reverb
<Lv2Effect object as 'Calf Reverb'  active at 0x7f60effb09e8>
>>> reverb.plugin
<Lv2Plugin object as Calf Reverb at 0x7f60effb9940>
```

Parameters **plugin** (Lv2Plugin) –

Lv2Input

class pluginsmanager.model.lv2.lv2_input.**Lv2Input**(*effect*, *effect_input*)
Representation of a Lv2 input audio port instance.

For general input use, see *Input* class documentation.

Parameters

- **effect** (Lv2Effect) – Effect that contains the input
- **effect_input** (*dict*) – *input audio port* json representation

Lv2Output

class pluginsmanager.model.lv2.lv2_output.**Lv2Output**(*effect*, *effect_output*)
Representation of a Lv2 output audio port instance.

For general input use, see *Output* class documentation.

Parameters

- **effect** (Lv2Effect) – Effect that contains the output
- **effect_output** (*dict*) – *output audio port* json representation

Lv2Param

class pluginsmanager.model.lv2.lv2_param.**Lv2Param**(*effect*, *param*)
Representation of a Lv2 input control port instance.

For general input use, see *Param* class documentation.

Parameters

- **effect** (Lv2Effect) – Effect that contains the param
- **param** (*dict*) – *input control port* json representation

Lv2Plugin

class pluginsmanager.model.lv2.lv2_plugin.**Lv2Plugin**(*json*)

__getitem__(*key*)

Parameters **key** (*string*) – Property key

Returns Returns a Plugin property

json
Json decodable representation of this plugin based in moddevices lilvlib.

PedalPi - PluginsManager - Model - System

SystemEffectBuilder

class pluginsmanager.model.system.system_effect_builder.**SystemEffectBuilder**(*jack_client*)

Automatic system physical ports detection

Parameters **jack_client** (*JackClient*) – JackClient instance that will get the information to generate *SystemEffect*

SystemEffect

class pluginsmanager.model.system.system_effect.**SystemEffect**(*representation*, *outputs*, *inputs*)

Representation of the system instance (audio cards).

System output is equivalent with audio input: You connect the instrument in the audio card input and it captures and send the audio to *SystemOutput* for you connect in a input plugins.

System input is equivalent with audio output: The audio card receives the audio processed in your *SystemInput* and send it to audio card output for you connects in amplifier, headset.

Because no autodetection of existing ports in audio card has been implemented, you must explicitly inform in the creation of the SystemEffect object:

```
>>> sys_effect = SystemEffect('system', ('capture_1', 'capture_2'), ('playback_1',
↪ 'playback_2'))
```

Unlike effects that should be added in the pedalboard, SystemEffects MUST NOT:

```
>>> builder = Lv2EffectBuilder()
```

```
>>> pedalboard = Pedalboard('Rocksmith')
>>> reverb = builder.build('http://calf.sourceforge.net/plugins/Reverb')
>>> pedalboard.append(reverb)
```

However the pedalboard must have the connections:

```
>>> pedalboard.connections.append(Connection(sys_effect.outputs[0], reverb.
↪inputs[0]))
```

An bypass example:

```
>>> pedalboard = Pedalboard('Bypass example')
>>> sys_effect = SystemEffect('system', ('capture_1', 'capture_2'), ('playback_1',
↪ 'playback_2'))
>>> pedalboard.connections.append(Connection(sys_effect.outputs[0], sys_effect.
↪inputs[0]))
>>> pedalboard.connections.append(Connection(sys_effect.outputs[1], sys_effect.
↪inputs[1]))
```

Parameters

- **representation** (*string*) – Audio card representation. Usually 'system'
- **outputs** (*tuple(string)*) – Tuple of outputs representation. Usually a output representation starts with *capture_*

- **inputs** (*tuple(string)*) – Tuple of inputs representation. Usually a input representation starts with *playback_*

is_possible_connect_itself
return bool: Is possible connect the with it self?

SystemInput

class pluginsmanager.model.system.system_input.**SystemInput**(*effect*, *system_input*)

SystemOutput

class pluginsmanager.model.system.system_output.**SystemOutput**(*effect*, *output*)

PedalPi - PluginsManager - Observers

An observer is a class that receives notifications of changes in model classes (*Bank*, *Pedalboard*, *Effect*, *Param* ...).

Implementations

Some useful *UpdatesObserver* classes have been implemented. They are:

- *Autosaver*: Allows save the changes automatically in *json* data files.
- ModHost: Allows use mod-host, a LV2 host for Jack controllable via socket or command line

Using

For use a observer, it's necessary register it in *BanksManager*:

