Sound Card Module Documentation

Release 0.1.5

Luís Teixeira

Nov 07, 2019

First steps

1 Table of contents **3**

1.1 Getting started 3

1.2 User interface 3

1.3 `SoundCard USB connection` 6

1.4 `pybpodapi` 12

Python Module Index **15**

Index **17**

The Harp Sound Card is a board developed by the Scientific Hardware Platform at the Champalimaud Foundation.

This documentation is for the Python 3 library to control the **pybpod_soundcard_module** module.

CHAPTER 1

Table of contents

1.1 Getting started

1.1.1 Requirements

This library requires the following Python 3 packages to be installed.

- libusb
- pyusb
- aenum

1.2 User interface

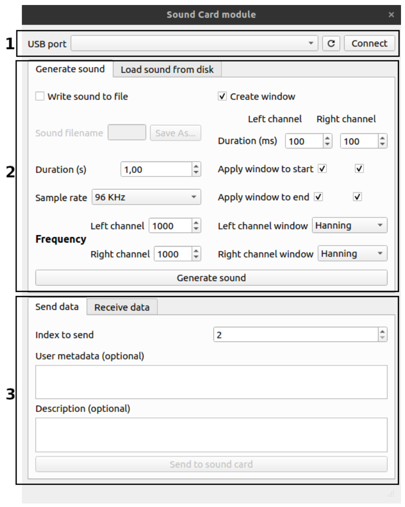
The main window that comprises the user interface of the SoundCard Module is divided in three main sections as it can be seen in the following Figure.

1. Connection to the Sound Card
2. Sound generation or loading sounds from the disk
3. Sending or receiving data to the Sound Card

1.2.1 1. Connection to the Sound Card

When connecting the Harp Sound Card to the computer, the device will appear in the dropdown control for the USB port. If you connect it later, or reset the Sound Card, you can press the refresh button to force a search for connected devices.

After a device is connected and selected in the dropdown, you can connect to it by pressing the Connect button.



Write sound to file ☐
Create window ✓

	Left channel	Right channel
Duration (ms)	100	100
Apply window to start	✓	✓
Apply window to end	✓	✓

Duration (s) 1,00
Sample rate 96 KHz
Frequency: Left channel 1000; Right channel 1000
Left channel window Hanning
Right channel window Hanning
Index to send 2
User metadata (optional)
Description (optional)

Fig. 1: Main window for the SoundCard Module

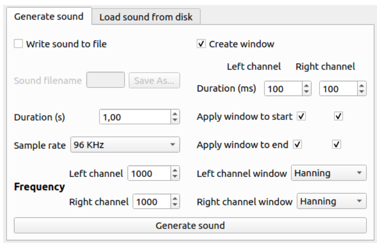
Note: The operations in the bottom section of the window require that a connect is established. When you press Connect and a successful connection is made, the operations buttons will enable.

1.2.2 2. Sound generation or loading sounds from the disk

If you want to send sounds to the Sound Card , it is necessary to either generate a sound using the provided UI, or you can load it from the disk.

Generating sounds

The UI allows to create sounds with a sinewave, with distinct frequencies for the left and right channel. The duration, in seconds, of the sound can also be configured. The two sample rates supported by the Harp Sound Card can be selected in the sound generation portion of the UI.



Write sound to file ☐
Create window ✓

	Left channel	Right channel
Duration (ms)	100	100
Apply window to start	✓	✓
Apply window to end	✓	✓

Duration (s) 1,00
Sample rate 96 KHz
Frequency: Left channel 1000; Right channel 1000
Left channel window Hanning
Right channel window Hanning

Fig. 2: Generate sounds portion of the window

When pressing the **Generate sound** button, the sound will be generated and it will be possible to send it to the Sound Card, at the bottom portion of the main UI.

Note: It is possible to generate a sound in memory and send it directly to the Sound Card. However, if you wish to save the generated sound to the disk, you can activate the checkbox **Write sound to file** and create a new file.

At the same time, it is possible to generate the sound with a window function at the start and/or end of the sound. The duration of this window can be different for the left and right channel, and its duration is defined in milliseconds. It is also possible to define if the window function should be applied to the start, end or both sides of the sound. The window function used can also be selected. The available options are: Hanning, Hamming, Blackman and Bartlett.

