Adafruit Soundboard Library Documentation

Release 0.1

Mike Mabey

Jul 31, 2017

Contents

The Adafruit Soundboards are an easy way to add sound to your maker project, but the library provided by Adafruit only supports Arduino.

If you've wanted to use one of these boards with a MicroPython microcontroller (MCU), this is the library you've been looking for.

CHAPTER 1

Installation

At this time, you have to install the library by copying the `soundboard.py` script to your MicroPython board along with your `main.py` file. At some point in the future it may be possible to `pip install` it.

Make sure to get the latest version of the code from GitHub.

CHAPTER 2

Quick Start

First, you'll need to decide which UART bus you want to use. To do this, you'll need to consult the documentation for your particular MCU. In these examples, I'm using the original `pyboard` (see documentation here) and I'm using UART bus 1 or `XB`, which uses pin `X9` for transmitting and ping `X10` for receiving.

Then, create an instance of the *`Soundboard`* class, like this:

```
sound = Soundboard('XB')
```

I *highly* recommend you also attach the `RST` pin on the soundboard to one of the other GPIO pins on the MCU (pin *X11* in the example). Also, my alternative method of getting the list of files from the board is more stable (in my own testing) than the method built-in to the soundboard. Also, I like getting the debug output and I turn the volume down to 50% while I'm coding. Doing all this looks like the following:

```
SB_RST = 'X11'
sound = Soundboard('XB', rst_pin=SB_RST, vol=0.5, debug=True, alt_get_files=True)
```

Once you've set up all of this, you're ready to play some tracks:

```
# Play track 0
sound.play(0)

# Stop playback
sound.stop()

# Play the test file that comes with the soundboard
sound.play('T00     OGG')

# Play track 1 immediately, stopping any currently playing tracks
sound.play_now(1)

# Pause and resume
sound.pause()
sound.unpause()
```

You can also control the volume in several different ways:

```
# Raise volume by 2 points (0 min volume, 204 max volume)
sound.vol_up()

# Turn down volume until lower than 125
sound.vol_down(125)

# Get the current volume
sound.vol

# Set volume to 56 (out of 204 maximum)
sound.vol = 56

# Set volume to 75% of maximum volume
sound.vol = 0.75
```

CHAPTER 3

Documentation

This is a MicroPython library for the Adafruit Sound Boards in UART mode!

This library has been adapted from the library written by Adafruit for Arduino, available at https://github.com/adafruit/Adafruit_Soundboard_library. I have no affiliation with Adafruit, and they have not sponsored or approved this library in any way. As such, please do not contact them for support regarding this library.

Commands the sound board understands (at least the ones I could discern from the Arduino library) are as follows:

- `L`: List files on the board
- `#`: Play a file by number
- `P`: Play a file by name
- `+`: Volume up. Range is 0-204, increments of 2.
- `-`: Volume down
- `=`: Pause playback
- `>`: Un-pause playback
- `q`: Stop playback
- `t`: Give current position of playback and total time of track
- `s`: Current track size and total size

`soundboard.`**`SB_BAUD`**
: The baud rate for the soundboards. This shouldn't ever change, since all of the soundboard models use the same value.

  See also:

  Adafruit Audio FX Sound Board Tutorial Adafruit's tutorial on the soundboards.

`soundboard.`**`MIN_VOL`**

`soundboard.`**`MAX_VOL`**
: Minimum volume is 0, maximum is 204.

soundboard.**MAX_FILES**
In the Arduino version of this library, it defines the max number of files to be 25.

soundboard.**DEBUG**
A flag for turning on/off debug messages.

See also:

Soundboard.toggle_debug(), *printif()*

class soundboard.**Soundboard**(*uart_id*, *rst_pin=None*, *vol=None*, *alt_get_files=False*, *debug=None*, ***uart_kwargs*)
Control an Adafruit Sound Board via UART.

The *Soundboard* class handles all communication with the sound board via UART, making it easy to get information about the sound files on the sound board and control playback.

If you need to reset the sound board from your MicroPython code, be sure to provide the `rst_pin` parameter. The soundboard sometimes gets out of UART mode and reverts to the factory default of GPIO trigger mode. When this happens, it will appear as if the soundboard has stoped working for no apparent reason. This library is designed to automatically attempt resetting the board if a command fails, since that is a common cause. So, it is a good idea to provide this parameter.

