pybpod-api Documentation

Release 1.6.4

Carlos Mão de Ferro

Nov 07, 2019

Getting started

Note: All examples and Bpod's state machine and communication logic were based on the original version made available by Josh Sanders (Sanworks).

CHAPTER 1

What is pybpod-api?

pybpod-api is a Python library that enables communication with the latest Bpod device version. You can use it directly as a CLI (Command Line Interface) or use your favorite GUI to interact with it.

This library is maintained by a team of SW developers at the Champalimaud Foundation. Please find more information on section *Project Info*.

1.1 What is Bpod?

Bpod is a system from Sanworks for precise measurement of small animal behavior. It is a family of open source hardware devices which includes also software and firmware to control these devices. The software was originally developed in Matlab providing retro-compatibility with the BControl system.

See also:

Bpod device: https://sanworks.io/shop/viewproduct?productID=1011

Bpod on Github: https://github.com/sanworks/Bpod

Bpod Wiki: https://sites.google.com/site/bpoddocumentation/

BControl project: http://brodywiki.princeton.edu/bcontrol/index.php/Main_Page/

1.2 Why a Python port?

Python is one of the most popular programming languages today [1]. This is special true for the science research community because it is an open language, easy to learn, with a strong support community and with a lot of libraries available.

CHAPTER 2

Questions?

If you have any questions or want to report a problem with this library please fill in an issue here.

2.1 Installing

Note: To install the **full pybpod package**, please follow the instructions located @ Pybpod.

2.1.1 Installing for using the library

The library is available through PyPI so you just have to do

```
pip install pybpod-api
```

2.1.2 Installing for making changes to the library

1. Clone the repository

   ```
   git clone https://github.com/pybpod/pybpod-api
   ```

2. On the project root folder (where *'setup.py'* is located) run the following command

   ```
   pip install -e . # installs this API in development mode
   ```

3. Use your code editor of choice to make your changes.

2.1.3 Settings file

```
# list of python libraries to interface with bpod modules.
PYBPOD_API_MODULES = [
    'pybpod_rotaryencoder_module'
]

PYBPOD_SERIAL_PORT = '/dev/ttyACM0'  # serial port settings
PYBPOD_NET_PORT    = '' # network port to receive remote commands like softcodes.

# enable or disable bpod ports
BPOD_BNC_PORTS_ENABLED      = [True, True]
BPOD_WIRED_PORTS_ENABLED    = [True, True]
BPOD_BEHAVIOR_PORTS_ENABLED = [True, True, True, True, True, True, True, True]

PYBPOD_PROTOCOL     = '' # Executed protocol
PYBPOD_CREATOR      = '' # Name of the user
PYBPOD_PROJECT      = '' # Name of the project
PYBPOD_EXPERIMENT   = '' # Name of the experiment
PYBPOD_BOARD        = '' # Board name
PYBPOD_SETUP        = '' # Setup name
PYBPOD_SESSION      = '' # Name of the session file
PYBPOD_SESSION_PATH = '' # Folder where the bpod output files are be saved
PYBPOD_SUBJECTS     = [] # List of subjects to be saved in the session file.
```

2.2 Running examples

2.2.1 Configure settings

In order to run protocols you need to specify a bpod *'user_settings.py'* file that should be located at the execution folder.

Example of *'examples/user_settings.py'* file:

```
# -*- coding: utf-8-*-

PYBPOD_API_LOG_LEVEL = None

PYBPOD_SESSION_PATH  = 'SESSION-WORKSPACE'

# if you do not define the next variable, the PYBPOD_SESSION
# will assume the current datetime value.
PYBPOD_SESSION = 'SESSION-NAME'

SERIAL_PORT   = '/dev/ttyACM0'
```

2.2.2 Running protocol examples

Example for running the *'add_trial_events.py'*:

```
cd PROJECT_FOLDER/examples
python -m function_examples.add_trial_events
```

2.2.3 Available examples

Obtain Bpod Info

Basic example demonstrating how to initialize Bpod and read version, firmware version and machine type version.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Get hardware info from Bpod

"""

from pybpodapi.protocol import Bpod
from confapp import conf


my_bpod = Bpod()

my_bpod.close()

print("Target Bpod firmware version: ", conf.TARGET_BPOD_FIRMWARE_VERSION)
print("Firmware version (read from device): ", my_bpod.hardware.firmware_version)
print("Machine type version (read from device): ", my_bpod.hardware.machine_type)
```

Run the example with:

```
python -m function_examples.bpod_info
```

One state example

Simple example of adding a state to the state machine and run it. A timer is used to change state.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine


"""
Run this protocol now
"""

my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.add_state(
        state_name='myState',
        state_timer=1,
        state_change_conditions={Bpod.Events.Tup: 'exit'},
        output_actions=[])
```

(continues on next page)

(continued from previous page)

```
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()
```

Run the example with:

```
python -m state_machine_examples.one_state
```

Light chasing example (3 pokes)

Simulation of a light chasing scenario. Follow the light on 3 pokes.

Connect noseports to ports 1-3.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Light Chasing example

Follow light on 3 pokes

Connect noseports to ports 1-3.

"""
from pybpodapi.protocol import Bpod, StateMachine


my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.add_state(
        state_name='Port1Active1',  # Add a state
        state_timer=0,
        state_change_conditions={Bpod.Events.Port1In: 'Port2Active1'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Active1',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port2In: 'Port3Active1'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

sma.add_state(
        state_name='Port3Active1',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port3In: 'Port1Active2'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])
```

(continues on next page)

(continued from previous page)

```
sma.add_state(
        state_name='Port1Active2',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port1In: 'Port2Active2'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Active2',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port2In: 'Port3Active2'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

sma.add_state(
        state_name='Port3Active2',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port3In: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.light_chasing
```

Light chasing example (2 pokes)

Simulation of a light chasing scenario. Follow the light on 2 pokes.

Connect noseports to ports 1-2.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Light Chasing example

Follow light on 2 pokes

Connect noseports to ports 1-2.

"""
from pybpodapi.protocol import Bpod, StateMachine


my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.add_state(
```

(continues on next page)

(continued from previous page)

```
        state_name='Port1Active1',  # Add a state
        state_timer=0,
        state_change_conditions={Bpod.Events.Port1In: 'Port2Active1'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Active1',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port2In: 'Port1Active2'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

sma.add_state(
        state_name='Port1Active2',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port1In: 'Port2Active2'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Active2',
        state_timer=0,
        state_change_conditions={Bpod.Events.Port2In: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.light_chasing_2_pokes
```

Add trial events

Demonstration of AddTrialEvents used in a simple visual 2AFC session.

AddTrialEvents formats each trial's data in a human-readable struct, and adds to myBpod.data (to save to disk later)

Connect noseports to ports 1-3.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Demonstration of AddTrialEvents used in a simple visual 2AFC session.
AddTrialEvents formats each trial's data in a human-readable struct, and adds to␣
↪myBpod.data (to save to disk later)
Connect noseports to ports 1-3.

Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""

import random
```

(continues on next page)

(continued from previous page)

```
from pybpodapi.protocol import Bpod, StateMachine

my_bpod = Bpod()

nTrials = 5
trialTypes = [1, 2]  # 1 (rewarded left) or 2 (rewarded right)

for i in range(nTrials):  # Main loop
        print('Trial: ', i + 1)

        thisTrialType = random.choice(trialTypes)  # Randomly choose trial type
        if thisTrialType == 1:
                stimulus = Bpod.OutputChannels.PWM1  # set stimulus channel for trial␣
↪type 1
                leftAction = 'Reward'
                rightAction = 'Punish'
                rewardValve = 1
        elif thisTrialType == 2:
                stimulus = Bpod.OutputChannels.PWM3  # set stimulus channel for trial␣
↪type 1
                leftAction = 'Punish'
                rightAction = 'Reward'
                rewardValve = 3

        sma = StateMachine(my_bpod)

        sma.add_state(
                state_name='WaitForPort2Poke',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port2In: 'FlashStimulus'},
                output_actions=[(Bpod.OutputChannels.PWM2, 255)])
        sma.add_state(
                state_name='FlashStimulus',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'WaitForResponse'},
                output_actions=[(stimulus, 255)])
        sma.add_state(
                state_name='WaitForResponse',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port1In: leftAction, Bpod.Events.
↪Port3In: rightAction},
                output_actions=[])
        sma.add_state(
                state_name='Reward',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.Valve, rewardValve)])  # Reward␣
↪correct choice
        sma.add_state(
                state_name='Punish',
                state_timer=3,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.LED, 1), (Bpod.OutputChannels.
↪LED, 2), (Bpod.OutputChannels.LED, 3)])  # Signal incorrect choice

        my_bpod.send_state_machine(sma)  # Send state machine description to Bpod␣
↪device
```

(continues on next page)

(continued from previous page)

```
        print("Waiting for poke. Reward: ", 'left' if thisTrialType == 1 else 'right')

        my_bpod.run_state_machine(sma)  # Run state machine

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

my_bpod.close()  # Disconnect Bpod
```

Run the example with:

```
python -m function_examples.add_trial_events
```

Add trial events 2

Similiar to previous example but using a global timer and adding more states.

Connect noseports to ports 1-3.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Demonstration of AddTrialEvents used in a simple visual 2AFC session.
AddTrialEvents formats each trial's data in a human-readable struct, and adds to␣
↪myBpod.data (to save to disk later)
Connect noseports to ports 1-3.

Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""

import random
from pybpodapi.protocol import Bpod, StateMachine


my_bpod = Bpod()

nTrials = 5
graceTime = 5
trialTypes = [1, 2]  # 1 (rewarded left) or 2 (rewarded right)

for i in range(nTrials):  # Main loop
        print('Trial: ', i + 1)

        thisTrialType = random.choice(trialTypes)  # Randomly choose trial type
        if thisTrialType == 1:
                stimulus = Bpod.OutputChannels.PWM1  # set stimulus channel for trial␣
↪type 1
                leftAction = 'Reward'
                rightAction = 'Punish'
                rewardValve = 1
        elif thisTrialType == 2:
                stimulus = Bpod.OutputChannels.PWM3  # set stimulus channel for trial␣
↪type 1
```

(continues on next page)