Note: The window functions implementation are provided by the NumPy library. For further details, please visit the (opens in new tab).

Loading sounds from disk

To load a sound from the disk, two options are available. You can either load a generated sound that was previously saved or you can load a WAV file (except 24bits WAV files).

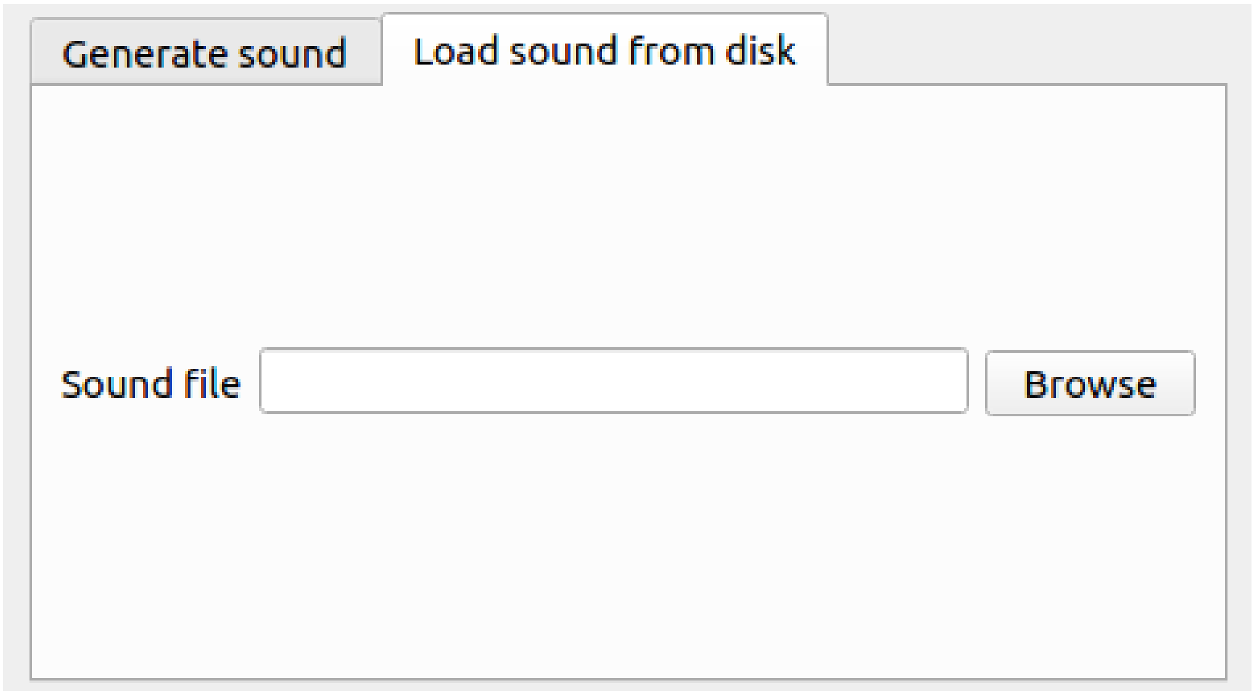


Fig. 3: Load sound from disk portion of the window

The procedure to load a sound is pressing Browse and select the file. The file will be loaded automatically. If there are loading errors, you will be notified.

1.2.3 3. Sending or receiving data to the Sound Card

Sending data to the Sound Card

Sending a sound to the Sound Card can be accompanied by some user metadata and some description information. Those fields are optional and can be defined in the UI.

Warning: The **user metadata** field has a limit of 1024 bytes and the **description** field has a limit of 512 bytes. Any data that passes this limit will be **truncated**.

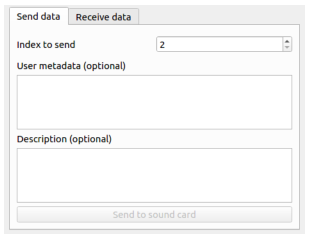


Fig. 4: Send data portion of the window

After selecting the index to where to write the sound and data, by pressing the **Send to sound card** button will send the sound and data to the Harp Sound Card.

