SpheroV2 Documentation

Release 0.11.1

Hanbang Wang

Nov 26, 2022

CONTENTS

An unofficial Python library for Sphero toys that supports its Version 2 Bluetooth low energy API described here. Toys that are supported includes (implemented ones are checked):

- [x] Sphero 2.0 / SPRK
- [x] Sphero Ollie
- [x] Sphero BB-8
- [x] Sphero BB-9E
- [x] Sphero R2-D2 / R2-Q5
- [x] Sphero BOLT (In Progress)
- [x] Sphero SPRK+ / SPRK 2.0
- [x] Sphero Mini
- [x] Sphero RVR

Current Progress:

- Found a better way to decompile, fixing a few things like Controls, Command Queueing, and Waiting for responses
- [ ] Controls
  - [ ] Animation Control
  - [ ] Drive Control
  - [x] LED Control
  - [ ] Sensor Control
  - [ ] Stats Control
  - [ ] Streaming Control

The logic is written based on reverse-engineering the official Sphero Edu for Android, with the help from available documentation and other unofficial community-based Sphero libraries like igbopie/spherov2.js and EnotYoyo/pysphero.

This project uses the hbldh/bleak Bluetooth Low Energy library, which works across all platforms.

CHAPTER

ONE

USAGE

To install the library, run `pip install spherov2`. Python version >= `3.7` are supported.

The library currently has two adapters, `BleakAdapter` and `TCPAdapter`. `BleakAdapter` is used by default when adapter is not specified, which connects to toys using the local Bluetooth adapter. For example:

```
from spherov2 import scanner

with scanner.find_toy() as toy:
    ...
```

`TCPAdapter` allows the user to send and receive Bluetooth packets connected to another host via a server running on that host as a relay. To start the server, run `python -m spherov2.adapter.tcp_server [host] [port]`, with `host` and `port` by default being `0.0.0.0` and `50004`. To use the adapter, for example:

```
from spherov2 import scanner
from spherov2.adapter.tcp_adapter import get_tcp_adapter

with scanner.find_toy(adapter=get_tcp_adapter('localhost')) as toy:
    ...
```

The TCP server is written in asynchronous fashion using `asyncio`, so that it supports `bleak` on all platforms.

On whichever device you decide to connect to the toys, you have to first install the BLE library by `pip install bleak`.

1.1 Scanner

You can scan the toys around you using the scanner helper. To find all possible toys, simply call `scanner.find_toys()`. To find only a single toy, use `scanner.find_toy()`.

You can also find toys using specific filters. Please refer to the document for more information.

1.2 APIs

There are two ways you can interact with the toys, one is to use the low-level APIs implemented for each toy with the commands they support. Low-level APIs can be found for each toy under `spherov2.toy.*`, and is not documented.

The other and recommended way is to use the high level API `spherov2.sphero_edu.SpheroEduAPI`, which is an implementation of the official Sphero Edu APIs. Documentations can be found inside the source files with the docstrings, or here as an HTML rendered version. For example:

```
from spherov2 import scanner
from spherov2.sphero_edu import SpheroEduAPI

toy = scanner.find_toy()
with SpheroEduAPI(toy) as api:
    api.spin(360, 1)
```

CHAPTER

TWO

ACKNOWLEDGMENTS

This library is made for educational purposes. It is used by students in CIS 521 - Artificial Intelligence at the University of Pennsylvania, where we use Sphero robots to help teach the foundations of AI.

It is published as an open-source library under the MIT License.

CHAPTER

THREE

AUTHORS

- **Hanbang Wang** - https://www.cis.upenn.edu/~hanbangw/
- **Elionardo Feliciano**

3.1 Scanner

spherov2.scanner.**find_toys**(*, *timeout=5.0*, *toy_types: Optional[Iterable[Type[Toy]]] = None*, *toy_names: Optional[Iterable[str]] = None*, *adapter=None*) → List[Toy]

Find toys that matches the criteria given.

Parameters

- **timeout** – Device scanning timeout, in seconds.
- **toy_types** – Iterable of toy types (subclasses of `Toy`) that needs to be scanned. Set to `None` to scan all toy types available.
- **toy_names** – Iterable of strings of toy names that needs to be scanned. Set to `None` to scan toys with all kinds of names.
- **adapter** – Kind of adapter to use for scanning bluetooth devices. Set to `None` to use default `BleakAdapter`.

Returns

A list of toys that are scanned.

spherov2.scanner.**find_toy**(*, *toy_name: Optional[str] = None*, ***kwargs*) → Toy

Find a single toy that matches the criteria given.

Parameters

- **toy_name** – A string of toy name that needs to be scanned. Set to `None` to scan toy with all kinds of names.
- **timeout** – Device scanning timeout, in seconds.
- **toy_types** – List of toy types (subclasses of `Toy`) that needs to be scanned. Set to `None` to scan all toy types available.
- **adapter** – Kind of adapter to use for scanning bluetooth devices. Set to `None` to use default `BleakAdapter`.

Returns

A toy that is scanned.

Raises
ToyNotFoundError – If no toys could be found

spherov2.scanner.**find_R2D2**(*toy_name: str = None*, ***kwargs*)
Same as `find_toy(toy_types=[R2D2], toy_name, **kwargs)`.

spherov2.scanner.**find_R2Q5**(*toy_name: str = None*, ***kwargs*)
Same as `find_toy(toy_types=[R2Q5], toy_name, **kwargs)`.

3.2 Adapter

This library abstracts the Bluetooth communication layer as a single adapter class, to potentially support all platforms and all environments.

3.2.1 BleakAdapter

class spherov2.adapter.bleak_adapter.**BleakAdapter**

This adapter uses hbldh/bleak library directly on the host machine. Make sure there is a low energy Bluetooth adapter (Bluetooth 4.0 or above) on your computer. Then, install the Bluetooth library by `pip install bleak`.

`BleakAdapter` is used by default by the scanner:

```
from spherov2 import scanner

with scanner.find_toy() as toy:
    ...
```

3.2.2 TCPAdapter

class spherov2.adapter.tcp_adapter.**TCPAdapter**

This adapter allows the user to send and receive Bluetooth packets connected to another host via a server running on that host as a relay.

To start the server, run `python -m spherov2.adapter.tcp_server [host] [port]`, with `host` and `port` are optional and by default being `0.0.0.0` and `50004`.

spherov2.adapter.tcp_adapter.**get_tcp_adapter**(*host: str*, *port: int = 50004*)
Gets an anonymous `TCPAdapter` with the given address and port.

To use the adapter, for example:

```
from spherov2 import scanner
from spherov2.adapter.tcp_adapter import get_tcp_adapter

with scanner.find_toy(adapter=get_tcp_adapter('localhost')) as toy:
    ...
```

3.3 Sphero Edu API

class spherov2.sphero_edu.**SpheroEduAPI**(*toy: Toy*)

Implementation of Sphero Edu Javascript APIs: https://sphero.docsapp.io/docs/get-started

3.3.1 Get Started

Sphero robots aren't just toys – they are programmable robots that are great for learning about computer science! This wiki is a guide to learn how to program Sphero robots with Python. You will need a Sphero robot, a Python 3 environment, and a hunger to learn.

Hello World!