(continued from previous page)

```
                leftAction = 'Punish'
                rightAction = 'Reward'
                rewardValve = 3

        sma = StateMachine(my_bpod)

        sma.set_global_timer_legacy(timer_id=1, timer_duration=graceTime)  # Set␣
↪timeout

        sma.add_state(
                state_name='WaitForPort2Poke',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port2In: 'FlashStimulus'},
                output_actions=[('PWM2', 255)])

        sma.add_state(
                state_name='FlashStimulus',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'WaitForResponse'},
                output_actions=[(stimulus, 255, Bpod.OutputChannels.GlobalTimerTrig,␣
↪1)])

        sma.add_state(
                state_name='WaitForResponse',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port1In: leftAction,
                                         Bpod.Events.Port3In: rightAction,
                                         Bpod.Events.Port2In: 'Warning',
                                         Bpod.Events.GlobalTimer1_End: 'MiniPunish'},
                output_actions=[])

        sma.add_state(
                state_name='Warning',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'WaitForResponse',
                                         Bpod.Events.GlobalTimer1_End: 'MiniPunish'},
                output_actions=[(Bpod.OutputChannels.LED, 1),
                                (Bpod.OutputChannels.LED, 2),
                                (Bpod.OutputChannels.LED, 3)])  # Reward correct␣
↪choice

        sma.add_state(
                state_name='Reward',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.Valve, rewardValve)])  # Reward␣
↪correct choice

        sma.add_state(
                state_name='Punish',
                state_timer=3,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.LED, 1),
                                (Bpod.OutputChannels.LED, 2),
                                (Bpod.OutputChannels.LED, 3)])  # Signal incorrect␣
↪choice
```

(continues on next page)

(continued from previous page)

```
        sma.add_state(
                state_name='MiniPunish',
                state_timer=1,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.LED, 1),
                                (Bpod.OutputChannels.LED, 2),
                                (Bpod.OutputChannels.LED, 3)])  # Signal incorrect␣
↪choice

        my_bpod.send_state_machine(sma)  # Send state machine description to Bpod␣
↪device

        print("Waiting for poke. Reward: ", 'left' if thisTrialType == 1 else 'right')

        my_bpod.run_state_machine(sma)  # Run state machine

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

my_bpod.close()  # Disconnect Bpod
```

Run the example with:

```
python -m function_examples.add_trial_events2
```

Manual override

Manually interact with Bpod hardware. For a detailed explanation, please refer to *Manual control of Bpod*.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Manually set values on Bpod channels via serial instructions.

Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""

import time

from pybpodapi.protocol import Bpod

import examples.settings as settings



my_bpod = Bpod()

wait_active_time_ms = 2

### INPUTS - BNC (1, 2) ###
print("Set BNC1 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, Bpod.ChannelNames.BNC, channel_
↪number=1, value=12)
time.sleep(wait_active_time_ms)
```

(continues on next page)

(continued from previous page)

```
print("Set BNC2 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, Bpod.ChannelNames.BNC, channel_
↪number=2, value=15)
time.sleep(wait_active_time_ms)

### INPUTS - PWM (1..4) ###
print("Set PWM1 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, 'Port', channel_number=1, value=12)
time.sleep(wait_active_time_ms)

print("Set PWM2 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, 'Port', channel_number=2, value=13)
time.sleep(wait_active_time_ms)

print("Set PWM3 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, 'Port', channel_number=3, value=14)
time.sleep(wait_active_time_ms)

print("Set PWM3 (Input) to a value")
my_bpod.manual_override(Bpod.ChannelTypes.INPUT, 'Port', channel_number=4, value=15)
time.sleep(wait_active_time_ms)


### PORT 1 LED ###

print("Set LED of port 1 to max intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=255)
time.sleep(wait_active_time_ms)

print("Set LED of port 1 to lower intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=8)
time.sleep(wait_active_time_ms)

print("Set LED of port 1 to zero intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=0)
time.sleep(1)

### PORT 2 LED ###

print("Set LED of port 2 to max intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=2, value=255)
time.sleep(wait_active_time_ms)

print("Set LED of port 2 to lower intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=2, value=8)
time.sleep(wait_active_time_ms)

print("Set LED of port 2 to zero intensity")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=2, value=0)
time.sleep(1)  # Wait 1s
```

(continues on next page)

(continued from previous page)

```
### PORT 1 VALVE ###

print("Set valve of port 1 to open")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.VALVE, 1, value=1)
time.sleep(wait_active_time_ms)

print("Set valve of port 1 to close")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.VALVE, 1, value=0)
time.sleep(1)  # Wait 1s

### PORT 3 VALVE ###

print("Set valve of port 3 to open")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.VALVE, channel_
↪number=3, value=1)
time.sleep(wait_active_time_ms)  # Wait 250ms

print("Set valve of port 3 to close")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.VALVE, channel_
↪number=3, value=0)
time.sleep(1)  # Wait 1s

### PORT 2 BNC ###

print("Set BNC output ch2 to high")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.BNC, channel_
↪number=2, value=1)
time.sleep(0.01)  # Wait 10ms

print("Set BNC output ch2 to low")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.BNC, channel_
↪number=2, value=0)
time.sleep(1)  # Wait 1s

### PORT 3 Wire ###

print("Set Wire output ch3 to high")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.WIRE, channel_
↪number=3, value=1)
time.sleep(0.01)  # Wait 10ms

print("Set Wire output ch3 to low")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.WIRE, channel_
↪number=3, value=0)
time.sleep(1)  # Wait 1s

### PORT 2 Serial ###

print("Send byte 65 on UART port 2")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.SERIAL, channel_
↪number=2, value=65)
time.sleep(0.01)  # Wait 10ms

print("Send byte 66 on UART port 1")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.SERIAL, channel_
↪number=1, value=66)
```

(continues on next page)

(continued from previous page)

```
# Stop Bpod
my_bpod.close()  # Sends a termination byte and closes the serial port. PulsePal␣
↪stores current params to its EEPROM.
```

Run the example with:

```
python -m function_examples.manual_override
```

Serial messages

Example on how to use serial capabilities of Bpod.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""

import time

from pybpodapi.protocol import Bpod


my_bpod = Bpod()

print("Send byte 65 on UART port 1 - by default, this is ASCII 'A'")
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.SERIAL, 1, 65)
time.sleep(1)  # Wait 1s

print("Set byte 65 ('A') on UART port 1 to trigger a 3-byte message: 'BCD'")
my_bpod.load_serial_message(1, 65, [66, 67, 68])
# Now, the same command has a different result
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.SERIAL, channel_
↪number=1, value=65)
time.sleep(1)  # Wait 1s

print("Reset the serial message library. Bytes will now pass through again.")
my_bpod.reset_serial_messages()
# Back to 'A'
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.SERIAL, channel_
↪number=1, value=65)

# Stop Bpod
my_bpod.close()  # Sends a termination byte and closes the serial port. PulsePal␣
↪stores current params to its EEPROM.
```

Run the example with:

```
python -m function_examples.serial_messages
```

Global timers examples

Several examples demonstrating how to interact with Bpod timers.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine



my_bpod = Bpod()

sma = StateMachine(my_bpod)

# Set global timer 1 for 3 seconds
sma.set_global_timer_legacy(timer_id=1, timer_duration=3)

sma.add_state(
        state_name='TimerTrig',  # Trigger global timer
        state_timer=0,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit'},
        output_actions=[(Bpod.OutputChannels.GlobalTimerTrig, 1)])

sma.add_state(
        state_name='Port1Lit',  # Infinite loop (with next state). Only a global␣
↪timer can save us.
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port3Lit', Bpod.Events.
↪GlobalTimer1_End: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port3Lit',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit', Bpod.Events.
↪GlobalTimer1_End: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.global_timer_example
```

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine
```

(continues on next page)

(continued from previous page)

```
"""
Run this protocol now
"""

my_bpod = Bpod()

sma = StateMachine(my_bpod)

# Set global timer 1 for 3 seconds, following a 1.5 second onset delay after trigger.␣
↪Link to channel BNC2.
sma.set_global_timer(timer_id=1, timer_duration=3, on_set_delay=1.5, channel='BNC2')

sma.add_state(
        state_name='TimerTrig',  # Trigger global timer
        state_timer=0,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit'},
        output_actions=[(Bpod.OutputChannels.GlobalTimerTrig, 1)])

sma.add_state(
        state_name='Port1Lit',  # Infinite loop (with next state). Only a global␣
↪timer can save us.
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port3Lit', 'GlobalTimer1_End':
↪'exit'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port3Lit',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit', 'GlobalTimer1_End':
↪'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.global_timer_example_digital
```

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example state machine: A global timer triggers passage through two infinite loops. It␣
↪is
triggered in the first state, but begins measuring its 3-second Duration
after a 1.5s onset delay. During the onset delay, an infinite loop
toggles two port LEDs (Port1, Port3) at low intensity. When the timer begins␣
↪measuring,
```

(continues on next page)

(continued from previous page)

```
it sets port 2 LED to maximum brightness, and triggers transition to a second␣
↪infinite loop with brighter port 1+3 LEDs.
When the timer's 3 second duration elapses, Port2LED is returned low,
and a GlobalTimer1_End event occurs (handled by exiting the state machine).


Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine

"""
Run this protocol now
"""

my_bpod = Bpod()

sma = StateMachine(my_bpod)

# Set global timer 1 for 3 seconds, following a 1.5 second onset delay after trigger.␣
↪Link to LED of port 2.
sma.set_global_timer(timer_id=1, timer_duration=3, on_set_delay=1.5, channel=Bpod.
↪OutputChannels.PWM2, on_message=255)

sma.add_state(
        state_name='TimerTrig',  # Trigger global timer
        state_timer=0,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit_Pre'},
        output_actions=[('GlobalTimerTrig', 1)])

sma.add_state(
        state_name='Port1Lit_Pre',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port3Lit_Pre', Bpod.Events.
↪GlobalTimer1_Start: 'Port1Lit_Post'},
        output_actions=[(Bpod.OutputChannels.PWM1, 16)])

sma.add_state(
        state_name='Port3Lit_Pre',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit_Pre', Bpod.Events.
↪GlobalTimer1_Start: 'Port3Lit_Post'},
        output_actions=[(Bpod.OutputChannels.PWM3, 16)])

sma.add_state(
        state_name='Port1Lit_Post',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port3Lit_Post', Bpod.Events.
↪GlobalTimer1_End: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port3Lit_Post',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit_Post', Bpod.Events.
↪GlobalTimer1_End: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])
```

(continues on next page)

(continued from previous page)

```
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()
```

Run the example with:

```
python -m state_machine_examples.global_timer_start_and_end_events
```

Global counter example

After poke2 (PWM2) LED turns off, one will have an infinite loop between LED of poke1 (PWM1) and LED of poke3 (PWM1).

To interrupt the infinite loop one have to interrupt poke1 or poke3 a number of times equal to threshold (in this case is 5 times).

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository

After poke2 (PWM2) LED turns off, one will have an infinite loop between LED of poke1␣
↪(PWM1) and LED of poke3 (PWM1).

To interrupt the infinite loop one have to interrupt poke1 or poke3 a number of times␣
↪equal to threshold (in this case is 5 times).

"""
from pybpodapi.protocol import Bpod, StateMachine

my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.set_global_counter(counter_number=1, target_event='Port1In', threshold=5)

sma.add_state(
        state_name='InitialDelay',
        state_timer=2,
        state_change_conditions={Bpod.Events.Tup: 'ResetGlobalCounter1'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

sma.add_state(
        state_name='ResetGlobalCounter1',
        state_timer=0,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit'},
        output_actions=[(Bpod.OutputChannels.GlobalCounterReset, 1)])

sma.add_state(
        state_name='Port1Lit',  # Infinite loop (with next state). Only a global␣
↪counter can save us.
```