```
>>> saver = Autosaver()  # Autosaver is a UpdatesObserver
>>> banks_manager = BanksManager()
>>> banks_manager.register(saver)
```

For access all observers registered, use *BanksManager.observers*:

```
>>> saver in banks_manager.observers
True
```

For remove a observer:

```
>>> banks_manager.unregister(saver)
```

Creating a observer

It is possible to create observers! Some ideas are:

- Allow the use of other hosts (such as Carla);
- Automatically persist changes;

- Automatically update a human-machine interface (such as LEDs and displays that inform the state of the effects).

For create a observer, is necessary create a class that extends *UpdatesObserver*:

```
class AwesomeObserver(UpdatesObserver):
    ...
```

UpdatesObserver contains a number of methods that must be implemented in the created class. These methods will be called when changes occur:

```
class AwesomeObserver(UpdatesObserver):

    def on_bank_updated(self, bank, update_type, index, origin, **kwargs):
        pass

    def on_pedalboard_updated(self, pedalboard, update_type, index, origin, **kwargs):
        pass

    def on_effect_status_toggled(self, effect, **kwargs):
        pass

    def on_effect_updated(self, effect, update_type, index, origin, **kwargs):
        pass

    def on_param_value_changed(self, param, **kwargs):
        pass

    def on_connection_updated(self, connection, update_type, pedalboard, **kwargs):
        pass
```

Use the `update_type` attribute to check what type of change occurred:

```
class AwesomeObserver(UpdatesObserver):
    """Registers all pedalboards that have been deleted"""

    def __init__(self):
        super(AwesomeObserver, self).__init__()
        self.pedalboards_removed = []

    ...

    def on_pedalboard_updated(self, pedalboard, update_type, index, origin, **kwargs):
        if update_type == UpdateType.DELETED:
            self.pedalboards_removed.append(update_type)

    ...
```

Scope

Notification problem

There are cases where it makes no sense for an observer to be notified of a change. Usually this occurs in interfaces for control, where through them actions can be performed (activate an effect when pressing on a footswitch). Control interfaces need to know of changes that occur so that their display mechanisms are updated when some change occurs through another control interface.

Note that it does not make sense for an interface to be notified of the occurrence of any change if it was the one that performed the action.

A classic example would be an interface for control containing footswitch and a led. The footswitch changes the state of an effect and the led indicates whether it is active or not. If another interface to control (a mobile application, for example) changes the state of the effect to off, the led should reverse its state:

```
class MyControllerObserver(UpdatesObserver):

    ...

    def on_effect_status_toggled(self, effect, **kwargs):
        # Using gpiozero
        # https://gpiozero.readthedocs.io/en/stable/recipes.html#led
        self.led.toggle()
```

However, in this situation, when the footswitch changes the effect state, it is notified of the change itself. What can lead to inconsistency in the led:

```
def pressed():
    effect.toggle()
    led.toggle()

# footswitch is a button
# https://gpiozero.readthedocs.io/en/stable/recipes.html#button
footswitch.when_pressed = pressed
```

In this example, pressing the button:

1. `pressed()` is called;
2. The effect has its changed state (`effect.toggle()`);
3. `on_effect_status_toggled(self, effect, ** kwargs)` is called and the led is changed state (`self.led.toggle()`);
4. Finally, in `pressed()` is called `led.toggle()`.

That is, `led.toggle()` will be **called twice instead of one**.

Scope solution

Using `with` keyword, you can indicate which observer is performing the action, allowing the observer not to be notified of the updates that occur in the `with` scope:

```
>>> with observer1:
>>>     del manager.banks[0]
```

Example

Note: The complete example can be obtained from the examples folder of the repository. observer_scope.py

Consider an Observer who only prints actions taken on a bank:

```
class MyAwesomeObserver(UpdatesObserver):

    def __init__(self, message):
        super(MyAwesomeObserver, self).__init__()
        self.message = message

    def on_bank_updated(self, bank, update_type, **kwargs):
        print(self.message)

    ...
```

We will create two instances of this observer and perform some actions to see how the notification will occur:

```
>>> observer1 = MyAwesomeObserver("Hi! I am observer1")
>>> observer2 = MyAwesomeObserver("Hi! I am observer2")

>>> manager = BanksManager()
>>> manager.register(observer1)
>>> manager.register(observer1)
```

When notification occurs outside a `with` scope, all observers are informed of the change:

```
>>> bank = Bank('Bank 1')
>>> manager.banks.append(bank)
"Hi! I am observer1"
"Hi! I am observer2"
```

We'll now limit the notification by telling you who performed the actions:

```
>>> with observer1:
>>>     del manager.banks[0]
"Hi! I am observer2"
>>> with observer2:
>>>     manager.banks.append(bank)
"Hi! I am observer1"
```

If there is `with` inside a `with` block, the behavior will not change, ie it will not be cumulative