Using your device with the Sphero Edu API, create a new python code file, and copy and paste these code samples into the file. Don't forget to aim your robot, and then run the script to see what happens!

```
import time
from spherov2 import scanner
from spherov2.sphero_edu import SpheroEduAPI
from spherov2.types import Color

toy = scanner.find_toy()
with SpheroEduAPI(toy) as droid:
    droid.set_main_led(Color(r=0, g=0, b=255))
    droid.set_speed(60)
    time.sleep(2)
    droid.set_speed(0)
```

3.3.2 Movement

Movements control the robot's motors and control system. You can use sequential movement commands by separating them with line breaks, like the *Hello World!* program. Sphero robots move with three basic instructions: heading, speed, and duration. For example, if you set heading = 0°, speed = 60, duration = 3s, the robot would roll forward for 3s at a moderate speed.

class spherov2.sphero_edu.**SpheroEduAPI**

roll(*heading: int*, *speed: int*, *duration: float*)

Combines heading(0-360°), speed(-255-255), and duration to make the robot roll with one line of code. For example, to have the robot roll at 90°, at speed 200 for 2s, use `roll(90, 200, 2)`

set_speed(*speed: int*)

Sets the speed of the robot from -255 to 255, where positive speed is forward, negative speed is backward, and 0 is stopped. Each robot type translates this value differently into a real world speed; Ollie is almost three times faster than Sphero. For example, use `set_speed(188)` to set the speed to 188 which persists until you set a different speed. You can also read the real-time velocity value in centimeters per second reported by the motor encoders.

stop_roll(*heading: Optional[int] = None*)

Sets the speed to zero to stop the robot, effectively the same as the `set_speed(0)` command.

set_heading(*heading: int*)

Sets the direction the robot rolls. Assuming you aim the robot with the blue tail light facing you, then 0° is forward, 90° is right, 270° is left, and 180° is backward. For example, use `set_heading(90)` to face right.

spin(*angle: int*, *duration: float*)

Spins the robot for a given number of degrees over time, with 360° being a single revolution. For example, to spin the robot 360° over 1s, use: `spin(360, 1)`. Use *set_speed()* prior to *spin()* to have the robot move in circle or an arc or circle.

Note: Unlike official API, performance of spin is guaranteed, but may be longer than the specified duration.

set_stabilization(*stabilize: bool*)

Turns the stabilization system on and `set_stabilization(false)` turns it off. Stabilization is normally on to keep the robot upright using the Inertial Measurement Unit (IMU), a combination of readings from the Accelerometer (directional acceleration), Gyroscope (rotation speed), and Encoders (location and distance). When `set_stabilization(false)` and you power the motors, the robot will not balance, resulting in possible unstable behaviors like wobbly driving, or even jumping if you set the power very high. Some use cases to turn it off are:

1. Jumping: Set Motor Power to max values and the robot will jump off the ground!
2. Gyro: Programs like the Spinning Top where you want to to isolate the Gyroscope readings rather than having the robot auto balance inside the shell.

When stabilization is off you can't use *set_speed()* to set a speed because it requires the control system to be on to function. However, you can control the motors using Motor Power with *raw_motor()* when the control system is off.

raw_motor(*left: int*, *right: int*, *duration: float*)

Controls the electrical power sent to the left and right motors independently, on a scale from -255 to 255 where positive is forward, negative is backward, and 0 is stopped. If you set both motors to full power the robot will jump because stabilization (use of the IMU to keep the robot upright) is disabled when using this command. This is different from *set_speed()* because Raw Motor sends an "Electromotive force" to the motors, whereas Set Speed is a target speed measured by the encoders. For example, to set the raw motor to full power for 4s, making the robot jump off the ground, use `raw_motor(255, 255, 4)`.

reset_aim()

Resets the heading calibration (aim) angle to use the current direction of the robot as 0°.

Sphero BOLT Movements

Sphero BOLT has a compass (magnetometer) sensor that has unique functionality. Nearby metallic and magnetic objects can affect the accuracy of the compass, so try to use this feature in an area without that interference, or hold it up in the air if you can't get away from interference.

Star Wars Droid Movements

class spherov2.sphero_edu.**SpheroEduAPI**

play_animation(*animation: IntEnum*)

Plays iconic Star Wars Droid animations unique to BB-8, BB-9E, R2-D2 and R2-Q5 that combine movement, lights and sound. All animation enums can be accessed under the droid class, such as `R2D2.Animations.CHARGER_1`.

R2-D2 & R2-Q5 Movements

The R2-D2 and R2-Q5 Droids are physically different from other Sphero robots, so there are some unique commands that only they can use.

class spherov2.sphero_edu.**SpheroEduAPI**

set_dome_position(*angle: float*)

Rotates the dome on its axis, from -160° to 180°. For example, set to 45° using `set_dome_position(45)`.

set_stance(*stance: Stance*)

Changes the stance between bipod and tripod. Set to bipod using `set_stance(Stance.Bipod)` and to tripod using `set_stance(Stance.Tripod)`. Tripod is required for rolling.

set_waddle(*waddle: bool*)

Turns the waddle walk on using *set_waddle(True)* ` and off using `set_waddle(False)`.

3.3.3 Lights

Lights control the color and brightness of LEDs on a robot.

class spherov2.sphero_edu.**SpheroEduAPI**

set_main_led(*color: Color*)

Changes the color of the main LED light, or the full matrix on Sphero BOLT. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, `set_main_led(Color(r=90, g=255, b=90))`.

set_back_led(*color: int*)

Sets the brightness of the back aiming LED, aka the "Tail Light." This LED is limited to blue only, with a brightness scale from 0 to 255. For example, use `set_back_led(255)` to set the back LED to full brightness. Use `time.sleep()` to set it on for a duration. For example, to create a dim and a bright blink sequence use:

```
set_back_led(0)  # Dim
delay(0.33)
set_back_led(255)  # Bright
delay(0.33)
```

fade(*from_color: Color*, *to_color: Color*, *duration: float*)

Changes the main LED lights from one color to another over a period of seconds. For example, to fade from green to blue over 3s, use: `fade(Color(0, 255, 0), Color(0, 0, 255), 3.0)`.

strobe(*color: Color*, *period: float*, *count: int*)

Repeatedly blinks the main LED lights. The period is the time, in seconds, the light stays on during a single blink; cycles is the total number of blinks. The time for a single cycle is twice the period (time for a blink plus the same amount of time for the light to be off). Another way to say this is the period is 1/2 the time it takes for a single cycle. So, to strobe red 15 times in 3 seconds, use: `strobe(Color(255, 57, 66), (3 / 15) * .5, 15)`.

Sphero BOLT Lights

Sphero BOLT has unique lighting capabilities with a front led, back led, and 8x8 led matrix. The matrix has 3 methods to program in increasing abstraction and sophistication that are very fun to play with! The 3 methods are setting pixels, text, and animations.

class spherov2.sphero_edu.**SpheroEduAPI**

set_front_led(*color: spherov2.types.Color*)

Changes the color of the front LED light. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the magenta color is expressed as `set_front_color(Color(239, 0, 255))`

set_back_led(*color: spherov2.types.Color*)

Changes the color of the back LED light, aka the "Tail Light" or "Aim Light." Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the green color is expressed as `set_back_led(Color(0, 255, 0))`. Sphero BOLT has this as an RGB LED on the back, whereas previous Spheros are limited to *blue only*.