(continues on next page)

(continued from previous page)

```
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port3Lit', 'GlobalCounter1_End':
↪'exit'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port3Lit',
        state_timer=.25,
        state_change_conditions={Bpod.Events.Tup: 'Port1Lit', 'GlobalCounter1_End':
↪'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.global_counter_example
```

Setting a condition example

Example on how to set a condition.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine


"""
Run this protocol now
"""

my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.set_condition(condition_number=1, condition_channel='Port2', channel_value=1)

sma.add_state(
        state_name='Port1Light',
        state_timer=1,
        state_change_conditions={Bpod.Events.Tup: 'Port2Light'},
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Light',
```

(continues on next page)

(continued from previous page)

```
        state_timer=1,
        state_change_conditions={Bpod.Events.Tup: 'Port3Light', Bpod.Events.
↪Condition1: 'Port3Light'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])

sma.add_state(
        state_name='Port3Light',
        state_timer=1,
        state_change_conditions={Bpod.Events.Tup: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM3, 255)])

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()
```

Run the example with:

```
python -m state_machine_examples.condition_example
```

UART triggered state example

Example on how a UART event can trigger a state change.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""
from pybpodapi.protocol import Bpod, StateMachine
"""
Run this protocol now
"""

my_bpod = Bpod()

sma = StateMachine(my_bpod)

sma.add_state(
        state_name='Port1Light',
        state_timer=0,
        state_change_conditions={Bpod.Events.Serial2_3: 'Port2Light'},  # Go to␣
↪Port2Light when byte 0x3 arrives on UART port 2
        output_actions=[(Bpod.OutputChannels.PWM1, 255)])

sma.add_state(
        state_name='Port2Light',
        state_timer=0,
        state_change_conditions={Bpod.Events.Tup: 'exit'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])
```

(continues on next page)

(continued from previous page)

```
my_bpod.send_state_machine(sma)

my_bpod.run_state_machine(sma)

print("Current trial info: ", my_bpod.session.current_trial)

my_bpod.close()
```

Run the example with:

```
python -m state_machine_examples.uart_triggered_state_change
```

2.3 Writing a protocol for Bpod

2.3.1 What is a Bpod protocol?

To use Bpod, you must first program a behavioral protocol. The following guide is based on the original version for Bpod Matlab.

2.3.2 Protocol example explained

1. Import the modules

First, you will need to import Bpod modules.

```
from pybpodapi.protocol import Bpod, StateMachine
```

2. Initialize Bpod

Initialize Bpod and provide serial connection.

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

Instead of hard coding the serial port in your scripts you can configure it using the **user_settings.py** file.

Create the files __init__.py and user_settings.py in the running directory (check the examples folder on pybpod source code). Now you can instantiate Bpod() without having to pass the serial port as parameter.

```
my_bpod = Bpod()
```

3. Run several trials

Run several trials in each Bpod execution. In this example, we will use 5 trials where each trial can be of type1 (rewarded left) or type2 (rewarded right).

```
nTrials = 5
trialTypes = [1, 2]  # 1 (rewarded left) or 2 (rewarded right)

for i in range(nTrials):  # Main loop
```

(continues on next page)

(continued from previous page)

```
    print('Trial: ', i+1)
    thisTrialType = random.choice(trialTypes)  # Randomly choose trial type =
    if thisTrialType == 1:
        stimulus = Bpod.OutputChannels.PWM1  # set stimulus channel for trial type 1
        leftAction = 'Reward'
        rightAction = 'Punish'
        rewardValve = 1
    elif thisTrialType == 2:
        stimulus = Bpod.OutputChannels.PWM3  # set stimulus channel for trial type 1
        leftAction = 'Punish'
        rightAction = 'Reward'
        rewardValve = 3
```

Now, inside the loop, we will create and configure a state machine for each trial. A state machine has *state name*, *state timer*, *names of states to enter if certain events occur* and *output actions*. Please see *State Machine API* for detailed information about state machine design.

Warning: We strongly advise to use the API available labels as described on the examples *output actions* and *input events*.

```
    sma = StateMachine(my_bpod)

    sma.add_state(
        state_name='WaitForPort2Poke',
        state_timer=1,
        state_change_conditions={Bpod.Events.Port2In: 'FlashStimulus'},
        output_actions=[(Bpod.OutputChannels.PWM2, 255)])
    sma.add_state(
        state_name='FlashStimulus',
        state_timer=0.1,
        state_change_conditions={Bpod.Events.Tup: 'WaitForResponse'},
        output_actions=[(stimulus, 255)])
    sma.add_state(
        state_name='WaitForResponse',
        state_timer=1,
        state_change_conditions={Bpod.Events.Port1In: leftAction, Bpod.Events.
↪Port3In: rightAction},
        output_actions=[])
    sma.add_state(
        state_name='Reward',
        state_timer=0.1,
        state_change_conditions={Bpod.Events.Tup: 'exit'},
        output_actions=[(Bpod.OutputChannels.Valve, rewardValve)])  # Reward correct␣
↪choice
    sma.add_state(
        state_name='Punish',
        state_timer=3,
        state_change_conditions={Bpod.Events.Tup: 'exit'},
        output_actions=[(Bpod.OutputChannels.LED, 1), (Bpod.OutputChannels.LED, 2),␣
↪(Bpod.OutputChannels.LED, 3)])  # Signal incorrect choice
```

After configuring the state machine, we send it to the Bpod device by calling the method *send_state_machine*. We are then ready to run the next trial, by calling the *run_state_machine* method. On run completion, we can print the data available for the current trial including events and states.

```
    my_bpod.send_state_machine(sma)  # Send state machine description to Bpod device

    print("Waiting for poke. Reward: ", 'left' if thisTrialType == 1 else 'right')

    my_bpod.run_state_machine(sma)  # Run state machine

    print("Current trial info: ", my_bpod.session.current_trial)
```

4. Stop Bpod execution

Finally, after the loop finishes, we can stop Bpod execution.

```
my_bpod.close()  # Disconnect Bpod and perform post-run actions
```

See also:

pybpodapi.bpod.bpod_base.BpodBase

pybpodapi.state_machine.state_machine_base.StateMachineBase

pybpodapi.state_machine.state_machine_base.StateMachineBase.add_state()

pybpodapi.bpod.hardware.output_channels.OutputChannel

pybpodapi.bpod.hardware.events.EventName

pybpodapi.bpod.bpod_base.BpodBase.send_state_machine()

pybpodapi.bpod.bpod_base.BpodBase.run_state_machine()

pybpodapi.bpod.bpod_base.BpodBase.close()

2.3.3 Try the example

You can try the full example by *installing* and *running* this library.

Full example (*function_examples/add_trial_events.py*):

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

"""
Demonstration of AddTrialEvents used in a simple visual 2AFC session.
AddTrialEvents formats each trial's data in a human-readable struct, and adds to␣
↪myBpod.data (to save to disk later)
Connect noseports to ports 1-3.

Example adapted from Josh Sanders' original version on Sanworks Bpod repository
"""

import random
from pybpodapi.protocol import Bpod, StateMachine

my_bpod = Bpod()

nTrials = 5
trialTypes = [1, 2]  # 1 (rewarded left) or 2 (rewarded right)
```

(continues on next page)

(continued from previous page)

```
for i in range(nTrials):  # Main loop
        print('Trial: ', i + 1)

        thisTrialType = random.choice(trialTypes)  # Randomly choose trial type
        if thisTrialType == 1:
                stimulus = Bpod.OutputChannels.PWM1  # set stimulus channel for trial␣
↪type 1
                leftAction = 'Reward'
                rightAction = 'Punish'
                rewardValve = 1
        elif thisTrialType == 2:
                stimulus = Bpod.OutputChannels.PWM3  # set stimulus channel for trial␣
↪type 1
                leftAction = 'Punish'
                rightAction = 'Reward'
                rewardValve = 3

        sma = StateMachine(my_bpod)

        sma.add_state(
                state_name='WaitForPort2Poke',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port2In: 'FlashStimulus'},
                output_actions=[(Bpod.OutputChannels.PWM2, 255)])
        sma.add_state(
                state_name='FlashStimulus',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'WaitForResponse'},
                output_actions=[(stimulus, 255)])
        sma.add_state(
                state_name='WaitForResponse',
                state_timer=1,
                state_change_conditions={Bpod.Events.Port1In: leftAction, Bpod.Events.
↪Port3In: rightAction},
                output_actions=[])
        sma.add_state(
                state_name='Reward',
                state_timer=0.1,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.Valve, rewardValve)])  # Reward␣
↪correct choice
        sma.add_state(
                state_name='Punish',
                state_timer=3,
                state_change_conditions={Bpod.Events.Tup: 'exit'},
                output_actions=[(Bpod.OutputChannels.LED, 1), (Bpod.OutputChannels.
↪LED, 2), (Bpod.OutputChannels.LED, 3)])  # Signal incorrect choice

        my_bpod.send_state_machine(sma)  # Send state machine description to Bpod␣
↪device

        print("Waiting for poke. Reward: ", 'left' if thisTrialType == 1 else 'right')

        my_bpod.run_state_machine(sma)  # Run state machine

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

(continues on next page)

(continued from previous page)

```
my_bpod.close()  # Disconnect Bpod
```

2.4 Manual control of Bpod

Using pybpod-api, you can directly interact with the Bpod hardware. This may be useful for testing and debug purposes.

After *installing* pybpod-api, open a python terminal and run the following commands:

```
from pybpodapi.protocol import Bpod # import Bpod main class

# connect to bpod

my_bpod = Bpod() # Start bpod

# set poke led connected on port 1 to maximum intensity
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=255)

# set poke led connected on port 1 to half intensity
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=128)

# turn off poke led connected on port 1
my_bpod.manual_override(Bpod.ChannelTypes.OUTPUT, Bpod.ChannelNames.PWM, channel_
↪number=1, value=128)

# disconnect from bpod
my_bpod.close()
```

See also:

For more available commands, please refer to:

- *pybpodapi.bpod.bpod_com_protocol.BpodCOMProtocol.manual_override()*
- *pybpodapi.bpod.hardware.channels.ChannelType*
- *pybpodapi.bpod.hardware.channels.ChannelName*

2.5 Firmware update

2.5.1 How to update Bpod firmware

- Download Arduino latest version, extract the zip folder and save the extracted folder somewhere permanent on your PC.
- Plug the Bpod device into a USB port of the computer.
- (*Windows only*) If the drivers are not yet installed (or if you're not sure), follow Arduino Due's Windows driver installation page here.
- Open the Arduino program folder and run Arduino.exe.