Note: The index to where to save the sound must be **greater or equal** than 2 and **lower** than 32. The interface will limit the input to those values.

Receiving data from the Sound Card

In the same manner that it is possible to send additional data with the sound (user metadata and a description), when receiving data from the Sound Card, that data will also be written in the destination folder, if it exists.

The UI allows to receive the data from a single index or from all the indexes.

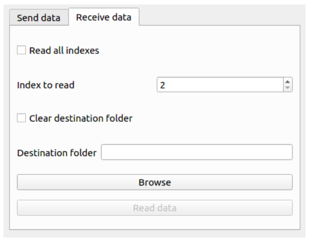


Fig. 5: Receive data portion of the window

The procedure to receive data from the Sound Card is by selecting a destination folder, by pressing the **Browse** button, and selecting if the application should clear the destination folder or not.

Warning: Even if the destination folder is not cleared, the files will be overwritten if they already exist.

1.3 `SoundCard USB connection`

class `pybpod_soundcard_module.module_api.`**`SampleRate`**

Enumeration for the Sample rate of the sounds in the Sound Card

`_96000HZ = 96000`

96KHz sample rate

`_192000HZ = 192000`

192KHz sample rate

class `pybpod_soundcard_module.module_api.`**`DataType`**

Type of the data to be send to the Sound Card

`INT32 = 0`

Integer 32 bits

`FLOAT32 = 1`

Single precision float

class `pybpod_soundcard_module.module_api.`**`SoundCardModule`**`(`*device=None*`)`

Provides access to the Harp Sound Card. It allows to send and read the sounds in the Sound Card, through a normal USB connection.

If a libUSB's device is given, it will try to open it. If none is given it will try to connect to the first Sound Card that is connected to the computer.

Parameters **`device`** – (Optional) libUSB device to use. If nothing is passed, it will try to connect automatically.

`open``(`*device=None*`)`

Opens the connection to the Sound Card. If no device is given, it will try to connect to the first Sound Card that is connected to the computer.

Parameters **`device`** – (Optional) Already initialized libUSB's device to use.

`close``()`

Closes the connection with the Sound Card. It will close USB connection (to read and save sounds)

`reset``()`

Resets the device, waits 700ms and tries to connect again so that the current instance of the SoundCard object can still be used.

Note: Necessary at the moment after sending a sound.

`read_sounds``(`*output_folder=None*, *sound_index=None*, *clean_dst_folder=True*`)`

Reads sounds from the sound card.

Note: by default, it will clear the destination folder of all data. It will also write by default to a "from_soundcard" folder in the working directory if none is given.

Parameters

- **`output_folder`** – Destination folder's path.
- **`sound_index`** – If a sound_index is given, it will get only that sound, if nothing is passed it will gather all sounds from all indexes.
- **`clean_dst_folder`** – Flag that defines if the method should clean the destination folder or not

`send_sound``(`*wave_int*, *sound_index*, *sample_rate*, *data_type*, *sound_filename=None*, *metadata_filename=None*, *description_filename=None*`)`

This method will send the sound to the Harp Sound Card as a byte array (int8)

Parameters

- **`wave_int`** – NumPy array as int32 that represents the sound data
- **`sound_index`** – The destination index in the Sound Card (>=2 and <= 32)

- **sample_rate** – The SampleRate enum value for either 96KHz or 192KHz
- **data_type** – The DataType enum value for either Int32 or Float32 (not implemented yet in the hardware)
- **sound_filename** – The name of the sound filename to be saved with the sound in the board (str)
- **metadata_filename** – The name of the metadata filename to be saved with the sound in the board (str)
- **description_filename** – The name of the description filename to be saved with the sound in the board (str)

class pybpod_soundcard_module.utils.generate_sound.**WindowConfiguration**(*left_duration=0.1*, *left_apply_window_start=Tr* *left_apply_window_end=Tru* *left_window_function='Hann* *right_duration=0.1*, *right_apply_window_start=T* *right_apply_window_end=Tr* *right_window_function='Ha*