Sphero RVR Lights

class spherov2.sphero_edu.**SpheroEduAPI**

set_left_headlight_led(*color: Color*)

Changes the color of the front left headlight LED on RVR. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the pink color is expressed as `set_left_headlight_led(Color(253, 159, 255))`.

set_right_headlight_led(*color: Color*)

Changes the color of the front right headlight LED on RVR. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the blue color is expressed as `set_right_headlight_led(0, 28, 255)`.

set_left_led(*color: Color*)

Changes the color of the LED on RVR's left side (which is the side with RVR's battery bay door). Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the green color is expressed as `set_left_led(Color(0, 255, 34))`.

set_right_led(*color: Color*)

Changes the color of the LED on RVR's right side (which is the side with RVR's power button). Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the red color is expressed as `set_right_led(Color(255, 18, 0))`.

set_front_led(*color: spherov2.types.Color*)

Changes the color of RVR's front two LED headlights together. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the magenta color is expressed as `set_front_color(Color(239, 0, 255))`.

set_back_led(*color: spherov2.types.Color*)

Changes the color of the back LED light, aka the "Tail Light" or "Aim Light." Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the above green color is expressed as `set_back_led(Color(0, 255, 0))`.

BB-9E Lights

class spherov2.sphero_edu.**SpheroEduAPI**

set_dome_leds(*brightness: int*)

Controls the brightness of the two single color LEDs (red and blue) in the dome, from 0 to 15. We don't use 0-255 for this light because it has less granular control. For example, set them to full brightness using `set_dome_leds(15)`.

R2-D2 & R2-Q5 Lights

class spherov2.sphero_edu.**SpheroEduAPI**

set_front_led(*color: spherov2.types.Color*)

Changes the color of the front LED light. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the fuchsia color is expressed as `set_front_color(Color(232, 0, 255))`

set_back_led(*color: spherov2.types.Color*)

Changes the color of the back LED light. Set this using RGB (red, green, blue) values on a scale of 0 - 255. For example, the above green color is expressed as `set_back_led(Color(0, 255, 0))`.

set_holo_projector_led(*brightness: int*)

Changes the brightness of the Holographic Projector white LED, from 0 to 255. For example, set it to full brightness using `set_holo_projector_led(255)`.

set_logic_display_leds(*brightness: int*)

Changes the brightness of the Logic Display LEDs, from 0 to 255. For example, set it to full brightness using `set_logic_display_leds(255)`.

3.3.4 Sounds

Control sounds and words which can play from your programming device's speaker or the robot (R2-D2 & R2-Q5 only).

class spherov2.sphero_edu.**SpheroEduAPI**

play_sound(*sound: IntEnum*)

Unique Star Wars Droid Sounds are available for BB-8, BB-9E and R2-D2. For example, to play the R2-D2 Burnout sound use `play_sound(R2D2.Audio.R2_BURNOUT)`.

3.3.5 Sensors

Querying sensor data allows you to react to real-time values coming from the robots' physical sensors. For example, "if accelerometer z-axis > 3G's, then set LED's to green."

class spherov2.sphero_edu.**SpheroEduAPI**

get_acceleration()

Provides motion acceleration data along a given axis measured by the Accelerometer, in g's, where g = 9.80665 m/s^2.

`get_acceleration()['x']` is the left-to-right acceleration, from -8 to 8 g's.

`get_acceleration()['y']` is the forward-to-back acceleration, from of -8 to 8 g's.

`get_acceleration()['z']` is the upward-to-downward acceleration, from -8 to 8 g's.

get_vertical_acceleration()

This is the upward or downward acceleration regardless of the robot's orientation, from -8 to 8 g's.

get_orientation()

Provides the tilt angle along a given axis measured by the Gyroscope, in degrees.

`get_orientation()['pitch']` is the forward or backward tilt angle, from -180° to 180°.

`get_orientation()['roll']` is left or right tilt angle, from -90° to 90°.

`get_orientation()['yaw']` is the spin (twist) angle, from -180° to 180°.

get_gyroscope()

Provides the rate of rotation around a given axis measured by the gyroscope, from -2,000° to 2,000° per second.

`get_gyroscope().['pitch']` is the rate of forward or backward spin, from -2,000° to 2,000° per second.

`get_gyroscope().['roll']` is the rate of left or right spin, from -2,000° to 2,000° per second.

`get_gyroscope().['yaw']` is the rate of sideways spin, from -2,000° to 2,000° per second.

get_velocity()

Provides the velocity along a given axis measured by the motor encoders, in centimeters per second.

`get_velocity()['x']` is the right (+) or left (-) velocity, in centimeters per second.

`get_velocity()['y']` is the forward (+) or back (-) velocity, in centimeters per second.

get_location()

Provides the location where the robot is in space (x,y) relative to the origin, in centimeters. This is not the distance traveled during the program, it is the offset from the origin (program start).

`get_location()['x']` is the right (+) or left (-) distance from the origin of the program start, in centimeters.

`get_location()['y']` is the forward (+) or backward (-) distance from the origin of the program start, in centimeters.

get_distance()

Provides the total distance traveled in the program, in centimeters.

get_speed()

Provides the current target speed of the robot, from -255 to 255, where positive is forward, negative is backward, and 0 is stopped.

get_heading()

Provides the target directional angle, in degrees. Assuming you aim the robot with the tail facing you, then 0° heading is forward, 90° is right, 180° is backward, and 270° is left.

get_main_led()

Provides the RGB color of the main LEDs, from 0 to 255 for each color channel.

`get_main_led().r` is the red channel, from 0 - 255.

`get_main_led().g` is the green channel, from 0 - 255.

`get_main_led().b` is the blue channel, from 0 - 255.

get_back_led()

`get_back_led().b` is the brightness of the back LED, from 0 to 255. For Sphero BOLT, use `get_back_led()` to get the RGB value.

Sphero BOLT Sensors

class spherov2.sphero_edu.**SpheroEduAPI**

get_last_ir_message()

Returns which channel the last infrared message was received on. You need to declare the `on_ir_message` event for each IR message you plan to see returned.

get_back_led()

Provides the RGB color of the back LED, from 0 to 255 for each color channel.

get_front_led()

Provides the RGB color of the front LED, from 0 to 255 for each color channel.

Sphero RVR Sensors

class spherov2.sphero_edu.**SpheroEduAPI**

get_color()

Provides the RGB color, from 0 to 255 for each color channel, that is returned from RVR's color sensor.

`get_color().r` is the red channel, from 0 - 255, that is returned from RVR's color sensor.

`get_color().g` is the green channel, from 0 - 255, that is returned from RVR's color sensor.

`get_color().b` is the blue channel, from 0 - 255, that is returned from RVR's color sensor.

R2-D2 & R2-Q5 Sensors

class spherov2.sphero_edu.**SpheroEduAPI**

get_back_led()

Provides the RGB color of the back LED, from 0 to 255 for each color channel.

get_front_led()

Provides the RGB color of the front LED, from 0 to 255 for each color channel.

get_dome_leds()

Provides the brightness of the Dome LEDs, from 0 to 15.

get_holo_projector_led()

Provides the brightness of the Holographic Projector LED, from 0 to 255.

get_logic_display_leds()

Provides the brightness of the white Logic Display LEDs, from 0 to 255.