- Install support for Arduino Due (if you haven't done this already):
  - From the "Tools" menu, choose "Board" and then "Boards Manager".
  - In the boards manager, install "Arduino SAM boards (32-bits ARM Cortex M3).
  - Restart Arduino
- From the "Tools" menu, choose "Board" and then "Arduino Due (Programming Port)".
- From the "Serial Port" menu, choose "COMX" (win) or "/dev/ttySX" (linux) where X is the port number.To find your port number in Windows, choose "Start" and type "device manager" in the search window. In the device manager, scroll down to "Ports (COM & LPT)" and expand the menu. The COM port will be listed as "Arduino Due Programming Port (COMX)".
- From the File menu in Arduino, choose "Open" and select the firmware project. A new window should open with the firmware. [Download the firmware here](https://bitbucket.org/fchampalimaud/bpod-firmware)
- In the new window, click the "upload" button (the right-pointing arrow under "edit").

If all went well, the green progress indicator should finish, and be replaced with a message: "Done uploading". Below that, in orange text, it should appear the message "Verify successful".

2.6 Output action codes

2.6.1 Overview

Output actions are specified via string labels. Although you can manually specify these values, **we strongly advise to use the API available labels instead** (*pybpodapi.bpod.hardware.output_channels.OutputChannel*).

Standard port setup LED control

For ease of use and convenience, you can use 'LED' label to control LEDs. This is equivalent to control PWM channels.

Action label	Action value	Example 'OutputActions' for state matrix
LED	LED Port number (1-8)	(OutputChannel.LED, 1) # Port 1 LED to full brightness (Equivalent to (OutputChannel.PWM1, 255))

Solenoid valve control

You can control one valve per each standard port setup.

Action label	Action value	Example 'OutputActions' for state matrix
ValveState or Valve	8 Bits = 8 valves	(OutputChannel.ValveState, 128) # Set valve 7 set to "open"

Pulse width modulated output line control (LED in standard port setup)

Action label	Action value	Example 'OutputActions' for state matrix
PWM1	Byte ~ duty cycle	(OutputChannel.PWM1, 255) # Set PWM 1 to 100% duty cycle / Port 1 LED to full brightness
...	...	...
PWM8	Byte ~ duty cycle	(OutputChannel.PWM8, 128) # Set PWM 8 to 50% duty cycle / Port 8 LED to half brightness

BNC output logic control

Action label	Action value	Example 'OutputActions' for state matrix
BNC1	2 Bits = 2 channels	(OutputChannel.BNC1, 3) # TODO:
BNC2	2 Bits = 2 channels	(OutputChannel.BNC2, 3) # TODO:

Wire output logic control

Action label	Action value	Example 'OutputActions' for state matrix
Wire1	4 Bits = 4 channels	(OutputChannel.Wire1, 5) # TODO:
Wire2	4 Bits = 4 channels	(OutputChannel.Wire2, 5) # TODO:
Wire3	4 Bits = 4 channels	(OutputChannel.Wire3, 5) # TODO:
Wire4	4 Bits = 4 channels	(OutputChannel.Wire4, 5) # TODO:

Hardware serial ports 1, 2 and 3

Action label	Action value	Example 'OutputActions' for state matrix
Serial1	Byte to send	(OutputChannel.Serial1, 129) # Send byte 129 to serial port 1
Serial2	Byte to send	(OutputChannel.Serial2, 129) # Send byte 129 to serial port 2
Serial3	Byte to send	(OutputChannel.Serial3, 129) # Send byte 129 to serial port 3

USB serial port byte

Action label	Action value	Example 'OutputActions' for state matrix
SoftCode	Byte to send	(OutputChannel.SoftCode, 129) # Send byte 129 to be handled by the governing computer

Global timer control

Action label	Action value	Example 'OutputActions' for state matrix
GlobalTimerTrig	Timer# to start (of 5)	(OutputChannel.GlobalTimerTrig, 2) # Start global timer 2
GlobalTimerCancel	Timer# to cancel (of 5)	(OutputChannel.GlobalTimerCancel, 2) # Start global timer 2

Global counter control

Action label	Action value	Example 'OutputActions' for state matrix
GlobalCounterReset	Counter# to reset (of 5)	(OutputChannel.GlobalCounterReset, 3) # Reset global counter 3

2.7 Input event codes

2.7.1 Overview

Input events are specified via string labels. Although you can manually specify these values, **we strongly advise to use the API available labels instead** (*pybpodapi.bpod.hardware.events.EventName*).

Port IR sensor events

Byte code	Event syntax
1	Port1In
2	Port1Out
3	Port2In
4	Port2Out
5	Port3In
6	Port3Out
7	Port4In
8	Port4Out
9	Port5In
10	Port5Out
11	Port6In
12	Port6Out
13	Port7In
14	Port7Out
15	Port8In
16	Port8Out

BNC input channel logic

Byte code	Event syntax
17	BNC1High
18	BNC1Low
19	BNC2High
20	BNC2Low

Wire input channel logic

Byte code	Event syntax
21	Wire1High
22	Wire1Low
23	Wire2High
24	Wire2Low
25	Wire3High
26	Wire3Low
27	Wire4High
28	Wire4Low

USB soft codes

29	SoftCode1
30	SoftCode2
31	SoftCode3
32	SoftCode4
33	SoftCode5
34	SoftCode6
35	SoftCode7
36	SoftCode8
37	SoftCode9
38	SoftCode10

State timer elapsed

40	Tup

Global timer elapsed

41	GlobalTimer1_End
42	GlobalTimer2_End
43	GlobalTimer3_End
44	GlobalTimer4_End
45	GlobalTimer5_End

Global counter threshold exceeded

46	GlobalCounter1_End
47	GlobalCounter2_End
48	GlobalCounter3_End
49	GlobalCounter4_End
50	GlobalCounter5_End

2.8 pybpod-api

2.8.1 bpod— Bpod

Bpod class

The Bpod class is composed by the next subclasses:

bpod_base— Bpod Base

Implementation

class pybpodapi.bpod.bpod_base.**BpodBase**(*serial_port=None*, *sync_channel=None*, *sync_mode=None*, *net_port=None*)

API to interact with Bpod

Variables

- ***session*** (Session) – Session for this bpod running experiment
- ***hardware*** (Hardware) – Hardware object representing Bpod hardware
- **message_api** (*MessageAPI*) – Abstracts communication with Bpod box
- **new_sma_sent** (*bool*) – whether a new state machine was already uploaded to Bpod box

class Events

class OutputChannels

class ChannelTypes

class ChannelNames

loop_handler()
handler that will execute on every loop when the bpod is running

open()
Starts Bpod.

Connect to Bpod board through serial port, test handshake, retrieve firmware version, retrieve hardware description, enable input ports and configure channel synchronization.

Example:

```
my_bpod = Bpod().open("/dev/tty.usbmodem1293", "/Users/John/Desktop/bpod_
↪workspace", "2afc_protocol")
```

Parameters

- **serial_port** (*str*) – serial port to connect
- **workspace_path** (*str*) – path for bpod output files (no folders will be created)
- **session_name** (*str*) – this name will be used for output files

- **baudrate [optional]** (*int*) – baudrate for serial connection
- **sync_channel [optional]** (*int*) – Serial synchronization channel: 255 = no sync, otherwise set to a hardware channel number
- **sync_mode [optional]** (*int*) – Serial synchronization mode: 0 = flip logic every trial, 1 = every state

Returns Bpod object created

Return type pybpodapi.model.bpod

close()
Close connection with Bpod

send_state_machine(*sma*, *run_asap=None*)
Builds message and sends state machine to Bpod

Parameters **sma** (*pybpodapi.model.state_machine*) – initialized state machine

load_serial_message(*serial_channel*, *message_ID*, *serial_message*)
Load serial message on Bpod

Parameters

- **serial_channel** (*int*) – Serial port to send, 1, 2 or 3
- **message_ID** (*int*) – Unique id for the message. Should be between 1 and 255
- **serial_message** (*list(int)*) – Message to send. The message should be bigger than 3 bytes.

reset_serial_messages()
Reset serial messages to equivalent byte codes (i.e. message# 4 = one byte, 0x4)

softcode_handler_function(*data*)
Users can override this function directly on the protocol to handle a softcode from Bpod

Parameters **data** (*int*) – soft code number

find_module_by_name(*name*)
Search for a module by name

_BpodBase__process_opcode(*sma*, *opcode*, *data*, *state_change_indexes*)
Process data from bpod board given an opcode

In original bpod, sma.raw_data == raw_events

Parameters

- **sma** – state machine object
- **opcode** (*int*) – opcode number
- **data** – data from bpod board
- **state_change_indexes** –

Returns

_BpodBase__update_timestamps(*sma*, *state_change_indexes*)
Read timestamps from Bpod and update state machine info

Parameters

- **sma** (*StateMachine*) –

- **state_change_indexes** (*list*) –

bpod_com_protocol— Bpod Communication Protocol

Implementation

class pybpodapi.bpod.bpod_com_protocol.**BpodCOMProtocol**(*serial_port=None*, *sync_channel=None*, *sync_mode=None*)

Define command actions that can be requested to Bpod device.

Private attributes

_arcom *pybpodapi.com.arcom.ArCOM*

ArCOM object that performs serial communication.

Methods

open()

Starts Bpod.

Connect to Bpod board through serial port, test handshake, retrieve firmware version, retrieve hardware description, enable input ports and configure channel synchronization.