Parameters

- **left_duration** – (Optional) Duration of the window in seconds, for the left channel. If zero, no window will be created for this channel. Default: 0.1s
- **left_apply_window_start** – (Optional) True if the window should be applied to the **start** of the sound for the left channel, False otherwise. Default: True
- **left_apply_window_end** – (Optional) True if the window should be applied to the **end** of the sound for the left channel, False otherwise. Default: True
- **left_window_function** – (Optional) Window function that should be used for the left channel. Possible values accepted: 'Hanning', 'Hamming', 'Blackman', 'Bartlett'. Default: 'Hanning"
- **right_duration** – (Optional) Duration of the window in seconds, for the right channel. If zero, no window will be created for this channel. Default: 0.1s
- **right_apply_window_start** – (Optional) True if the window should be applied to the **start** of the sound, for the right channel, False otherwise. Default: True
- **right_apply_window_end** – (Optional) True if the window should be applied to the **end** of the sound for the right channel, False otherwise. Default: True
- **right_window_function** – (Optional) Window function that should be used for the left channel. Possible values accepted: 'Hanning', 'Hamming', 'Blackman', 'Bartlett'. Default: 'Hanning"

pybpod_soundcard_module.utils.generate_sound.**generate_sound**(*filename=None*, *fs=96000*, *duration=1*, *frequency_left=1000*, *frequency_right=1000*, *window_configuration: pybpod_soundcard_module.utils.generate_sound* *= None*)

Helper method to dynamically generated a sound that can be used in with the Sound Card module.

Parameters

- **filename** – (Optional)
- **fs** – (Optional) number of samples per second (standard)
- **duration** – (Optional) sound duration in seconds
- **frequency_left** – (Optional) number of cycles per second (Hz) (frequency of the sine wave for the left channel)
- **frequency_right** – (Optional) number of cycles per second (Hz) (frequency of the sine wave for the right channel)
- **window_configuration** – (Optional) WindowConfiguration object to apply to the generated sound.

Returns Returns the **flatten** generated sound as a numpy array (as np.int8)

pybpod_soundcard_module.utils.generate_sound.**generate_window**(*fs*, *wave_int*, *duration*, *apply_start*, *apply_end*, *window_function*)

Parameters

- **fs** – number of samples per second (standard)
- **wave_int** – base sound where the window will be applied
- **duration** – duration of the window (it will be the same on the start and end)
- **apply_start** – True if the window should be created at the start, False otherwise.
- **apply_end** – True if the window should be created at the end, False otherwise.
- **window_function** – window function to be generated. Possible values accepted: 'Hanning', 'Hamming', 'Blackman', 'Bartlett'. It will revert to 'Hanning' if an unknown option is given.

Returns Returns the modified sound with the window applied to it.

1.3.1 Usage Example

```
import numpy as np
from pybpod_soundcard_module.module import SoundCard, SoundCommandType
from pybpod_soundcard_module.module_api import SoundCardModule, DataType, SampleRate
from pybpod_soundcard_module.utils.generate_sound import generate_sound

card = SoundCardModule()
card.open()

sound_filename = 'sound.bin'
sound_index = 4

# load file and read data (we are using the numpy's fromfile method)
wave_int = np.fromfile(sound_filename, dtype=np.int32)

# NOTE: As an alternative, we can generate a sound dynamically with the helper method␣
↪generate_sound
```

(continues on next page)

(continued from previous page)

```
wave_int = generate_sound(sound_filename,          # optional, if given, it will␣
↪save the generated sound to the hard drive
                          fs=96000,                # sample rate in Hz
                          duration=4,              # duration of the sound in seconds
                          frequency_left=1500,     # frequency of the sinusoidal␣
↪signal generated in Hz for the left channel
                          frequency_right=1200)    # frequency of the sinusoidal␣
↪signal generated in Hz for the right channel

# send sound
card.send_sound(wave_int,
                sound_index,
                SampleRate._96000HZ,
                DataType.INT32,
                sound_filename,
                'sound_metadata.bin',     # optional
                'sound_description.txt') # optional

# reads the files related with the sound in index 4, without cleaning the destination␣
↪folder
card.read_sounds(output_folder='folder', sound_index=sound_index, clean_dst_
↪folder=False)

card.close()
```