3.3.6 Communications

Infrared (IR) is invisible light with longer wavelengths than visible light, and it is commonly used in TV remote controls to transmit small amounts of data. IR is used in Sphero BOLT to transmit data such as relative distance and heading between robots to enable following and evading behavior among multiple robots, as well as to send custom messages. There are four IR emitters and receivers (pairs) for 360° awareness assuming there is a clear line of sight between two or more robots. The effective range is up to about 3 meters.

class spherov2.sphero_edu.**SpheroEduAPI**

start_ir_broadcast(*near: int*, *far: int*)

Sets the IR emitters to broadcast on two specified channels, from 0 to 7, so other BOLTs can follow or evade. The broadcaster uses two channels because the first channel emits near IR pulses (< 1 meter), and the second channel emits far IR pulses (1 to 3 meters) so the following and evading BOLTs can detect these messages on their IR receivers with a sense of relative proximity to the broadcaster. You can't use a channel for more than one purpose at time, such as sending messages along with broadcasting, following, or evading. For example, use `start_ir_broadcast(0, 1)` to broadcast on channels 0 and 1, so that other BOLTs following or evading on 0 and 1 will recognize this robot.

stop_ir_broadcast()

Stops the broadcasting behavior.

start_ir_follow(*near: int*, *far: int*)

Sets the IR receivers to look for broadcasting BOLTs on the same channel pair, from 0 to 7. Upon receiving messages from a broadcasting BOLT, the follower will adjust its heading and speed to follow the broadcaster. When a follower loses sight of a broadcaster, the follower will spin in place to search for the broadcaster. You can't use a channel for more than one purpose at time, such as sending messages along with broadcasting, following, or evading. For example, use `start_ir_follow(0, 1)` to follow another BOLT that is broadcasting on channels 0 and 1.

stop_ir_follow()

Stops the following behavior.

start_ir_evade(*near: int*, *far: int*)

Sets the IR receivers to look for broadcasting BOLTs on the same channel pair, from 0 to 7. Upon receiving messages from a broadcasting BOLT, the evader will adjust its heading to roll away from the broadcaster. When an evader loses sight of a broadcaster, the evader will spin in place to search for the broadcaster. The evader may stop if it is in the far range for a period of time so it does not roll too far away from the broadcaster. You can't use a channel for more than one purpose at time, such as sending messages along with broadcasting, following, or evading. For example, use `start_ir_evade(0, 1)` to evade another BOLT that is broadcasting on channels 0 and 1.

stop_ir_evade()

Stops the evading behavior.

send_ir_message(*channel: int*, *intensity: int*)

Sends a message on a given IR channel, at a set intensity, from 1 to 64. Intensity is proportional to proximity, where a 1 is the closest, and 64 is the farthest. For example, use `send_ir_message(4, 5)` to send message 4 at intensity 5. You will need to use `onIRMessage4(channel)` event for on a corresponding robot to

receive the message. Also see the `getLastIRMessage()` sensor to keep track of the last message your robot received. You can't use a channel for more than one purpose at time, such as sending messages along with broadcasting, following, or evading.

listen_for_ir_message()

Refer to *IR Message Received Event* for usage.

listen_for_color_sensor()

Refer to *Color Event* for usage.

3.3.7 Events

Events are predefined robot functions into which you can embed conditional logic. When an event occurs, the conditional logic is called in a newly spawned thread. The event will be called every time it occurs by default, unless you customize it. For example, "on collision, change LED lights to red and play the Collision sound," while the main loop is still running.

class spherov2.sphero_edu.**SpheroEduAPI**

register_event(*event_type: EventType*, *listener: Callable[[...], None]*)

Registers the event type with listener. If listener is `None` then it removes all listeners of the specified event type.

Note: listeners will be called in a newly spawned thread, meaning the caller have to deal with concurrency if needed. This library is thread-safe.

On Collision

Executes conditional logic when the robot collides with an object.

```
def on_collision(api):
    # code to execute on collision

api.register_event(EventType.on_collision, on_collision)
```

For example, below is a basic pong program where Sphero bounces off walls, or your hands/feet in perpetuity. Place the robot on the floor between two parallel walls/objects and run the program with the robot pointed perpendicularly at one wall. On collision, the program will print "collision" and change the LED's to red, then continue in the opposite direction:

```
def on_collision(api):
    api.stop_roll()
    api.set_main_led(Color(255, 0, 0))
    print('Collision')
    api.set_heading(api.get_heading() + 180)
    time.sleep(0.5)
    api.set_main_led(Color(255, 22, 255))
    api.set_speed(100)

api.register_event(EventType.on_collision, on_collision)
api.set_main_led(Color(255, 255, 255))
api.set_speed(100)
```

On Freefall

Executes conditional logic when gravity is the only force acting on the robot, such as when dropping or throwing it. Freefall is measured by an accelerometer reading of < 0.1g for => 0.1s, where 1g is resting. On earth, objects in freefall accelerate downwards at 9.81 m/s^2. If you are in orbit, objects appear to be at rest with a reading of 0g because they (and you) are always in freefall, but they never hit the Earth.

```
def on_freefall(api):
    # code to execute on freefall

api.register_event(EventType.on_freefall, on_freefall)
```

For example, to print "freefall" and change the LED's to red on freefall use:

```
def on_freefall(api):
    api.set_main_led(Color(255, 0, 0))
    print('freefall')

api.register_event(EventType.on_freefall, on_freefall)
api.set_main_led(Color(255, 255, 255))
```

On Landing

Executes conditional logic when the robot lands after an being in freefall. You don't need to define an `on_freefall` event for the robot to experience an `on_landing`, but the robot must meet the conditions for freefall before land.

```
def on_landing(api):
    # code to execute on landing

api.register_event(EventType.on_landing, on_landing)
```

For example, to print "landing" and change the LED's to green after landing use:

```
def on_landing(api):
    api.set_main_led(Color(0, 255, 0))
    print('land')

api.register_event(EventType.on_landing, on_landing)
api.set_main_led(Color(255, 255, 255))
```

On Gyro Max

Executes conditional logic when the robot exceeds the bounds of measurable rotational velocity of -2,000° - 2,000° per second. This can be triggered by spinning the robot around like a top on a table really fast. You need to spin it around > 5.5 revolutions per second.

```
def on_gyro_max(api):
    # code to execute on gyromax

api.register_event(EventType.on_gyro_max, on_gyro_max)
```

For example, to print "gyromax" and change the LED's to red when you reach gyromax, use:

```
def on_gyro_max(api):
    api.set_main_led(Color(255, 0, 0))
    print('gyromax')

api.register_event(EventType.on_gyro_max, on_gyro_max)
api.set_stabilization(False)
api.set_back_led(255)
api.set_main_led(Color(255, 255, 255))
```

On Charging

Executes conditional logic called when the robot starts charging its battery. This can be triggered by placing your robot in it's charging cradle, or by plugging it in.

```
def on_charging(api):
    # code to execute on charging

api.register_event(EventType.on_charging, on_charging)
```

On Not Charging

Executes conditional logic called when the robot stops charging its battery. This can be triggered by removing your robot from it's charging cradle, or unplugging it.

```
def on_not_charging(api):
    # code to execute on not charging

api.register_event(EventType.on_not_charging, on_not_charging)
```

For example, to have Sphero execute 2 different conditions for on charging, and on not charging, use the below.

```
def on_charging(api):
    api.set_main_led(Color(6, 0, 255))
    print('charging')
    time.sleep(1)
    print('remove me from my charger')

api.register_event(EventType.on_charging, on_charging)

def on_not_charging(api):
    api.set_main_led(Color(255, 0, 47))
    print('not charging')

api.register_event(EventType.on_not_charging, on_not_charging)

print('place me in my charger')
while True:
    api.set_main_led(Color(3, 255, 0))
    time.sleep(0.5)
```

On IR Message Received

Executes conditional logic called when an infrared message is received on the specified channel. This can be triggered by one Sphero BOLT robot receiving a message from another Sphero BOLT. For example, to have Sphero BOLT change the matrix to red when receiving a message on channel 4:

```
message_channels = (4, )

def on_ir_message_4(api, channel):
    if channel != 4:
        return
    api.set_main_led(Color(255, 0, 0))
    api.listen_for_ir_message(message_channels)

api.register_event(EventType.on_ir_message, on_ir_message_4)
api.listen_for_ir_message(message_channels)
```

On Color

Executes conditional logic called when Sphero RVR's color sensor returns a specified RGB color value.

```
color = (Color(255, 15, 60), )

def on_color(api, color):
    if color != Color(255, 15, 60):
        return

api.register_event(EventType.on_color, on_color)
api.listen_for_color_sensor(colors)
```

The color that RVR's color sensor returns needs to be very close to the color set with `listen_for_color_sensor()` for the event to execute correctly.