Example:

```
my_bpod = Bpod().open("/dev/tty.usbmodem1293", "/Users/John/Desktop/bpod_
↪workspace", "2afc_protocol")
```

Parameters

- **serial_port** (*str*) – serial port to connect
- **workspace_path** (*str*) – path for bpod output files (no folders will be created)
- **session_name** (*str*) – this name will be used for output files
- **baudrate [optional]** (*int*) – baudrate for serial connection
- **sync_channel [optional]** (*int*) – Serial synchronization channel: 255 = no sync, otherwise set to a hardware channel number
- **sync_mode [optional]** (*int*) – Serial synchronization mode: 0 = flip logic every trial, 1 = every state

Returns Bpod object created

Return type pybpodapi.model.bpod

close()

Close connection with Bpod

manual_override(*channel_type*, *channel_name*, *channel_number*, *value*)

Manually override a Bpod channel

Parameters

- **channel_type** (ChannelType) – channel type input or output
- **channel_name** (ChannelName) – channel name like PWM, Valve, etc.
- **channel_number** –

- **value** (*int*) – value to write on channel

_bpodcom_connect(*serial_port*, *baudrate=115200*, *timeout=1*)
Connect to Bpod using serial connection

Parameters

- **serial_port** (*str*) – serial port to connect
- **baudrate** (*int*) – baudrate for serial connection
- **timeout** (*float*) – timeout which controls the behavior of read()

_bpodcom_disconnect()
Signal Bpod device to disconnect now

_bpodcom_handshake()
Test connectivity by doing an handshake

Returns True if handshake received, False otherwise

Return type bool

_bpodcom_firmware_version()
Request firmware and machine type from Bpod

Returns firmware and machine type versions

Return type int, int

_bpodcom_reset_clock()
Reset session clock

_bpodcom_stop_trial()
Pause ongoing trial (We recommend using computer-side pauses between trials, to keep data uniform)

_bpodcom_pause_trial()
Pause ongoing trial (We recommend using computer-side pauses between trials, to keep data uniform)

_bpodcom_resume_trial()
Pause ongoing trial (We recommend using computer-side pauses between trials, to keep data uniform)

_bpodcom_get_timestamp_transmission()
Return timestamp transmission scheme

_bpodcom_hardware_description(*hardware*)
Request hardware description from Bpod

Parameters **hardware** (*Hardware*) – hardware

_bpodcom_enable_ports(*hardware*)
Enable input ports on Bpod device

Parameters **inputs_enabled** (*list[int]*) – list of inputs to be enabled (0 = disabled, 1 = enabled)

Return type bool

_bpodcom_set_sync_channel_and_mode(*sync_channel*, *sync_mode*)
Request sync channel and sync mode configuration

Parameters

- **sync_channel** (*int*) – 255 = no sync, otherwise set to a hardware channel number
- **sync_mode** (*int*) – 0 = flip logic every trial, 1 = every state

Return type bool

_bpodcom_echo_softcode(*softcode*)
Send soft code

_bpodcom_manual_override_exec_event(*event_index*, *event_data*)
Send soft code

_bpodcom_override_input_state(*channel_number*, *value*)
Manually set digital value on channel

Parameters

- **channel_number** (*int*) – number of Bpod port
- **value** (*int*) – value to be written

_bpodcom_send_softcode(*softcode*)
Send soft code

_bpodcom_send_state_machine(*message*)
Sends state machine to Bpod

Parameters

- **message** (*list (int)*) – TODO
- **ThirtyTwoBitMessage** (*list (int)*) – TODO

_bpodcom_run_state_machine()
Request to run state machine now

_bpodcom_read_trial_start_timestamp_seconds()
A new incoming timestamp message is available. Read trial start timestamp in millseconds and convert to seconds.

Returns trial start timestamp in milliseconds

Return type float

_bpodcom_state_machine_installation_status()
Confirm if new state machine was correctly installed

Return type bool

data_available()
Finds out if there is data received from Bpod

Return type bool

_bpodcom_read_opcode_message()
A new incoming opcode message is available. Read opcode code and data.

Returns opcode and data

Return type tuple(int, int)

_bpodcom_read_alltimestamps()
A new incoming timestamps message is available. Read number of timestamps to be sent and then read timestamps array.

Returns timestamps array

Return type list(float)

_bpodcom_read_current_events(*n_events*)
A new incoming events message is available. Read number of timestamps to be sent and then read timestamps array.

Parameters **n_events** (*int*) – number of events to read

Returns a list with events

Return type list(int)

_bpodcom_load_serial_message(*serial_channel*, *message_id*, *serial_message*, *n_messages*)
Load serial message on channel

:param TODO :rtype: bool

_bpodcom_reset_serial_messages()
Reset serial messages on Bpod device

Return type bool

_bpodcom_override_digital_hardware_state(*channel_number*, *value*)
Manually set digital value on channel

Parameters

- **channel_number** (*int*) – number of Bpod port
- **value** (*int*) – value to be written

_bpodcom_send_byte_to_hardware_serial(*channel_number*, *value*)
Send byte to hardware serial channel 1-3

Parameters

- **channel_number** (*int*) –
- **value** (*int*) – value to be written

bpod_com_protocol_modules— Bpod Modules Communication Protocol

Implementation

class pybpodapi.bpod.bpod_com_protocol_modules.**BpodCOMProtocolModules**(*serial_port=None*, *sync_channel=None*, *sync_mode=None*)

Define command actions that can be requested to Bpod device.

Private attributes

_arcom *pybpodapi.com.arcom.ArCOM*

ArCOM object that performs serial communication.

Methods

bpod_io— Bpod IO

Implementation

class pybpodapi.bpod.bpod_io.**BpodIO**(*serial_port=None*, *session_name=None*, *sync_mode=None*, *workspace_path=None*, *sync_channel=None*)

Bpod I/O logic.

close()

Close connection with Bpod

Inheritance

Hardware module

hardware— Hardware

hardware — Hardware Description For Bpod Device

Overview

Hardware description ...

Implementation

class pybpodapi.bpod.hardware.hardware.**Hardware**

Represents an hardware description based on information received from the current connected Bpod deviced.

setup(*modules*)

Set up hardware based on hardware description obtained from Bpod device

Parameters **hw_info_container** (*HardwareInfoContainer*) – hardware parameters received from Bpod

channels — Bpod channel configuration

Overview

The purpose of these classes is to abstract low level numbers and IDs for identifying channel names and types.

Implementation

class pybpodapi.bpod.hardware.channels.**ChannelType**
Define if channel type is input or output. These values must be set according to Bpod firmware specification.

INPUT = 1
Input channel

OUTPUT = 2
Output channel

class pybpodapi.bpod.hardware.channels.**ChannelName**
Available channel names. These values must be set according to Bpod firmware specification.

PWM = 'PWM'
Analog channel with PWM support (e.g. Led)

VALVE = 'Valve'
Analog channel for connecting a valve

BNC = 'BNC'
BNC channel

WIRE = 'Wire'
Wire channel

SERIAL = 'Serial'
Serial channel

events — Bpod input events

Overview

Input events available on Bpod box that can trigger a state change.

Implementation

class pybpodapi.bpod.hardware.events.**EventName**
Input event codes These values must be set according to Bpod firmware specification.