1.3.2 Usage Example (using 'with' statement)

```
import numpy as np
from pybpod_soundcard_module.module import SoundCard, SoundCommandType
from pybpod_soundcard_module.module_api import SoundCardModule, DataType, SampleRate
from pybpod_soundcard_module.utils.generate_sound import generate_sound

sound_filename = 'sound.bin'
sound_index = 4

# load file and read data (we are using the numpy's fromfile method)
wave_int = np.fromfile(sound_filename, dtype=np.int32)

# NOTE: As an alternative, we can generate a sound dynamically with the helper method␣
↪generate_sound
wave_int = generate_sound(sound_filename,          # optional, if given, it will␣
↪save the generated sound to the hard drive
                          fs=96000,                # sample rate in Hz
                          duration=4,              # duration of the sound in seconds
                          frequency_left=1500,     # frequency of the sinusoidal␣
↪signal generated in Hz for the left channel
                          frequency_right=1200)    # frequency of the sinusoidal␣
↪signal generated in Hz for the right channel


# the with statement will call 'close' automatically at the end of the block
with SoundCardModule() as card:
    # send sound
    card.send_sound(wave_int,
                    sound_index,
```

(continues on next page)

(continued from previous page)

```
                        SampleRate._96000HZ,
                        DataType.INT32,
                        sound_filename,
                        'sound_metadata.bin',     # optional
                        'sound_description.txt') # optional

    # reads the files related with the sound in index 4 without cleaning the␣
↪destination folder
    card.read_sounds(output_folder='folder', sound_index=4, clean_dst_folder=False)
```

1.3.3 Usage Example (sound generation with WindowConfiguration)

```
import numpy as np
from pybpod_soundcard_module.module import SoundCard, SoundCommandType
from pybpod_soundcard_module.module_api import SoundCardModule, DataType, SampleRate
from pybpod_soundcard_module.utils.generate_sound import generate_sound,␣
↪WindowConfiguration

sound_filename = 'sound.bin'
sound_index = 4

# create WindowConfiguration to later pass it to generate_sound
# NOTE: (exemplification of options available, change options according to your needs)
window_config = WindowConfiguration( left_duration = 0.2,                   # It is␣
↪possible to define different durations for the left and right channel.
                                     left_apply_window_start = True,        # For␣
↪this example, we want a start window
                                     left_apply_window_end = False,         # ... and␣
↪no 'end' window for the left channel
                                     left_window_function = 'Blackman',     # ... and␣
↪with 'Blackman' window function
                                     right_duration = 0.1,                  # It is␣
↪possible to define different durations for the left and right channel.
                                     right_apply_window_start = False,      # For␣
↪this example, we don't want a start window
                                     right_apply_window_end = True,         # ... and␣
↪a 'end' window for the right channel
                                     right_window_function = 'Bartlett')    # ... and␣
↪with 'Bartlett' window function

wave_int = generate_sound(sound_filename,                       # optional, if given,␣
↪it will save the generated sound to the hard drive
                          fs=96000,                             # sample rate in Hz
                          duration=4,                           # duration of the␣
↪sound in seconds
                          frequency_left=1500,                  # frequency of the␣
↪sinusoidal signal generated in Hz for the left channel
                          frequency_right=1200,                 # frequency of the␣
↪sinusoidal signal generated in Hz for the right channel
                          window_configuration=window_config)   # window configuration


# the with statement will call 'close' automatically at the end of the block
with SoundCardModule() as card:
    # send sound
```

(continues on next page)

(continued from previous page)

```
    card.send_sound(wave_int,
                    sound_index,
                    SampleRate._96000HZ,
                    DataType.INT32,
                    sound_filename,
                    'sound_metadata.bin',     # optional
                    'sound_description.txt') # optional

    # reads the files related with the sound in index 4 without cleaning the␣
↪destination folder
    card.read_sounds(output_folder='folder', sound_index=4, clean_dst_folder=False)
```

1.4 pybpodapi

class pybpod_soundcard_module.module.**SoundCommandType**

Enumeration for the commands that can be sent through the Sound Card module, when connected through the BPod's State Machine.