3.4 Toys

3.4.1 Sphero 2.0 / SPRK

class spherov2.toy.sphero.**Sphero**(*toy*, *adapter_cls*)

Bases: Toy

toy_type = ToyType(display_name='SPRK/2.0', prefix=None, filter_prefix='Sphero', cmd_safe_interval=0.06)

sensors = {'accelerometer': {'x': ToySensor(bit=32768, min_value=-32768.0, max_value=32767.0, modifier=<function Sphero.<lambda>>), 'y': ToySensor(bit=16384, min_value=-32768.0, max_value=32767.0, modifier=<function Sphero.<lambda>>), 'z': ToySensor(bit=8192, min_value=-32768.0, max_value=32767.0, modifier=<function Sphero.<lambda>>)}, 'attitude': {'pitch': ToySensor(bit=262144, min_value=-179.0, max_value=180.0, modifier=None), 'roll': ToySensor(bit=131072, min_value=-179.0, max_value=180.0, modifier=None), 'yaw': ToySensor(bit=65536, min_value=-179.0, max_value=180.0, modifier=None)}, 'back_emf': {'left': ToySensor(bit=64, min_value=-32768.0, max_value=32767.0, modifier=None), 'right': ToySensor(bit=32, min_value=-32768.0, max_value=32767.0, modifier=None)}, 'gyroscope': {'x': ToySensor(bit=4096, min_value=-20000.0, max_value=20000.0, modifier=<function Sphero.<lambda>>), 'y': ToySensor(bit=2048, min_value=-20000.0, max_value=20000.0, modifier=<function Sphero.<lambda>>), 'z': ToySensor(bit=1024, min_value=-20000.0, max_value=20000.0, modifier=<function Sphero.<lambda>>)}}

extended_sensors = {'accel_one': {'accel_one': ToySensor(bit=33554432, min_value=0.0, max_value=8000.0, modifier=None)}, 'locator': {'x': ToySensor(bit=134217728, min_value=-32768.0, max_value=32767.0, modifier=None), 'y': ToySensor(bit=67108864, min_value=-32768.0, max_value=32767.0, modifier=None)}, 'quaternion': {'w': ToySensor(bit=268435456, min_value=-10000.0, max_value=10000.0, modifier=<function Sphero.<lambda>>), 'x': ToySensor(bit=2147483648, min_value=-10000.0, max_value=10000.0, modifier=<function Sphero.<lambda>>), 'y': ToySensor(bit=1073741824, min_value=-10000.0, max_value=10000.0, modifier=<function Sphero.<lambda>>), 'z': ToySensor(bit=536870912, min_value=-10000.0, max_value=10000.0, modifier=<function Sphero.<lambda>>)}, 'speed': {'speed': ToySensor(bit=4194304, min_value=0.0, max_value=32767.0, modifier=None)}, 'velocity': {'x': ToySensor(bit=16777216, min_value=-32768.0, max_value=32767.0, modifier=<function Sphero.<lambda>>), 'y': ToySensor(bit=8388608, min_value=-32768.0, max_value=32767.0, modifier=<function Sphero.<lambda>>)}}

wake()

add_battery_state_changed_notify_listener(*listener: Callable*)

remove_battery_state_changed_notify_listener(*listener: Callable*)

add_collision_detected_notify_listener(*listener: Callable*)

remove_collision_detected_notify_listener(*listener: Callable*)

add_did_sleep_notify_listener(*listener: Callable*)

remove_did_sleep_notify_listener(*listener: Callable*)

add_gyro_max_notify_listener(*listener: Callable*)

remove_gyro_max_notify_listener(*listener: Callable*)

add_sensor_streaming_data_notify_listener(*listener: Callable*)

remove_sensor_streaming_data_notify_listener(*listener: Callable*)

add_will_sleep_notify_listener(*listener: Callable*)

remove_will_sleep_notify_listener(*listener: Callable*)

begin_reflash(*proc=None*)

here_is_page(*proc=None*)

jump_to_main(*proc=None*)

enable_battery_state_changed_notify(*enable: bool*, *proc=None*)

get_bluetooth_info(*proc=None*)

get_charger_state(*proc=None*)

get_power_state(*proc=None*)

get_versions(*proc=None*)

jump_to_bootloader(*proc=None*)

ping(*proc=None*)

set_bluetooth_name(*name: bytes*, *proc=None*)

set_inactivity_timeout(*timeout: int*, *proc=None*)

sleep(*interval_option: IntervalOptions*, *unk: int*, *unk2: int*, *proc=None*)

boost(*s*, *s2*, *proc=None*)

configure_collision_detection(*collision_detection_method: CollisionDetectionMethods*, *x_threshold*, *y_threshold*, *x_speed*, *y_speed*, *dead_time*, *proc=None*)

configure_locator(*flags*, *x*, *y*, *yaw_tare*, *proc=None*)

get_chassis_id(*proc=None*)

get_persistent_options(*proc=None*)

get_temperature(*proc=None*)

set_temporary_options(*options: Options*, *proc=None*)

roll(*speed*, *heading*, *roll_mode: RollModes*, *reverse_flag: ReverseFlags*, *proc=None*)

self_level(*opt1: bool*, *opt2: bool*, *opt3: bool*, *opt4: bool*, *angle_limit*, *timeout*, *true_time*, *proc=None*)

set_back_led_brightness(*brightness*, *proc=None*)

set_data_streaming(*interval*, *num_samples_per_packet*, *mask*, *count*, *extended_mask*, *proc=None*)

set_heading(*heading: int*, *proc=None*)

set_main_led(*r*, *g*, *b*, *proc=None*)

set_motion_timeout(*timeout: int*, *proc=None*)

set_persistent_options(*options: Options*, *proc=None*)

set_raw_motors(*left_mode: RawMotorModes*, *left_speed*, *right_mode: RawMotorModes*, *right_speed*, *proc=None*)

set_rotation_rate(*rate: int*, *proc=None*)

set_stabilization(*stabilize: bool*, *proc=None*)

property drive_control

property sensor_control

property stats_control

property firmware_update_control

3.4.2 Ollie

class spherov2.toy.ollie.**Ollie**(*toy*, *adapter_cls*)

Bases: *Sphero*

toy_type = ToyType(display_name='Ollie', prefix='2B-', filter_prefix='2B', cmd_safe_interval=0.06)

get_sku(*proc=None*)

3.4.3 BB-8

class spherov2.toy.bb8.**BB8**(*toy*, *adapter_cls*)

Bases: *Ollie*

toy_type = ToyType(display_name='BB-8', prefix='BB-', filter_prefix='BB', cmd_safe_interval=0.06)

get_factory_config_block_crc(*proc=None*)

3.4.4 BB-9E

class spherov2.toy.bb9e.**BB9E**(*toy*, *adapter_cls*)

Bases: ToyV2

toy_type = ToyType(display_name='BB-9E', prefix='GB-', filter_prefix='GB', cmd_safe_interval=0.12)

class LEDs(*value*)

Bases: IntEnum

An enumeration.