```
1 with observer1:
2     manager.banks.remove(bank)
3     with observer2:
4         manager.banks.append(bank)
```

Line 2 will result in `Hi! I am observer2` and line 4 in `Hi! I am observer1`

Base API

UpdateType

class `pluginsmanager.observer.update_type.`**`UpdateType`**

Enumeration for informs the change type.

See *`UpdatesObserver`* for more details

`CREATED` = **0**

Informs that the change is caused by the creation of an object

DELETED = 2
Informs that the change is caused by the removal of an object

UPDATED = 1
Informs that the change is caused by the update of an object

UpdatesObserver

class pluginsmanager.observer.updates_observer.**UpdatesObserver**
The *UpdatesObserver* is an abstract class definition for treatment of changes in some class model. Your methods are called when occurs any change in *Bank*, *Pedalboard*, *Effect*, etc.

To do this, it is necessary that the *UpdatesObserver* objects be registered in some manager, so that it reports the changes. An example of a manager is *BanksManager*.

on_bank_updated(*bank*, *update_type*, *index*, *origin*, ***kwargs*)
Called when changes occurs in any *Bank*

Parameters

- **bank** (Bank) – Bank changed.
- **update_type** (UpdateType) – Change type
- **index** (*int*) – Bank index (or old index if update_type == UpdateType.DELETED)
- **origin** (BanksManager) – BanksManager that the bank is (or has) contained
- **Bank** – Contains the old bank occurs a *UpdateType.UPDATED*

on_connection_updated(*connection*, *update_type*, *pedalboard*, ***kwargs*)
Called when changes occurs in any *pluginsmanager.model.connection.Connection* of Pedalboard (adding, updating or removing connections)

Parameters

- **connection** (pluginsmanager.model.connection.Connection) – Connection changed
- **update_type** (UpdateType) – Change type
- **pedalboard** (Pedalboard) – Pedalboard that the connection is (or has) contained

on_effect_status_toggled(*effect*, ***kwargs*)
Called when any *Effect* status is toggled

Parameters **effect** (Effect) – Effect when status has been toggled

on_effect_updated(*effect*, *update_type*, *index*, *origin*, ***kwargs*)
Called when changes occurs in any *Effect*

Parameters

- **effect** (Effect) – Effect changed
- **update_type** (UpdateType) – Change type
- **index** (*int*) – Effect index (or old index if update_type == UpdateType.DELETED)
- **origin** (Pedalboard) – Pedalboard that the effect is (or has) contained

on_param_value_changed(*param*, ***kwargs*)
Called when a param value change

Parameters **param** (Param) – Param with value changed

on_pedalboard_updated(*pedalboard*, *update_type*, *index*, *origin*, ***kwargs*)
Called when changes occurs in any *Pedalboard*

Parameters

- **pedalboard** (Pedalboard) – Pedalboard changed
- **update_type** (UpdateType) – Change type
- **index** (*int*) – Pedalboard index (or old index if update_type == UpdateType.DELETED)
- **origin** (Bank) – Bank that the pedalboard is (or has) contained
- **old** (Pedalboard) – Contains the old pedalboard when occurs a *UpdateType.UPDATED*

pluginsmanager.observer.observable_list.ObservableList

class pluginsmanager.observer.observable_list.**ObservableList**(*lista=None*)
Detects changes in list.

In append, in remove and in setter, the *observer* is callable with changes details

Based in https://www.pythonsheets.com/notes/python-basic.html#emulating-a-list

__contains__(*item*)
See `list.__contains__()` method

__delitem__(*sliced*)
See `list.__delitem__()` method

Calls observer `self.observer(UpdateType.DELETED, item, index)` where **item** is *self[index]*

__getitem__(*index*)
See `list.__getitem__()` method

__iter__()
See `list.__iter__()` method

__len__()
See `list.__len__()` method

__repr__()
See `list.__repr__()` method

__setitem__(*index*, *val*)
See `list.__setitem__()` method

Calls observer `self.observer(UpdateType.UPDATED, item, index)` if `val != self[index]`

__str__()
See `list.__repr__()` method

append(*item*)
See `list.append()` method

Calls observer `self.observer(UpdateType.CREATED, item, index)` where **index** is *item position*

index(*x*)
See `list.index()` method

insert (*index*, *x*)
See `list.insert()` method

Calls observer `self.observer(UpdateType.CREATED, item, index)`

move (*item*, *new_position*)
Moves a item list to new position

Calls observer `self.observer(UpdateType.DELETED, item, index)` and observer `self.observer(UpdateType.CREATED, item, index)` if `val != self[index]`

Parameters

- **item** – Item that will be moved to new_position
- **new_position** – Item's new position

pop (*index=None*)
See `list.pop()` method

Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

Parameters **index** (*int*) – element index that will be removed

Returns item removed

remove (*item*)
See `list.remove()` method

Calls observer `self.observer(UpdateType.DELETED, item, index)` where **index** is *item position*

Implementations API

pluginsmanager.observer.autosaver.autosaver.Autosaver

class `pluginsmanager.observer.autosaver.autosaver.`**Autosaver** (*data_path*, *auto_save=True*)
The UpdatesObserver *Autosaver* allows save any changes automatically in json data files. Save all plugins changes in json files in a specified path.

It also allows loading of saved files:

```
>>> system_effect = SystemEffect('system', ('capture_1', 'capture_2'), ('playback_
↪1', 'playback_2'))
>>>
>>> autosaver = Autosaver('my/path/data/')
>>> banks_manager = autosaver.load(system_effect)
```

When loads data with *Autosaver*, the autosaver has registered in observers of the banks_manager generated:

```
>>> autosaver in banks_manager.observers
True
```

For manual registering in *BanksManager* uses *register()*:

```
>>> banks_manager = BanksManager()
>>> autosaver = Autosaver('my/path/data/')
>>> autosaver in banks_manager.observers
False
```

```
>>> banks_manager.register(autosaver)
>>> autosaver in banks_manager.observers
True
```

After registered, any changes in *Bank*, *Pedalboard*, *Effect*, *Connection* or *Param* which belong to the structure of *BanksManager* instance are persisted automatically by *Autosaver*:

```
>>> banks_manager = BanksManager()
>>> banks_manager.register(autosaver)
>>> my_bank = Bank('My bank')
>>> banks_manager.append(my_bank)
>>> # The bank will be added in banksmanger
>>> # and now is observable (and persisted) by autosaver
```

It's possible disables autosaver for saves manually:

```
>>> autosaver.auto_save = False
>>> autosaver.save(banks_manager)   # save() method saves all banks data
```

Parameters

- **data_path** (*string*) – Path that banks will be saved (each bank in one file)
- **auto_save** (*bool*) – Auto save any change?

load(*system_effect*)

Return a *BanksManager* instance contains the banks present in `data_path`

Parameters **system_effect** (`SystemEffect`) – SystemEffect used in pedalboards

Return BanksManager *BanksManager* with banks persisted in `data_path`

save(*banks_manager*)

Save all data from a banks_manager

Parameters **banks_manager** (`BanksManager`) – BanksManager that your banks data will be persisted

PedalPi - PluginsManager - Util

pluginsmanager.util.dict_tuple.DictTuple

class `pluginsmanager.util.dict_tuple.`**DictTuple**(*elements*, *key_function*)

Dict tuple is a union with dicts and tuples. It's possible obtains an element by index or by a key.

The key is not been a int or long instance

Based in http://jfine-python-classes.readthedocs.io/en/latest/subclass-tuple.html

Parameters

- **elements** (*iterable*) – Elements for the tuple
- **key_function** (*lambda*) – Function mapper: it obtains an element and returns your key.

pluginsmanager.util.pairs_list.PairsList

class pluginsmanager.util.pairs_list.**PairsList**(*similarity_key_function*)
Receives two lists and generates a result list of pairs of equal elements

Uses *calculate* method for generate list

Parameters **similarity_key_function** – Function that receives a element and returns your identifier to do a mapping with elements from another list

pluginsmanager.util.pairs_list.PairsListResult

class pluginsmanager.util.pairs_list.**PairsListResult**

pluginsmanager.util.persistence_decoder

class pluginsmanager.util.persistence_decoder.**PersistenceDecoderError**

class pluginsmanager.util.persistence_decoder.**PersistenceDecoder**(*system_effect*)

class pluginsmanager.util.persistence_decoder.**Reader**(*system_effect*)

class pluginsmanager.util.persistence_decoder.**BankReader**(*system_effect*)

class pluginsmanager.util.persistence_decoder.**PedalboardReader**(*system_effect*)

class pluginsmanager.util.persistence_decoder.**EffectReader**(*system_effect*)

class pluginsmanager.util.persistence_decoder.**ConnectionReader**(*pedalboard*, *system_effect*)

generate_builder(*json*, *audio_port*)
:return AudioPortBuilder

Index

Symbols

__contains__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__del__() (pluginsmanager.observer.mod_host.mod_host.ModHost method), 17
__delitem__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__getitem__() (pluginsmanager.model.lv2.lv2_plugin.Lv2Plugin method), 35
__getitem__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__iter__() (pluginsmanager.banks_manager.BanksManager method), 24
__iter__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__len__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__repr__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__setitem__() (pluginsmanager.observer.observable_list.ObservableList method), 42
__str__() (pluginsmanager.observer.observable_list.ObservableList method), 42

A

active (pluginsmanager.model.effect.Effect attribute), 27
add() (pluginsmanager.observer.mod_host.host.Host method), 18
add() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 19
append() (pluginsmanager.banks_manager.BanksManager method), 24
append() (pluginsmanager.model.bank.Bank method), 26
append() (pluginsmanager.model.pedalboard.Pedalboard method), 32
append() (pluginsmanager.observer.observable_list.ObservableList method), 42
Autosaver (class in pluginsmanager.observer.autosaver.autosaver), 43

B

Bank (class in pluginsmanager.model.bank), 25
BankReader (class in pluginsmanager.util.persistence_decoder), 45
BanksManager (class in pluginsmanager.banks_manager), 24
build() (pluginsmanager.model.lv2.lv2_effect_builder.Lv2EffectBuilder method), 33
bypass() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 19

C

close() (pluginsmanager.observer.mod_host.connection.Connection method), 18