PLAY = 1

Plays a specific sound index

STOP_SPECIFIC = 2

Stops playing a specific sound index

STOP_ALL = 3

Stops all sounds

class pybpod_soundcard_module.module.**SoundCard**(*connected=False*, *module_name=''*, *firmware_version=0*, *events_names=[]*, *n_serial_events=0*, *serial_port=None*)

Bases: pybpodapi.bpod_modules.bpod_module.BpodModule

static get_command(*command_type*, *sound_index=None*)

Returns the proper bytes to send as output_actions in the BPod StateMachine's states. In the case of the Play and StopSpecific command_types, it is required to use the BPod's load_serial_message method to be able to send to the module more than 1 byte properly.

Note: This might not be required in a future version of BPod's firmware

Parameters

- **command_type** – Instruction of type *SoundCommandType* to generate the command
- **sound_index** – The sound index to play or stop

1.4.1 Usage Example

```
from pybpodapi.protocol import Bpod, StateMachine
from pybpod_soundcard_module.module import SoundCard, SoundCommandType

# sound index to play
sound_index = 2
```

(continues on next page)

(continued from previous page)

```
my_bpod = Bpod(serial_port='/dev/ttyACM0')

# get first SoundBoard module connected to a Bpod serial port
sound_module = [x for x in my_bpod.modules if x.name == 'SoundCard1'][0]
sound_module_play = 1

card = (SoundCard)(sound_module)

# define serial message to be used in the states with the sound_module_play id
my_bpod.load_serial_message(sound_module, sound_module_play,
                      card.get_command(SoundCommandType.PLAY, sound_index))

sma = StateMachine(my_bpod)

sma.add_state(
    state_name='myState',
    state_timer=2,
    state_change_conditions={Bpod.Events.Tup: 'myState2'},
    output_actions=[(Bpod.OutputChannels.Serial1, sound_module_play)])

sma.add_state(
    state_name='myState2',
    state_timer=3,
    state_change_conditions={Bpod.Events.Tup: 'exit'},
    output_actions=[(Bpod.OutputChannels.Serial1, card.get_command(SoundCommandType.
↪STOP_ALL))])

my_bpod.send_state_machine(sma)

my_bpod.run_state_machine(sma)

print("Current trial info: {0}".format(my_bpod.session.current_trial))

my_bpod.close()
```

Python Module Index

p

`pybpod_soundcard_module.utils.generate_sound`, 8

`pybpodapi`, 3

Index

Symbols

_192000HZ (*pybpod_soundcard_module.module_api.SampleRate attribute*), 6

_96000HZ (*pybpod_soundcard_module.module_api.SampleRate attribute*), 6

C

close() (*pybpod_soundcard_module.module_api.SoundCardModule method*), 7

D

DataType (*class in pybpod_soundcard_module.module_api*), 6

F

FLOAT32 (*pybpod_soundcard_module.module_api.DataType attribute*), 7

G

generate_sound() (*in module pybpod_soundcard_module.utils.generate_sound*), 8

generate_window() (*in module pybpod_soundcard_module.utils.generate_sound*), 9

get_command() (*pybpod_soundcard_module.module.SoundCard static method*), 12

I

INT32 (*pybpod_soundcard_module.module_api.DataType attribute*), 7

O

open() (*pybpod_soundcard_module.module_api.SoundCardModule method*), 7

P

PLAY (*pybpod_soundcard_module.module.SoundCommandType attribute*), 12

pybpod_soundcard_module.utils.generate_sound (*module*), 8

pybpodapi (*module*), 3, 6, 12

R

read_sounds() (*pybpod_soundcard_module.module_api.SoundCardModule method*), 7

reset() (*pybpod_soundcard_module.module_api.SoundCardModule method*), 7

S

SampleRate (*class in pybpod_soundcard_module.module_api*), 6

send_sound() (*pybpod_soundcard_module.module_api.SoundCardModule method*), 7

SoundCard (*class in pybpod_soundcard_module.module*), 12

SoundCardModule (*class in pybpod_soundcard_module.module_api*), 7

SoundCommandType (*class in pybpod_soundcard_module.module*), 12

STOP_ALL (*pybpod_soundcard_module.module.SoundCommandType attribute*), 12

STOP_SPECIFIC (*pybpod_soundcard_module.module.SoundCommandType attribute*), 12

W

WindowConfiguration (*class in pybpod_soundcard_module.utils.generate_sound*), 8