BODY_RED = 0

BODY_GREEN = 1

BODY_BLUE = 2

AIMING = 3

HEAD = 4

class Animations(*value*)

Bases: IntEnum

An enumeration.

EMOTE_ALARM = 0

EMOTE_NO = 1

EMOTE_SCAN_SWEEP = 2

EMOTE_SCARED = 3

EMOTE_YES = 4

EMOTE_AFFIRMATIVE = 5

EMOTE_AGITATED = 6

EMOTE_ANGRY = 7

EMOTE_CONTENT = 8

EMOTE_EXCITED = 9

EMOTE_FIERY = 10

EMOTE_GREETINGS = 11

EMOTE_NERVOUS = 12

EMOTE_SLEEP = 14

EMOTE_SURPRISED = 15

EMOTE_UNDERSTOOD = 16

HIT = 17

WWM_ANGRY = 18

WWM_ANXIOUS = 19

WWM_BOW = 20

WWM_CURIOUS = 22

WWM_DOUBLE_TAKE = 23

WWM_EXCITED = 24

WWM_FIERY = 25

WWM_HAPPY = 26

WWM_JITTERY = 27

WWM_LAUGH = 28

WWM_LONG_SHAKE = 29

```
    WWM_NO = 30

    WWM_OMINOUS = 31

    WWM_RELIEVED = 32

    WWM_SAD = 33

    WWM_SCARED = 34

    WWM_SHAKE = 35

    WWM_SURPRISED = 36

    WWM_TAUNTING = 37

    WWM_WHISPER = 38

    WWM_YELLING = 39

    WWM_YOOHOO = 40

    WWM_FRUSTRATED = 41

    IDLE_1 = 42

    IDLE_2 = 43

    IDLE_3 = 44

    EYE_1 = 45

    EYE_2 = 46

    EYE_3 = 47

    EYE_4 = 48
sensors = {'accel_one': {'accel_one': ToySensor(bit=512, min_value=0.0,
max_value=8000.0, modifier=None)}, 'accelerometer': {'x': ToySensor(bit=32768,
min_value=-8.19, max_value=8.19, modifier=None), 'y': ToySensor(bit=16384,
min_value=-8.19, max_value=8.19, modifier=None), 'z': ToySensor(bit=8192,
min_value=-8.19, max_value=8.19, modifier=None)}, 'attitude': {'pitch':
ToySensor(bit=262144, min_value=-179.0, max_value=180.0, modifier=None), 'roll':
ToySensor(bit=131072, min_value=-179.0, max_value=180.0, modifier=None), 'yaw':
ToySensor(bit=65536, min_value=-179.0, max_value=180.0, modifier=None)},
'core_time': {'core_time': ToySensor(bit=2, min_value=0.0, max_value=0.0,
modifier=None)}, 'locator': {'x': ToySensor(bit=64, min_value=-32768.0,
max_value=32767.0, modifier=<function BB9E.<lambda>>), 'y': ToySensor(bit=32,
min_value=-32768.0, max_value=32767.0, modifier=<function BB9E.<lambda>>)},
'quaternion': {'w': ToySensor(bit=4194304, min_value=-1.0, max_value=1.0,
modifier=None), 'x': ToySensor(bit=33554432, min_value=-1.0, max_value=1.0,
modifier=None), 'y': ToySensor(bit=16777216, min_value=-1.0, max_value=1.0,
modifier=None), 'z': ToySensor(bit=8388608, min_value=-1.0, max_value=1.0,
modifier=None)}, 'speed': {'speed': ToySensor(bit=4, min_value=0.0,
max_value=32767.0, modifier=None)}, 'velocity': {'x': ToySensor(bit=16,
min_value=-32768.0, max_value=32767.0, modifier=<function BB9E.<lambda>>), 'y':
ToySensor(bit=8, min_value=-32768.0, max_value=32767.0, modifier=<function
BB9E.<lambda>>)}}
```

extended_sensors = {'gyroscope': {'x': ToySensor(bit=33554432, min_value=-20000.0, max_value=20000.0, modifier=None), 'y': ToySensor(bit=16777216, min_value=-20000.0, max_value=20000.0, modifier=None), 'z': ToySensor(bit=8388608, min_value=-20000.0, max_value=20000.0, modifier=None)}}

play_animation(*animation: IntEnum*, *proc=None*)

stop_animation(*proc=None*)

enable_idle_animations(*enable: bool*, *proc=None*)

enable_trophy_mode(*enable: bool*, *proc=None*)

get_trophy_mode_enabled(*proc=None*)

ping(*data*, *proc=None*) → bytearray

get_api_protocol_version(*proc=None*) → ApiProtocolVersion

send_command_to_shell(*command: bytes*, *proc=None*)

add_send_string_to_console_listener(*listener: Callable*)

remove_send_string_to_console_listener(*listener: Callable*)

set_bluetooth_name(*name: bytes*, *proc=None*)

get_bluetooth_name(*proc=None*)

set_raw_motors(*left_mode: RawMotorModes*, *left_speed*, *right_mode: RawMotorModes*, *right_speed*, *proc=None*)

reset_yaw(*proc=None*)

drive_with_heading(*speed*, *heading*, *drive_flags: DriveFlags*, *proc=None*)

set_stabilization(*stabilization_index: StabilizationIndexes*, *proc=None*)

get_factory_mode_challenge(*proc=None*)

enter_factory_mode(*challenge: int*, *proc=None*)

exit_factory_mode(*proc=None*)

get_chassis_id(*proc=None*)

get_pending_update_flags(*proc=None*)

play_audio_file(*sound*, *playback_mode: AudioPlaybackModes*, *proc=None*)

set_all_leds_with_16_bit_mask(*mask*, *values*, *proc=None*)

start_idle_led_animation(*proc=None*)

set_sensor_streaming_mask(*interval*, *count*, *sensor_masks*, *proc=None*)

get_sensor_streaming_mask(*proc=None*)

add_sensor_streaming_data_notify_listener(*listener: Callable*)

remove_sensor_streaming_data_notify_listener(*listener: Callable*)

set_extended_sensor_streaming_mask(*sensor_masks*, *proc=None*)

get_extended_sensor_streaming_mask(*proc=None*)

enable_gyro_max_notify(*enable*, *proc=None*)

add_gyro_max_notify_listener(*listener: Callable*)

remove_gyro_max_notify_listener(*listener: Callable*)

configure_collision_detection(*collision_detection_method: CollisionDetectionMethods*, *x_threshold*, *y_threshold*, *x_speed*, *y_speed*, *dead_time*, *proc=None*)

add_collision_detected_notify_listener(*listener: Callable*)

remove_collision_detected_notify_listener(*listener: Callable*)

reset_locator_x_and_y(*proc=None*)

set_locator_flags(*locator_flags: bool*, *proc=None*)

set_accelerometer_activity_threshold(*threshold: float*, *proc=None*)

enable_accelerometer_activity_notify(*enable: bool*, *proc=None*)

add_accelerometer_activity_notify_listener(*listener: Callable*)

remove_accelerometer_activity_notify_listener(*listener: Callable*)

set_gyro_activity_threshold(*threshold: float*, *proc=None*)