close() (pluginsmanager.observer.mod_host.host.Host method), 18
close() (pluginsmanager.observer.mod_host.mod_host.ModHost method), 17
connect() (pluginsmanager.model.output.Output method), 29
connect() (pluginsmanager.observer.mod_host.host.Host method), 18
connect() (pluginsmanager.observer.mod_host.mod_host.ModHost method), 17
connect() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 19

Connection (class in pluginsmanager.model.connection), 26
Connection (class in pluginsmanager.observer.mod_host.connection), 18
ConnectionReader (class in pluginsmanager.util.persistence_decoder), 45
connections (pluginsmanager.model.effect.Effect attribute), 28
connections (pluginsmanager.model.pedalboard.Pedalboard attribute), 33
CREATED (pluginsmanager.observer.update_type.UpdateType attribute), 40

D

default (pluginsmanager.model.param.Param attribute), 31
DELETED (pluginsmanager.observer.update_type.UpdateType attribute), 40
DictTuple (class in pluginsmanager.util.dict_tuple), 44
disconnect() (pluginsmanager.model.output.Output method), 30
disconnect() (pluginsmanager.observer.mod_host.host.Host method), 18
disconnect() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 19

E

Effect (class in pluginsmanager.model.effect), 27
effect (pluginsmanager.model.input.Input attribute), 29
effect (pluginsmanager.model.output.Output attribute), 30
effect (pluginsmanager.model.param.Param attribute), 31
EffectReader (class in pluginsmanager.util.persistence_decoder), 45
effects (pluginsmanager.model.pedalboard.Pedalboard attribute), 33
enter_scope() (pluginsmanager.banks_manager.BanksManager method), 24
exit_scope() (pluginsmanager.banks_manager.BanksManager method), 24

G

generate_builder() (pluginsmanager.util.persistence_decoder.ConnectionReader method), 45

H

help() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 19
Host (class in pluginsmanager.observer.mod_host.host), 18

I

index (pluginsmanager.model.bank.Bank attribute), 26
index (pluginsmanager.model.effect.Effect attribute), 28
index (pluginsmanager.model.input.Input attribute), 29
index (pluginsmanager.model.output.Output attribute), 30
index (pluginsmanager.model.pedalboard.Pedalboard attribute), 33
index() (pluginsmanager.observer.observable_list.ObservableList method), 42
Input (class in pluginsmanager.model.input), 28
input (pluginsmanager.model.connection.Connection attribute), 27
inputs (pluginsmanager.model.effect.Effect attribute), 28
insert() (pluginsmanager.observer.observable_list.ObservableList method), 42
is_possible_connect_itself (pluginsmanager.model.effect.Effect attribute), 28
is_possible_connect_itself (pluginsmanager.model.system.system_effect.SystemEffect attribute), 37

J

json (pluginsmanager.model.bank.Bank attribute), 26
json (pluginsmanager.model.connection.Connection attribute), 27
json (pluginsmanager.model.effect.Effect attribute), 28
json (pluginsmanager.model.input.Input attribute), 29
json (pluginsmanager.model.lv2.lv2_plugin.Lv2Plugin attribute), 35
json (pluginsmanager.model.output.Output attribute), 30
json (pluginsmanager.model.param.Param attribute), 31
json (pluginsmanager.model.pedalboard.Pedalboard attribute), 33

L

load() (pluginsmanager.observer.autosaver.autosaver.Autosaver method), 44
load() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 20
lv2_plugins_data() (pluginsmanager.model.lv2.lv2_effect_builder.Lv2EffectBuilder method), 33
Lv2Effect (class in pluginsmanager.model.lv2.lv2_effect), 34
Lv2EffectBuilder (class in pluginsmanager.model.lv2.lv2_effect_builder), 33
Lv2Input (class in pluginsmanager.model.lv2.lv2_input), 35
Lv2Output (class in pluginsmanager.model.lv2.lv2_output), 35

Lv2Param (class in pluginsmanager.model.lv2.lv2_param), 35
Lv2Plugin (class in pluginsmanager.model.lv2.lv2_plugin), 35

M

maximum (pluginsmanager.model.param.Param attribute), 31
midi_learn() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 20
midi_map() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 20
midi_unmap() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 20
minimum (pluginsmanager.model.param.Param attribute), 31