Serial1_1 = 'Serial1_1'
Serial1_1

Serial1_2 = 'Serial1_2'
Serial1_2

Serial1_3 = 'Serial1_3'
Serial1_3

Serial1_4 = 'Serial1_4'
Serial1_4

Serial1_5 = 'Serial1_5'
Serial1_5

Serial1_6 = 'Serial1_6'
Serial1_6

Serial1_7 = 'Serial1_7'
Serial1_7

Serial1_8 = 'Serial1_8'
Serial1_8

Serial1_9 = 'Serial1_9'
Serial1_9

Serial1_10 = 'Serial1_10'
Serial1_10

Serial1_11 = 'Serial1_11'
Serial1_11

Serial1_12 = 'Serial1_12'
Serial1_12

Serial1_13 = 'Serial1_13'
Serial1_13

Serial1_14 = 'Serial1_14'
Serial1_14

Serial1_15 = 'Serial1_15'
Serial1_15

Serial1_16 = 'Serial1_16'
Serial1_16

Serial1_17 = 'Serial1_17'
Serial1_17

Serial1_18 = 'Serial1_18'
Serial1_18

Serial1_19 = 'Serial1_19'
Serial1_19

Serial1_20 = 'Serial1_20'
Serial1_20

Serial1_21 = 'Serial1_21'
Serial1_21

Serial1_22 = 'Serial1_22'
Serial1_22

Serial1_23 = 'Serial1_23'
Serial1_23

Serial1_24 = 'Serial1_24'
Serial1_24

Serial1_25 = 'Serial1_25'
Serial1_25

Serial1_26 = 'Serial1_26'
Serial1_26

Serial1_27 = 'Serial1_27'
Serial1_27

Serial1_28 = 'Serial1_28'
Serial1_28

Serial1_29 = 'Serial1_29'
Serial1_29

Serial1_30 = 'Serial1_30'
Serial1_30

Serial1_31 = 'Serial1_31'
Serial1_31

Serial1_32 = 'Serial1_32'
Serial1_32

Serial1_33 = 'Serial1_33'
Serial1_33

Serial1_34 = 'Serial1_34'
Serial1_34

Serial1_35 = 'Serial1_35'
Serial1_35

Serial1_36 = 'Serial1_36'
Serial1_36

Serial1_37 = 'Serial1_37'
Serial1_37

Serial1_38 = 'Serial1_38'
Serial1_38

Serial1_39 = 'Serial1_39'
Serial1_39

Serial1_40 = 'Serial1_40'
Serial1_40

Serial1_41 = 'Serial1_41'
Serial1_41

Serial1_42 = 'Serial1_42'
Serial1_42

Serial1_43 = 'Serial1_43'
Serial1_43

Serial1_44 = 'Serial1_44'
Serial1_44

Serial1_45 = 'Serial1_45'
Serial1_45

Serial1_46 = 'Serial1_46'
Serial1_46

Serial1_47 = 'Serial1_47'
Serial1_47

Serial1_48 = 'Serial1_48'
Serial1_48

Serial1_49 = 'Serial1_49'
Serial1_49

Serial1_50 = 'Serial1_50'
Serial1_50

Serial1_51 = 'Serial1_51'
Serial1_51

Serial1_52 = 'Serial1_52'
Serial1_52

Serial1_53 = 'Serial1_53'
Serial1_53

Serial1_54 = 'Serial1_54'
Serial1_54

Serial1_55 = 'Serial1_55'
Serial1_55

Serial1_56 = 'Serial1_56'
Serial1_56

Serial1_57 = 'Serial1_57'
Serial1_57

Serial1_58 = 'Serial1_58'
Serial1_58

Serial1_59 = 'Serial1_59'
Serial1_59

Serial1_60 = 'Serial1_60'
Serial1_60

Serial2_1 = 'Serial2_1'
Serial2_1

Serial2_2 = 'Serial2_2'
Serial2_2

Serial2_3 = 'Serial2_3'
Serial2_3

Serial2_4 = 'Serial2_4'
Serial2_4

Serial2_5 = 'Serial2_5'
Serial2_5

Serial2_6 = 'Serial2_6'
Serial2_6

Serial2_7 = 'Serial2_7'
Serial2_7

Serial2_8 = 'Serial2_8'
Serial2_8

Serial2_9 = 'Serial2_9'
Serial2_9

Serial2_10 = 'Serial2_10'
Serial2_10

Serial2_11 = 'Serial2_11'
Serial2_11

Serial2_12 = 'Serial2_12'
Serial2_12

Serial2_13 = 'Serial2_13'
Serial2_13

Serial2_14 = 'Serial2_14'
Serial2_14

Serial2_15 = 'Serial2_15'
Serial2_15

Serial2_16 = 'Serial2_16'
Serial2_16

Serial2_17 = 'Serial2_17'
Serial2_17

Serial2_18 = 'Serial2_18'
Serial2_18

Serial2_19 = 'Serial2_19'
Serial2_19

Serial2_20 = 'Serial2_20'
Serial2_20

Serial2_21 = 'Serial2_21'
Serial2_21

Serial2_22 = 'Serial2_22'
Serial2_22

Serial2_23 = 'Serial2_23'
Serial2_23

Serial2_24 = 'Serial2_24'
Serial2_24

Serial2_25 = 'Serial2_25'
Serial2_25

Serial2_26 = 'Serial2_26'
Serial2_26

Serial2_27 = 'Serial2_27'
Serial2_27

Serial2_28 = 'Serial2_28'
Serial2_28

Serial2_29 = 'Serial2_29'
Serial2_29

Serial2_30 = 'Serial2_30'
Serial2_30

Serial2_31 = 'Serial2_31'
Serial2_31

Serial2_32 = 'Serial2_32'
Serial2_32

Serial2_33 = 'Serial2_33'
Serial2_33

Serial2_34 = 'Serial2_34'
Serial2_34

Serial2_35 = 'Serial2_35'
Serial2_35

Serial2_36 = 'Serial2_36'
Serial2_36

Serial2_37 = 'Serial2_37'
Serial2_37

Serial2_38 = 'Serial2_38'
Serial2_38

Serial2_39 = 'Serial2_39'
Serial2_39

Serial2_40 = 'Serial2_40'
Serial2_40

Serial2_41 = 'Serial2_41'
Serial2_41

Serial2_42 = 'Serial2_42'
Serial2_42

Serial2_43 = 'Serial2_43'
Serial2_43

Serial2_44 = 'Serial2_44'
Serial2_44

Serial2_45 = 'Serial2_45'
Serial2_45

Serial2_46 = 'Serial2_46'
Serial2_46

Serial2_47 = 'Serial2_47'
Serial2_47

Serial2_48 = 'Serial2_48'
Serial2_48

Serial2_49 = 'Serial2_49'
Serial2_49

Serial2_50 = 'Serial2_50'
Serial2_50

Serial2_51 = 'Serial2_51'
Serial2_51

Serial2_52 = 'Serial2_52'
Serial2_52

Serial2_53 = 'Serial2_53'
Serial2_53

Serial2_54 = 'Serial2_54'
Serial2_54

Serial2_55 = 'Serial2_55'
Serial2_55

Serial2_56 = 'Serial2_56'
Serial2_56

Serial2_57 = 'Serial2_57'
Serial2_57

Serial2_58 = 'Serial2_58'
Serial2_58

Serial2_59 = 'Serial2_59'
Serial2_59

Serial2_60 = 'Serial2_60'
Serial2_60

Serial3_1 = 'Serial3_1'
Serial3_1

Serial3_2 = 'Serial3_2'
Serial3_2

Serial3_3 = 'Serial3_3'
Serial3_3

Serial3_4 = 'Serial3_4'
Serial3_4

Serial3_5 = 'Serial3_5'
Serial3_5

Serial3_6 = 'Serial3_6'
Serial3_6

Serial3_7 = 'Serial3_7'
Serial3_7

Serial3_8 = 'Serial3_8'
Serial3_8

Serial3_9 = 'Serial3_9'
Serial3_9

Serial3_10 = 'Serial3_10'
Serial3_10

Serial3_11 = 'Serial3_11'
Serial3_11

Serial3_12 = 'Serial3_12'
Serial3_12

Serial3_13 = 'Serial3_13'
Serial3_13

Serial3_14 = 'Serial3_14'
Serial3_14

Serial3_15 = 'Serial3_15'
Serial3_15

Serial3_16 = 'Serial3_16'
Serial3_16

Serial3_17 = 'Serial3_17'
Serial3_17

Serial3_18 = 'Serial3_18'
Serial3_18

Serial3_19 = 'Serial3_19'
Serial3_19

Serial3_20 = 'Serial3_20'
Serial3_20

Serial3_21 = 'Serial3_21'
Serial3_21

Serial3_22 = 'Serial3_22'
Serial3_22

Serial3_23 = 'Serial3_23'
Serial3_23

Serial3_24 = 'Serial3_24'
Serial3_24

Serial3_25 = 'Serial3_25'
Serial3_25

Serial3_26 = 'Serial3_26'
Serial3_26

Serial3_27 = 'Serial3_27'
Serial3_27

Serial3_28 = 'Serial3_28'
Serial3_28

Serial3_29 = 'Serial3_29'
Serial3_29

Serial3_30 = 'Serial3_30'
Serial3_30

Serial3_31 = 'Serial3_31'
Serial3_31

Serial3_32 = 'Serial3_32'
Serial3_32

Serial3_33 = 'Serial3_33'
Serial3_33

Serial3_34 = 'Serial3_34'
Serial3_34

Serial3_35 = 'Serial3_35'
Serial3_35

Serial3_36 = 'Serial3_36'
Serial3_36

Serial3_37 = 'Serial3_37'
Serial3_37

Serial3_38 = 'Serial3_38'
Serial3_38

Serial3_39 = 'Serial3_39'
Serial3_39

Serial3_40 = 'Serial3_40'
Serial3_40

Serial3_41 = 'Serial3_41'
Serial3_41

Serial3_42 = 'Serial3_42'
Serial3_42

Serial3_43 = 'Serial3_43'
Serial3_43

Serial3_44 = 'Serial3_44'
Serial3_44

Serial3_45 = 'Serial3_45'
Serial3_45

Serial3_46 = 'Serial3_46'
Serial3_46

Serial3_47 = 'Serial3_47'
Serial3_47

Serial3_48 = 'Serial3_48'
Serial3_48

Serial3_49 = 'Serial3_49'
Serial3_49

Serial3_50 = 'Serial3_50'
Serial3_50

Serial3_51 = 'Serial3_51'
Serial3_51

Serial3_52 = 'Serial3_52'
Serial3_52

Serial3_53 = 'Serial3_53'
Serial3_53

Serial3_54 = 'Serial3_54'
Serial3_54

Serial3_55 = 'Serial3_55'
Serial3_55

Serial3_56 = 'Serial3_56'
Serial3_56

Serial3_57 = 'Serial3_57'
Serial3_57

Serial3_58 = 'Serial3_58'
Serial3_58

Serial3_59 = 'Serial3_59'
Serial3_59

Serial3_60 = 'Serial3_60'
Serial3_60

SoftCode1 = 'SoftCode1'
SoftCode1

SoftCode2 = 'SoftCode2'
SoftCode2

SoftCode3 = 'SoftCode3'
SoftCode3

SoftCode4 = 'SoftCode4'
SoftCode4

SoftCode5 = 'SoftCode5'
SoftCode5

SoftCode6 = 'SoftCode6'
SoftCode6

SoftCode7 = 'SoftCode7'
SoftCode7

SoftCode8 = 'SoftCode8'
SoftCode8

SoftCode9 = 'SoftCode9'
SoftCode9

SoftCode10 = 'SoftCode10'
SoftCode10

SoftCode11 = 'SoftCode11'
SoftCode11

SoftCode12 = 'SoftCode12'
SoftCode12

SoftCode13 = 'SoftCode13'
SoftCode13

SoftCode14 = 'SoftCode14'
SoftCode14

SoftCode15 = 'SoftCode15'
SoftCode15

SoftCode16 = 'SoftCode16'
SoftCode16

SoftCode17 = 'SoftCode17'
SoftCode17

SoftCode18 = 'SoftCode18'
SoftCode18

SoftCode19 = 'SoftCode19'
SoftCode19

SoftCode20 = 'SoftCode20'
SoftCode20

SoftCode21 = 'SoftCode21'
SoftCode21

SoftCode22 = 'SoftCode22'
SoftCode22

SoftCode23 = 'SoftCode23'
SoftCode23

SoftCode24 = 'SoftCode24'
SoftCode24

SoftCode25 = 'SoftCode25'
SoftCode25

SoftCode26 = 'SoftCode26'
SoftCode26

SoftCode27 = 'SoftCode27'
SoftCode27

SoftCode28 = 'SoftCode28'
SoftCode28

SoftCode29 = 'SoftCode29'
SoftCode29

SoftCode30 = 'SoftCode30'
SoftCode30

SoftCode31 = 'SoftCode31'
SoftCode31

SoftCode32 = 'SoftCode32'
SoftCode32

SoftCode33 = 'SoftCode33'
SoftCode33

SoftCode34 = 'SoftCode34'
SoftCode34

SoftCode35 = 'SoftCode35'
SoftCode35

SoftCode36 = 'SoftCode36'
SoftCode36

SoftCode37 = 'SoftCode37'
SoftCode37

SoftCode38 = 'SoftCode38'
SoftCode38

SoftCode39 = 'SoftCode39'
SoftCode39

SoftCode40 = 'SoftCode40'
SoftCode40

SoftCode41 = 'SoftCode41'
SoftCode41

SoftCode42 = 'SoftCode42'
SoftCode42

SoftCode43 = 'SoftCode43'
SoftCode43

SoftCode44 = 'SoftCode44'
SoftCode44

SoftCode45 = 'SoftCode45'
SoftCode45

SoftCode46 = 'SoftCode46'
SoftCode46

SoftCode47 = 'SoftCode47'
SoftCode47

SoftCode48 = 'SoftCode48'
SoftCode48

SoftCode49 = 'SoftCode49'
SoftCode49

SoftCode50 = 'SoftCode50'
SoftCode50

SoftCode51 = 'SoftCode51'
SoftCode51

SoftCode52 = 'SoftCode52'
SoftCode52

SoftCode53 = 'SoftCode53'
SoftCode53

SoftCode54 = 'SoftCode54'
SoftCode54

SoftCode55 = 'SoftCode55'
SoftCode55

SoftCode56 = 'SoftCode56'
SoftCode56

SoftCode57 = 'SoftCode57'
SoftCode57

SoftCode58 = 'SoftCode58'
SoftCode58

SoftCode59 = 'SoftCode59'
SoftCode59

SoftCode60 = 'SoftCode60'
SoftCode60

BNC1High = 'BNC1High'
BNC1In

BNC1Low = 'BNC1Low'
BNC1Out

BNC2High = 'BNC2High'
BNC2In

BNC2Low = 'BNC2Low'
BNC2Out

Wire1High = 'Wire1High'
Wire1In

Wire1Low = 'Wire1Low'
Wire1Out

Wire2High = 'Wire2High'
Wire2In

Wire2Low = 'Wire2Low'
Wire2Out

Port1In = 'Port1In'
Input port 1

Port1Out = 'Port1Out'
Output port 1