enable_gyro_activity_notify(*enable: bool*, *proc=None*)

add_gyro_activity_notify_listener(*listener: Callable*)

remove_gyro_activity_notify_listener(*listener: Callable*)

enter_deep_sleep(*s*, *proc=None*)

sleep(*proc=None*)

get_battery_voltage(*proc=None*)

get_battery_state(*proc=None*)

enable_battery_state_changed_notify(*enable: bool*, *proc=None*)

add_battery_state_changed_notify_listener(*listener: Callable*)

remove_battery_state_changed_notify_listener(*listener: Callable*)

wake(*proc=None*)

add_will_sleep_notify_listener(*listener: Callable*)

remove_will_sleep_notify_listener(*listener: Callable*)

add_did_sleep_notify_listener(*listener: Callable*)

remove_did_sleep_notify_listener(*listener: Callable*)

enable_battery_voltage_state_change_notify(*enable: bool*, *proc=None*)

add_battery_voltage_state_change_notify_listener(*listener: Callable*)

remove_battery_voltage_state_change_notify_listener(*listener: Callable*)

get_main_app_version(*proc=None*)

get_bootloader_version(*proc=None*)

get_board_revision(*proc=None*)

get_mac_address(*proc=None*)

get_stats_id(*proc=None*)

get_secondary_main_app_version(*proc=None*)

add_secondary_main_app_version_notify_listener(*listener: Callable*)

remove_secondary_main_app_version_notify_listener(*listener: Callable*)

get_processor_name(*proc=None*)

get_secondary_mcu_bootloader_version(*proc=None*)

add_get_secondary_mcu_bootloader_version_notify_listener(*listener: Callable*)

remove_get_secondary_mcu_bootloader_version_notify_listener(*listener: Callable*)

get_three_character_sku(*proc=None*)

property drive_control

property firmware_update_control

property multi_led_control

property sensor_control

property stats_control

3.4.5 R2-D2

class spherov2.toy.r2d2.**R2D2**(*toy*, *adapter_cls*)

Bases: *BB9E*

toy_type = ToyType(display_name='R2-D2', prefix='D2-', filter_prefix='D2', cmd_safe_interval=0.12)

class LEDs(*value*)

Bases: IntEnum

An enumeration.

FRONT_RED = 0

FRONT_GREEN = 1

FRONT_BLUE = 2

LOGIC_DISPLAYS = 3

BACK_RED = 4

BACK_GREEN = 5

BACK_BLUE = 6

HOLO_PROJECTOR = 7

class Audio(*value*)

Bases: IntEnum

An enumeration.

TEST_1497HZ = 1

TEST_200HZ = 32

TEST_2517HZ = 63

TEST_3581HZ = 94

TEST_431HZ = 125

TEST_6011HZ = 156

TEST_853HZ = 187

BB8_ALARM_1 = 218

BB8_ALARM_10 = 235

BB8_ALARM_11 = 254

BB8_ALARM_12 = 258

BB8_ALARM_2 = 264

BB8_ALARM_3 = 268

BB8_ALARM_4 = 272

BB8_ALARM_6 = 279

BB8_ALARM_7 = 288

BB8_ALARM_8 = 296

BB8_ALARM_9 = 301

BB8_BOOT_UP = 309

BB8_BOOR_UP_2 = 330

BB8_CHATTY_1 = 352

```
BB8_CHATTY_10 = 356
BB8_CHATTY_11 = 360
BB8_CHATTY_12 = 368
BB8_CHATTY_13 = 375
BB8_CHATTY_14 = 381
BB8_CHATTY_15 = 390
BB8_CHATTY_16 = 397
BB8_CHATTY_17 = 404
BB8_CHATTY_18 = 410
BB8_CHATTY_19 = 417
BB8_CHATTY_2 = 430
BB8_CHATTY_20 = 464
BB8_CHATTY_22 = 471
BB8_CHATTY_23 = 479
BB8_CHATTY_24 = 492
BB8_CHATTY_25 = 518
BB8_CHATTY_26 = 525
BB8_CHATTY_27 = 537
BB8_CHATTY_3 = 544
BB8_CHATTY_4 = 557
BB8_CHATTY_5 = 570
BB8_CHATTY_6 = 577
BB8_CHATTY_7 = 581
BB8_CHATTY_8 = 587
BB8_CHATTY_9 = 599
BB8_DONT_KNOW = 606
BB8_EXCITED_1 = 612
BB8_EXCITED_2 = 622
BB8_EXCITED_3 = 630
BB8_EXCITED_4 = 644
BB8_HEY_1 = 671
```