ModHost (class in pluginsmanager.observer.mod_host.mod_host), 16
monitor() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 20
move() (pluginsmanager.observer.observable_list.ObservableList method), 43

O

ObservableList (class in pluginsmanager.observer.observable_list), 42
observers (pluginsmanager.banks_manager.BanksManager attribute), 24
on_bank_updated() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
on_connection_updated() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
on_effect_status_toggled() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
on_effect_updated() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
on_param_value_changed() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
on_pedalboard_updated() (pluginsmanager.observer.updates_observer.UpdatesObserver method), 41
Output (class in pluginsmanager.model.output), 29
output (pluginsmanager.model.connection.Connection attribute), 27
outputs (pluginsmanager.model.effect.Effect attribute), 28

P

PairsList (class in pluginsmanager.util.pairs_list), 45
PairsListResult (class in pluginsmanager.util.pairs_list), 45
Param (class in pluginsmanager.model.param), 30
param_get() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 21
param_monitor() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 21
param_set() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 21
params (pluginsmanager.model.effect.Effect attribute), 28
Pedalboard (class in pluginsmanager.model.pedalboard), 32
pedalboard (pluginsmanager.observer.mod_host.mod_host.ModHost attribute), 17
PedalboardReader (class in pluginsmanager.util.persistence_decoder), 45
PersistenceDecoder (class in pluginsmanager.util.persistence_decoder), 45
PersistenceDecoderError (class in pluginsmanager.util.persistence_decoder), 45
plugins_json_file (pluginsmanager.model.lv2.lv2_effect_builder.Lv2EffectBuilder attribute), 34
pop() (pluginsmanager.observer.observable_list.ObservableList method), 43
preset_load() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 21
preset_save() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 21
preset_show() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 22
ProtocolParser (class in pluginsmanager.observer.mod_host.protocol_parser), 19

Q

quit() (pluginsmanager.observer.mod_host.host.Host method), 18
quit() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 22

R

Reader (class in pluginsmanager.util.persistence_decoder), 45
register() (pluginsmanager.banks_manager.BanksManager method), 25
reload() (pluginsmanager.model.lv2.lv2_effect_builder.Lv2EffectBuilder method), 34
remove() (pluginsmanager.observer.mod_host.host.Host method), 18
remove() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 22
remove() (pluginsmanager.observer.observable_list.ObservableList method), 43

S

save() (pluginsmanager.observer.autosaver.autosaver.Autosaver method), 44
save() (pluginsmanager.observer.mod_host.protocol_parser.ProtocolParser static method), 22
send() (pluginsmanager.observer.mod_host.connection.Connection method), 18
set_param_value() (pluginsmanager.observer.mod_host.host.Host method), 18
set_status() (pluginsmanager.observer.mod_host.host.Host method), 18
start() (pluginsmanager.observer.mod_host.mod_host.ModHost method), 17
symbol (pluginsmanager.model.input.Input attribute), 29
symbol (pluginsmanager.model.output.Output attribute), 30
symbol (pluginsmanager.model.param.Param attribute), 31
SystemEffect (class in pluginsmanager.model.system.system_effect), 36
SystemEffectBuilder (class in pluginsmanager.model.system.system_effect_builder), 36
SystemInput (class in pluginsmanager.model.system.system_input), 37
SystemOutput (class in pluginsmanager.model.system.system_output), 37

T

toggle() (pluginsmanager.model.effect.Effect method), 28

U

unregister() (pluginsmanager.banks_manager.BanksManager method), 25
UPDATED (pluginsmanager.observer.update_type.UpdateType attribute), 41
UpdatesObserver (class in pluginsmanager.observer.updates_observer), 41
UpdateType (class in pluginsmanager.observer.update_type), 40

V

value (pluginsmanager.model.param.Param attribute), 31