Port2In = 'Port2In'
Input port 2

Port2Out = 'Port2Out'
Output port 2

Port3In = 'Port3In'
Input port 3

Port3Out = 'Port3Out'
Output port 3

Port4In = 'Port4In'
Input port 4

Port4Out = 'Port4Out'
Output port 4

Port5In = 'Port5In'
Input port 5

Port5Out = 'Port5Out'
Output port 5

Port6In = 'Port6In'
Input port 6

Port6Out = 'Port6Out'
Output port 6

Port7In = 'Port7In'
Input port 7

Port7Out = 'Port7Out'
Output port 7

Port8In = 'Port8In'
Input port 8

Port8Out = 'Port8Out'
Output port 8

GlobalTimer1_Start = 'GlobalTimer1_Start'
GlobalTimer1_Start

GlobalTimer2_Start = 'GlobalTimer2_Start'
GlobalTimer2_Start

GlobalTimer3_Start = 'GlobalTimer3_Start'
GlobalTimer3_Start

GlobalTimer4_Start = 'GlobalTimer4_Start'
GlobalTimer4_Start

GlobalTimer5_Start = 'GlobalTimer5_Start'
GlobalTimer5_Start

GlobalTimer1_End = 'GlobalTimer1_End'
GlobalTimer1_End

GlobalTimer2_End = 'GlobalTimer2_End'
GlobalTimer2_End

GlobalTimer3_End = 'GlobalTimer3_End'
GlobalTimer3_End

GlobalTimer4_End = 'GlobalTimer4_End'
GlobalTimer4_End

GlobalTimer5_End = 'GlobalTimer5_End'
GlobalTimer5_End

GlobalCounter1_End = 'GlobalCounter1_End'
GlobalCounter1_End

GlobalCounter2_End = 'GlobalCounter2_End'
GlobalCounter2_End

GlobalCounter3_End = 'GlobalCounter3_End'
GlobalCounter3_End

GlobalCounter4_End = 'GlobalCounter4_End'
GlobalCounter4_End

GlobalCounter5_End = 'GlobalCounter5_End'
GlobalCounter5_End

Condition1 = 'Condition1'
Condition1

Condition2 = 'Condition2'
Condition2

Condition3 = 'Condition3'
Condition3

Condition4 = 'Condition4'
Condition4

Condition5 = 'Condition5'
Condition5

Condition6 = 'Condition6'
Condition6

Condition7 = 'Condition7'
Condition7

Condition8 = 'Condition8'
Condition8

Condition9 = 'Condition9'
Condition9

Condition10 = 'Condition10'
Condition10

Condition11 = 'Condition11'
Condition11

Condition12 = 'Condition12'
Condition12

Condition13 = 'Condition13'
Condition13

Condition14 = 'Condition14'
Condition14

Condition15 = 'Condition15'
Condition15

Condition16 = 'Condition16'
Condition16

Tup = 'Tup'
Tup

output_channels — Bpod output channels

Overview

Output channels available on Bpod box.

Implementation

class pybpodapi.bpod.hardware.output_channels.**OutputChannel**

Available output channels These values must be set according to Bpod firmware specification.

LED = 'LED'
LED

Valve = 'Valve'
Valve

Serial1 = 'Serial1'
Serial 1

Serial2 = 'Serial2'
Serial 2

Serial3 = 'Serial3'
Serial 3

SoftCode = 'SoftCode'
SoftCode

ValveState = 'ValveState'
ValveState

BNC1 = 'BNC1'
BNC1

BNC2 = 'BNC2'
BNC2

Wire1 = 'Wire1'
Wire1

Wire2 = 'Wire2'
Wire2

Wire3 = 'Wire3'
Wire3

Wire4 = 'Wire4'
Wire3

PWM1 = 'PWM1'
PWM1

PWM2 = 'PWM2'
PWM2

PWM3 = 'PWM3'
PWM3

PWM4 = 'PWM4'
PWM4

PWM5 = 'PWM5'
PWM5

PWM6 = 'PWM6'
PWM6

PWM7 = 'PWM7'
PWM7

PWM8 = 'PWM8'
PWM8

GlobalTimerTrig = 'GlobalTimerTrig'
GlobalTimerTrig

GlobalTimerCancel = 'GlobalTimerCancel'
GlobalTimerCancel

GlobalCounterReset = 'GlobalCounterReset'
GlobalCounterReset

2.8.2 bpod_modules— Bpod Modules

structure to manage the bpod modules

bpod_module— Bpod Module

Implementation

class pybpodapi.bpod_modules.bpod_module.**BpodModule**(*connected=False*, *module_name=''*, *firmware_version=0*, *events_names=[]*, *n_serial_events=0*, *serial_port=None*)

load_message(*msg*, *msg_id=None*)
Load a message through bpod to the module and associate an ID to it.

Parameters

- **msg** (*list (int)*) – Message to send
- **msg_id** (*int*) – Id of the message to use

bpod_modules— Bpod Modules

Implementation

class pybpodapi.bpod_modules.bpod_modules.**BpodModules**(*bpod*)

2.8.3 com— Communication

messaging— Types of messages

BaseMessage

class pybpodapi.com.messaging.base_message.**BaseMessage**(*content*, *host_timestamp=None*)

Represents a session message It may have been originated from the board of from pc

MESSAGE_TYPE_ALIAS = 'MESSAGE'

MESSAGE_COLOR = (200, 200, 200)

classmethod check_type(*typestr*)

Returns True if the typestr represents the class

tolist()

classmethod fromlist(*row*)

Returns True if the typestr represents the class

EndTrial

class pybpodapi.com.messaging.end_trial.**EndTrial**(*content*, *host_timestamp=None*)

Stderr message from the server process

See also:

pybpodgui_plugin.com.messaging.board_message.BoardMessage

MESSAGE_TYPE_ALIAS = 'END-TRIAL'

MESSAGE_COLOR = (0, 100, 200)

EventOccurrence

class pybpodapi.com.messaging.event_occurrence.**EventOccurrence**(*event_id*, *event_name*, *host_timestamp=None*)

Message from board that represents state change (an event)

Variables

- ***event_name*** (*str*) – name of the event
- ***event_id*** (*int*) – index of the event
- **board_timestamp** (*float*) – timestamp associated with this event (from bpod)

Parameters

- **event_id** –
- **event_name** –
- **host_timestamp** –

MESSAGE_TYPE_ALIAS = 'EVENT'

event_name

event_id

tolist()

classmethod fromlist(*row*)
Returns True if the typestr represents the class

EventResume

class pybpodapi.com.messaging.event_resume.**EventResume**(*event_id*, *event_name*, *host_timestamp=None*)
Message from board that represents state change (an event)

Variables

- ***event_name*** (*str*) – name of the event
- ***event_id*** (*int*) – index of the event
- **board_timestamp** (*float*) – timestamp associated with this event (from bpod)

Parameters

- **event_id** –
- **event_name** –
- **host_timestamp** –

MESSAGE_TYPE_ALIAS = 'EVENT-SUMMARY'

classmethod check_type(*typestr*)
Returns True if the typestr represents the class

event_name

event_id

tolist()

classmethod fromlist(*row*)
Returns True if the typestr represents the class

SessionInfo

class pybpodapi.com.messaging.session_info.**SessionInfo**(*infoname*, *infovalue=None*, *start_time=None*, *end_time=None*)
Stderr message from the server process

See also:

pybpodgui_plugin.com.messaging.board_message.BoardMessage

MESSAGE_TYPE_ALIAS = 'INFO'

MESSAGE_COLOR = (150, 150, 255)

tolist()

classmethod fromlist(*row*)
Returns True if the typestr represents the class

infoname

infovalue

SoftcodeOccurrence

class pybpodapi.com.messaging.softcode_occurrence.**SoftcodeOccurrence**(*softcode*, *host_timestamp=None*)

Message from board that represents state change (an event)

Variables

- ***event_name*** (*str*) – name of the event
- ***event_id*** (*int*) – index of the event
- **board_timestamp** (*float*) – timestamp associated with this event (from bpod)

Parameters

- **event_id** –
- **event_name** –
- **host_timestamp** –

MESSAGE_TYPE_ALIAS = 'SOFTCODE'

MESSAGE_COLOR = (40, 30, 30)

softcode

StateOccurrence

class pybpodapi.com.messaging.state_occurrence.**StateOccurrence**(*state_name*, *host_timestamp*, *end_timestamp*)

Store timestamps for a specific state occurrence of the state machine

Variables

- **name** (*str*) – name of the state
- **timestamps** (*list(StateDuration)*) – a list of timestamps (start and end) that corresponds to occurrences of this state

Parameters **name** (*str*) – name of the state

MESSAGE_TYPE_ALIAS = 'STATE'

MESSAGE_COLOR = (0, 100, 0)

tolist()

classmethod fromlist(*row*)

Returns True if the typestr represents the class

state_name

Trial

class pybpodapi.com.messaging.trial.**Trial**(*sma=None*)

Variables

- **trial_start_timestamp** (*float*) – None

- **sma** (*StateMachine*) – sma
- **states_occurrences** (*list(StateOccurrence)*) – list of state occurrences
- **events_occurrences** (*list(EventOccurrence)*) – list of event occurrences

MESSAGE_TYPE_ALIAS = 'TRIAL'

MESSAGE_COLOR = (0, 0, 255)

get_timestamps_by_event_name(*event_name*)
Get timestamps by event name

Parameters **event_name** – name of the event to get timestamps

Return type list(float)

get_events_names()
Get events names without repetitions

Return type list(str)

get_all_timestamps_by_event()
Create a dictionary whose keys are events names and values are corresponding timestamps

Example:

```
{
        'Tup': [429496.7295, 429496.7295],
        'Port3In': [429496.7295, 429496.7295],
        'Port2In': [429496.7295, 429496.7295],
        'Port2Out': [429496.7295, 429496.7295],
        'Port3Out': [429496.7295],
        'Port1Out': [429496.7295]
}
```

Return type dict

export()

pformat()

classmethod fromlist(*row*)
Returns True if the typestr represents the class

UntaggedMessage

class pybpodapi.com.messaging.untagged_message.**UntaggedMessage**(*content*, *host_timestamp=None*)
Stderr message from the server process

See also:

pybpodgui_plugin.com.messaging.board_message.BoardMessage

MESSAGE_TYPE_ALIAS = 'UNTAGGED'

MESSAGE_COLOR = (230, 230, 230)

classmethod check_type(*typestr*)
Returns True if the typestr represents the class

arcom— Receive Arduino Communication Wrapper headers

Contents

Overview

TODO

Implementation

class pybpodapi.com.arcom.**ArCOM**

ArCOM is an interface to simplify data transactions between Arduino and Python.

open(*serial_port*, *baudrate=115200*, *timeout=1*)

Open serial connection :param serialPortName: :param baudRate: :return:

close()

Close serial connection :return:

bytes_available()

Returns

protocol.send_msg_headers— Send message headers

Contents

Overview

TODO

Implementation

class pybpodapi.com.protocol.send_msg_headers.**SendMessageHeader**

Define names for message headers sent to the Bpod device.

The message header is the first byte (character) on a message sent.