Parameters

- **uart_id** – ID for the UART bus to use. Acceptable values vary by board. Check the documentation for your board for more info.
- **rst_pin** – Identifier for the pin (on the MicroPython board) connected to the `RST` pin of the sound board. Valid identifiers vary by board.
- **vol** (*int or float*) – Initial volume level to set. See *vol* for more info.
- **alt_get_files** (*bool*) – Uses an alternate method to get the list of track file names. See *use_alt_get_files()* method for more info.
- **debug** (*bool*) – When not None, will set the debug output flag to the boolean value of this argument using the *toggle_debug()* method.
- **uart_kwargs** (*dict*) – Additional values passed to the UART.init() method of the UART bus object. Acceptable values here also vary by board. It is not necessary to include the baud rate among these keyword values, because it will be set to `SB_BAUD` before the `UART.init` function is called.

files
Return a `list` of the files on the sound board.

Return type list

sizes
Return a `list` of the files' sizes on the sound board.

See also:

use_alt_get_files()

Return type list

lengths
Return a `list` of the track lengths in seconds.

Note: In my own testing of this method, the board always returns a value of zero seconds for the length for every track, no matter if it's a WAV or OGG file, short or long track.

Return type list

file_name(*n*)
Return the name of track `n`.

Parameters **n** (*int*) – Index of a file on the sound board or `False` if the track number doesn't exist.

Returns Filename of track `n`.

Return type str or bool

track_num(*file_name*)
Return the track number of the given file name.

Parameters **file_name** (*str*) – File name of the track. Should be one of the values from the *files* property.

Returns The track number of the file name or `False` if not found.

Return type int or bool

play(*track=None*)
Play a track on the board.

Parameters **track** (*int or str*) – The index (`int`) or filename (`str`) of the track to play.

Returns If the command was successful.

Return type bool

play_now(*track*)
Play a track on the board now, stopping current track if necessary.

Parameters **track** (*int or str*) – The index (`int`) or filename (`str`) of the track to play.

Returns If the command was successful.

Return type bool

vol
Current volume.

This is implemented as a class property, so you can get and set its value directly. When setting a new volume, you can use an `int` or a `float` (assuming your board supports floats). When setting to an `int`, it should be in the range of 0-204. When set to a `float`, the value will be interpreted as a percentage of *MAX_VOL*.

Return type int

vol_up(*vol=None*)
Turn volume up by 2 points, return current volume level [0-204].

Parameters **vol** (*int*) – Target volume. When not `None`, volume will be turned up to be greater than or equal to this value.

Return type int

vol_down(*vol=None*)
Turn volume down by 2 points, return current volume level [0-204].

Parameters **vol** (*int*) – Target volume. When not `None`, volume will be turned down to be less than or equal to this value.

Return type int

pause()
Pause playback, return if the command was successful.

Return type bool

unpause()
Continue playback, return if the command was successful.

Return type bool

stop()
Stop playback, return if the command was successful.

Return type bool

track_time()
Return the current position of playback and total time of track.

Return type tuple

track_size()
Return the remaining size and total size.

It seems the remaining track size refers to the number of bytes left for the soundboard to process before the playing of the track will be over.

Returns Remaining track size and total size

Return type tuple

reset()
Reset the sound board.

Soft reset the board by bringing the `RST` pin low momentarily (10 ms). This only has effect if the reset pin has been initialized in the constructor.

Doing a soft reset on the board before doing any other actions can help ensure that it has been started in UART control mode, rather than GPIO trigger mode.

See also:

Soundboard Pinout Documentation on the soundboards' pinouts.

Returns Whether the reset was successful. If the reset pin was not initialized in the constructor, this will always return `False`.

Return type bool

use_alt_get_files(*now=False*)
Get list of track files using an alternate method.

If the list of files is missing tracks you know are on the soundboard, try calling this method. It doesn't depend on the soundboard's internal command for returning a list of files. Instead, it plays each of the tracks using their track numbers and gets the filename and size from the output of the play command.

Parameters **now** (*bool*) – When set to `True`, the alternate method of getting the files list will be called immediately. Otherwise, the list of files will be populated the next time the *files* property is accessed (lazy loading).

Return type None

static **toggle_debug**(*debug=None*)
Turn on/off *DEBUG* flag.

Parameters **debug** – If None, the *DEBUG* flag will be toggled to have the value opposite of its current value. Otherwise, *DEBUG* will be set to the boolean value of `debug`.

Return type None

`soundboard.`**printif**(**values*, ***kwargs*)
Print a message if *DEBUG* is set to `True`.

Python Module Index

S

Index