HANDSHAKE = '6'

Request initialization handshake

FIRMWARE_VERSION = 'F'
Request firmware build number

RESET_CLOCK = '*'
Reset session clock

PAUSE_TRIAL = '$'
Pause ongoing trial (We recommend using computer-side pauses between trials, to keep data uniform)

GET_TIMESTAMP_TRANSMISSION = 'G'
Return timestamp transmission scheme

HARDWARE_DESCRIPTION = 'H'
Request hardware configuration

ENABLE_PORTS = 'E'
Request enable input ports

SYNC_CHANNEL_MODE = 'K'
Set sync channel and sync mode

NEW_STATE_MATRIX = 'C'
Send new compressed state matrix

RUN_STATE_MACHINE = 'R'
Request to run state matrix now

LOAD_SERIAL_MESSAGE = 'L'
Load serial message

RESET_SERIAL_MESSAGES = '>'
Reset serial messages to equivalent byte codes (i.e. message# 4 = one byte, 0x4)

OVERRIDE_DIGITAL_HW_STATE = 'O'
Override digital hardware state

SEND_TO_HW_SERIAL = 'U'
Send byte to hardware serial channel 1-3

DISCONNECT = 'Z'
Request end of connection now

GET_MODULES = 'M'
Get the modules connected to bpod

SET_MODULE_RELAY = 'J'
Set module relay

WRITE_TO_MODULE = 'T'
Write to the module

ECHO_SOFTCODE = 'S'
Echo soft code

MANUAL_OVERRIDE_EXEC_EVENT = 'V'
Manual override: execute virtual event

TRIGGER_SOFTCODE = '~'
Trigger soft code

EXIT_AND_RETURN = 'X'
Exit state matrix and return data

protocol.recv_msg_headers— Receive message headers

Contents

Overview

TODO

Implementation

class pybpodapi.com.protocol.recv_msg_headers.**ReceiveMessageHeader**

Define names for message headers received from the Bpod device.

The message header is the first byte (character) on a message received.

HANDSHAKE_OK = '5'

Success code from HANDSHAKE command

ENABLE_PORTS_OK = 1

Success code from ENABLE_PORTS command

SYNC_CHANNEL_MODE_OK = 1

Success code from SYNC_CHANNEL_MODE command

STATE_MACHINE_INSTALLATION_STATUS = 1

Success code from RUN_STATE_MACHINE command

LOAD_SERIAL_MESSAGE_OK = 1

Success code from LOAD_SERIAL_MESSAGE command

RESET_SERIAL_MESSAGES = 1

Success code from RESET_SERIAL_MESSAGES command

MODULE_REQUESTED_EVENT = 35

Module requested event

MODULE_EVENT_NAMES = 69

Module events names

2.8.4 exceptions— Bpod exceptions

Implementation

class pybpodapi.exceptions.bpod_error.**BpodErrorException**

2.8.5 state_machine— State Machine

state_machine_base— State Machine Base

Implementation

class pybpodapi.state_machine.state_machine_base.**StateMachineBase**(*bpod*)

Each Bpod trial is programmed as a virtual finite state machine. This ensures precise timing of events - for any state machine you program, state transitions will be completed in less than 250 microseconds - so inefficient coding won't reduce the precision of events in your data.

> **Warning:** A lot of data structures are kept here for compatibility with original matlab library which are not so python-like. Anyone is welcome to enhance this class but keep in mind that it will affect the whole pybpodapi library.

Variables

- ***hardware*** (*Hardware*) – bpod box hardware description associated with this state machine
- ***channels*** (*Channels*) – bpod box channels handling
- **state_names** (*list(str)*) – list that holds state names added to this state machine
- **state_timers** (*list(float)*) – list that holds state timers
- **total_states_added** (*int*) – holds all states added, even if name is repeated
- **state_timer_matrix** (*list(int)*) – TODO:
- ***conditions*** (*Conditions*) – holds conditions
- ***global_counters*** (*GlobalCounters*) – holds global timers
- ***global_timers*** (*GlobalTimers*) – holds global counters
- **input_matrix** (*list(tuple(int))*) – TODO:
- **manifest** (*list(str)*) – list of states names that have been added to the state machine
- **undeclared** (*list(str)*) – list of states names that have been referenced but not yet added
- **meta_output_names** (*tuple(str)*) – TODO:
- **output_matrix** (*list(tuple(int))*) – TODO:
- **is_running** (*bool*) – whether this state machine is being run on bpod box

Parameters **hardware** (*Hardware*) – hardware description associated with this state machine

add_state(*state_name*, *state_timer=0*, *state_change_conditions={}*, *output_actions=()*)

Adds a state to an existing state matrix.

Parameters

- **name** (*str*) – A character string containing the unique name of the state. The state will automatically be assigned a number for internal use and state synchronization via the sync port
- **timer** (*float*) – The state timer value, given in seconds. This value must be zero or positive, and can range between 0-3600s. If set to 0s and linked to a state transition, the state will still take ~100us to execute the state's output actions before the transition completes

- **state_change_conditions** (*dict*) – Dictionary whose keys are names of a valid input event (state change) and values are names of states to enter if the previously listed event occurs (or 'exit' to exit the matrix and return all captured data)
- **output_actions** (*list(tuple)*) – a list of binary tuples where first value should contain the name of a valid output action and the second value should contain the value of the previously listed output action (see output actions for valid values).

Example:

```
sma.add_state(
    state_name='Port1Lit',
    state_timer=.25,
    state_change_conditions={'Tup': 'Port3Lit', 'GlobalTimer1_End': 'exit'},
    output_actions=[('PWM1', 255)])
```

set_global_timer_legacy(*timer_id=None*, *timer_duration=None*)
Set global timer (legacy version)

Parameters

- **timer_ID** (*int*) –
- **timer_duration** (*float*) – timer duration in seconds

set_global_timer(*timer_id*, *timer_duration*, *on_set_delay=0*, *channel=None*, *on_message=1*, *off_message=0*, *loop_mode=0*, *loop_intervals=0*, *send_events=0*, *one-set_triggers=None*)
Sets the duration of a global timer. Unlike state timers, global timers can be triggered from any state (as an output action), and handled from any state (by causing a state change).

Parameters

- **timer_ID** (*int*) – the number of the timer you are setting (an integer, 1-5).
- **timer_duration** (*float*) – the duration of the timer, following timer start (0-3600 seconds)
- **on_set_delay** (*float*) –
- **channel** (*str*) – channel/port name Ex: 'PWM2'
- **on_message** (*int*) –

set_global_counter(*counter_number=None*, *target_event=None*, *threshold=None*)
Sets the threshold and monitored event for one of the 5 global counters. Global counters can count instances of events, and handle when the count exceeds a threshold from any state (by triggering a state change).

Parameters

- **counter_number** (*int*) – the number of the counter you are setting (an integer, 1-5).
- **target_event** (*str*) – port where to listen for event to count
- **threshold** (*int*) – number of times that should be count until trigger timer

set_condition(*condition_number*, *condition_channel*, *channel_value*)
Set condition

Parameters

- **condition_number** (*int*) –
- **condition_channel** (*str*) –

- **channel_value** (*int*) –

exception pybpodapi.state_machine.state_machine_base.**SMAError**

state_machine_builder— State Machine Builder

Implementation

class pybpodapi.state_machine.state_machine_builder.**StateMachineBuilder**(*bpod*)
Extend state machine with builder logic

> **Warning:** A lot of data structures are kept here for compatibility with original matlab library which are not so python-like. Anyone is welcome to enhance this class but keep in mind that it will affect the whole pybpodapi library.

Parameters **hardware** (Hardware) – hardware description associated with this state machine

update_state_numbers()
Replace undeclared states (at the time they were referenced) with actual state numbers

build_message()
Builds state machine to send to Bpod box

Return type list(int)

build_message_32_bits()
Builds a 32 bit message to send to Bpod box

Return type list(float)

exception pybpodapi.state_machine.state_machine_builder.**StateMachineBuilderError**

state_machine_runner— State Machine Runner

Implementation

class pybpodapi.state_machine.state_machine_runner.**StateMachineRunner**(*bpod*)
Extends state machine with running logic

Variables

- **is_running** (*bool*) – Whether this state machine is being run on bpod hardware
- **current_state** (*int*) – Holds state machine current state while running

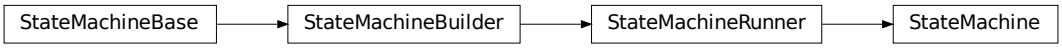
exception pybpodapi.state_machine.state_machine_runner.**StateMachineRunnerError**

Overview

Each Bpod trial is programmed as a virtual finite state machine. This ensures precise timing of events - for any state machine you program, state transitions will be completed in less than 250 microseconds - so inefficient coding won't reduce the precision of events in your data.

For more information, please see https://sites.google.com/site/bpoddocumentation/bpod-user-guide/using-state-matrices .

Inheritance



2.8.6 session— Session

Overview

Everytime a *pybpodapi.bpod* object is created, a new session is instantiated which stores information about the new experiment being run. There is only one session per Bpod. This session contains the list of trials (see pybpodapi.trial.Trial).

Besides storing trials, the session is also responsible for processing pybpodapi.state_occurrences.StateOccurrences and pybpodapi.event_occurrence.EventOccurrence when trial has finished. At this point, the information collected temporarily on the pybpodapi.state_machine.raw_data.RawData object is then persisted on the trial.

Implementation

class pybpodapi.session.**Session**(*path=None*)

Stores information about bpod run, including the list of trials.

Variables

- **trials** (*list(Trial)*) – a list of trials
- **firmware_version** (*int*) – firmware version of Bpod when experiment was run
- **bpod_version** (*int*) – version of Bpod hardware when experiment was run
- **start_timestamp** (*datetime*) – it stores session start timestamp

current_trial

Get current trial

Return type *Trial*

2.9 Diagrams

2.9.1 Class Diagrams (and modules)

Main entities

`pybpodapi.bpod.Bpod`

`pybpodapi.state_machine.StateMachine`

2.9.2 Sequence Diagrams

Start Bpod

Send state machine

Run state machine

2.10 Project Info

2.10.1 The SWP Team

Scientific Software Platform (Champalimaud Foundation)

The Scientific Software Platform (SWP) from the Champalimaud Foundation provides technical know-how in software engineering and high quality software support for the Neuroscience and Cancer research community at the Champalimaud Foundation.

We typical work on computer vision / tracking, behavioral experiments, image registration and database management.

2.10.2 Bpod project

pybpod-api is a python port of the Bpod Matlab project.

All examples and Bpod's state machine and communication logic were based on the original version made available by Josh Sanders (Sanworks).

2.10.3 License

This is Open Source software with a MIT license.

2.10.4 Maintenance team

The current and past members of the **pybpod-api** team.

- @cajomferro Carlos Mão de Ferro
- @JBauto João Baúto
- @UmSenhorQualquer Ricardo Ribeiro
- @sergio-copedo Sérgio Copeto
- @MicBoucinha Luís Teixeira

2.10.5 Questions?

If you have any questions or want to report a problem with this library please fill a issue here.

2.11 Indices and tables

- genindex
- modindex

Python Module Index

b

p

s

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W