BB8_HEY_10 = 682

BB8_HEY_11 = 686

BB8_HEY_12 = 690

BB8_HEY_13 = 700

BB8_HEY_2 = 724

BB8_HEY_3 = 732

BB8_HEY_4 = 734

BB8_HEY_5 = 739

BB8_HEY_6 = 743

BB8_HEY_7 = 747

BB8_HEY_8 = 753

BB8_HEY_9 = 757

BB8_LAUGH_1 = 761

BB8_LAUGH_2 = 764

BB8_NEGATIVE_1 = 787

BB8_NEGATIVE_10 = 792

BB8_NEGATIVE_11 = 802

BB8_NEGATIVE_12 = 813

BB8_NEGATIVE_13 = 825

BB8_NEGATIVE_14 = 831

BB8_NEGATIVE_15 = 836

BB8_NEGATIVE_16 = 859

BB8_NEGATIVE_17 = 867

BB8_NEGATIVE_18 = 874

BB8_NEGATIVE_19 = 888

BB8_NEGATIVE_2 = 896

BB8_NEGATIVE_20 = 902

BB8_NEGATIVE_21 = 916

BB8_NEGATIVE_22 = 927

BB8_NEGATIVE_23 = 935

BB8_NEGATIVE_24 = 946

BB8_NEGATIVE_25 = 953

BB8_NEGATIVE_26 = 963

BB8_NEGATIVE_27 = 969

BB8_NEGATIVE_28 = 983

BB8_NEGATIVE_29 = 988

BB8_NEGATIVE_3 = 998

BB8_NEGATIVE_30 = 1003

BB8_NEGATIVE_4 = 1010

BB8_NEGATIVE_5 = 1020

BB8_NEGATIVE_6 = 1032

BB8_NEGATIVE_7 = 1040

BB8_NEGATIVE_8 = 1052

BB8_NEGATIVE_9 = 1064

BB8_POSITIVE_1 = 1077

BB8_POSITIVE_10 = 1082

BB8_POSITIVE_11 = 1087

BB8_POSITIVE_12 = 1092

BB8_POSITIVE_13 = 1096

BB8_POSITIVE_14 = 1102

BB8_POSITIVE_15 = 1109

BB8_POSITIVE_16 = 1113

BB8_POSITIVE_2 = 1118

BB8_POSITIVE_3 = 1123

BB8_POSITIVE_4 = 1130

BB8_POSITIVE_5 = 1135

BB8_POSITIVE_6 = 1138

BB8_POSITIVE_7 = 1147

BB8_POSITIVE_8 = 1150

BB8_POSITIVE_9 = 1157

BB8_SAD_1 = 1163

BB8_SAD_10 = 1170

BB8_SAD_11 = 1178

BB8_SAD_12 = 1186

BB8_SAD_13 = 1192

BB8_SAD_14 = 1199

BB8_SAD_15 = 1205

BB8_SAD_16 = 1213

BB8_SAD_17 = 1220

BB8_SAD_18 = 1230

BB8_SAD_2 = 1236

BB8_SAD_3 = 1240

BB8_SAD_4 = 1250

BB8_SAD_5 = 1268

BB8_SAD_6 = 1275

BB8_SAD_7 = 1281

BB8_SAD_8 = 1295

BB8_SAD_9 = 1303

BB8_SHORTCUT = 1308

BB8_WOW_1 = 1324

BB9E_ALARM_1 = 1329

BB9E_ALARM_2 = 1347

BB9E_ALARM_3 = 1363

BB9E_ALARM_4 = 1375

BB9E_ALARM_5 = 1384

BB9E_CHATTY_1 = 1394

BB9E_CHATTY_2 = 1408

BB9E_EXCITED_1 = 1432

BB9E_EXCITED_2 = 1442

BB9E_EXCITED_3 = 1463

BB9E_HEY_1 = 1476

BB9E_HEY_2 = 1486

BB9E_NEGATIVE_1 = 1494

```
BB9E_NEGATIVE_2 = 1505

BB9E_NEGATIVE_3 = 1516

BB9E_NEGATIVE_4 = 1523

BB9E_POSITIVE_1 = 1531

BB9E_POSITIVE_2 = 1549

BB9E_POSITIVE_3 = 1558

BB9E_POSITIVE_4 = 1571

BB9E_POSITIVE_5 = 1583

BB9E_SAD_1 = 1592

BB9E_SAD_2 = 1600

R2_FALL = 1609

R2_HIT_1 = 1623

R2_HIT_10 = 1628

R2_HIT_11 = 1635

R2_HIT_2 = 1642

R2_HIT_3 = 1647

R2_HIT_4 = 1653

R2_HIT_5 = 1659

R2_HIT_6 = 1664

R2_HIT_7 = 1669

R2_HIT_8 = 1676

R2_HIT_9 = 1684

R2_STEP_1 = 1690

R2_STEP_2 = 1693

R2_STEP_3 = 1696

R2_STEP_4 = 1698

R2_STEP_5 = 1700

R2_STEP_6 = 1702

R2_ACCESS_PANELS = 1704

R2_ALARM_1 = 1737

R2_ALARM_10 = 1747
```

R2_ALARM_12 = 1756

R2_ALARM_13 = 1763

R2_ALARM_14 = 1771

R2_ALARM_15 = 1784

R2_ALARM_16 = 1791

R2_ALARM_2 = 1809

R2_ALARM_3 = 1821

R2_ALARM_4 = 1831

R2_ALARM_5 = 1835

R2_ALARM_6 = 1843

R2_ALARM_7 = 1858

R2_ALARM_8 = 1867

R2_ALARM_9 = 1893

R2_ANNOYED = 1910

R2_BURNOUT = 1915

R2_CHATTY_1 = 1950

R2_CHATTY_10 = 1959

R2_CHATTY_11 = 1966

R2_CHATTY_12 = 1977

R2_CHATTY_13 = 1987

R2_CHATTY_14 = 2002

R2_CHATTY_15 = 2007

R2_CHATTY_16 = 2010

R2_CHATTY_17 = 2019

R2_CHATTY_18 = 2028

R2_CHATTY_19 = 2039

R2_CHATTY_2 = 2061

R2_CHATTY_20 = 2072

R2_CHATTY_21 = 2080

R2_CHATTY_22 = 2085

R2_CHATTY_23 = 2095

R2_CHATTY_24 = 2105

R2_CHATTY_25 = 2121

R2_CHATTY_26 = 2132

R2_CHATTY_27 = 2143

R2_CHATTY_28 = 2157

R2_CHATTY_29 = 2170

R2_CHATTY_3 = 2174

R2_CHATTY_30 = 2184

R2_CHATTY_31 = 2188

R2_CHATTY_32 = 2198

R2_CHATTY_33 = 2202

R2_CHATTY_34 = 2211

R2_CHATTY_35 = 2221

R2_CHATTY_36 = 2232

R2_CHATTY_37 = 2241

R2_CHATTY_38 = 2253

R2_CHATTY_39 = 2264

R2_CHATTY_4 = 2276

R2_CHATTY_40 = 2285

R2_CHATTY_41 = 2292

R2_CHATTY_42 = 2307

R2_CHATTY_43 = 2322

R2_CHATTY_44 = 2332

R2_CHATTY_45 = 2344

R2_CHATTY_46 = 2357

R2_CHATTY_47 = 2368

R2_CHATTY_48 = 2377

R2_CHATTY_49 = 2387

R2_CHATTY_5 = 2399

R2_CHATTY_50 = 2413

R2_CHATTY_51 = 2424

R2_CHATTY_52 = 2439

R2_CHATTY_53 = 2452

R2_CHATTY_54 = 2457

R2_CHATTY_55 = 2463

R2_CHATTY_56 = 2474

R2_CHATTY_57 = 2492

R2_CHATTY_58 = 2509

R2_CHATTY_59 = 2519

R2_CHATTY_6 = 2524

R2_CHATTY_60 = 2535

R2_CHATTY_61 = 2543

R2_CHATTY_62 = 2554

R2_CHATTY_7 = 2562

R2_CHATTY_8 = 2572

R2_CHATTY_9 = 2579

R2_ENGAGE_HYPER_DRIVE = 2586

R2_EXCITED_1 = 2600

R2_EXCITED_10 = 2615

R2_EXCITED_11 = 2633

R2_EXCITED_12 = 2644

R2_EXCITED_13 = 2654

R2_EXCITED_14 = 2662

R2_EXCITED_15 = 2680

R2_EXCITED_16 = 2691

R2_EXCITED_2 = 2708

R2_EXCITED_3 = 2726

R2_EXCITED_4 = 2730

R2_EXCITED_5 = 2736

R2_EXCITED_6 = 2753

R2_EXCITED_7 = 2767

R2_EXCITED_8 = 2777

```
R2_EXCITED_9 = 2787

R2_HEAD_SPIN = 2797

R2_HEY_1 = 2813

R2_HEY_10 = 2824

R2_HEY_11 = 2828

R2_HEY_12 = 2833

R2_HEY_2 = 2841

R2_HEY_3 = 2856

R2_HEY_4 = 2861

R2_HEY_5 = 2882

R2_HEY_6 = 2893

R2_HEY_7 = 2898

R2_HEY_8 = 2904

R2_HEY_9 = 2912

R2_LAUGH_1 = 2919

R2_LAUGH_2 = 2935

R2_LAUGH_3 = 2950

R2_LAUGH_4 = 2955

R2_MOTOR = 2970

R2_NEGATIVE_1 = 3101

R2_NEGATIVE_10 = 3111

R2_NEGATIVE_11 = 3115

R2_NEGATIVE_12 = 3121

R2_NEGATIVE_13 = 3132

R2_NEGATIVE_14 = 3136

R2_NEGATIVE_15 = 3148

R2_NEGATIVE_16 = 3152

R2_NEGATIVE_17 = 3157

R2_NEGATIVE_18 = 3164

R2_NEGATIVE_19 = 3167

R2_NEGATIVE_2 = 3172
```

```
R2_NEGATIVE_20 = 3178

R2_NEGATIVE_21 = 3191

R2_NEGATIVE_22 = 3200

R2_NEGATIVE_23 = 3213

R2_NEGATIVE_24 = 3219

R2_NEGATIVE_25 = 3226

R2_NEGATIVE_26 = 3230

R2_NEGATIVE_27 = 3233

R2_NEGATIVE_28 = 3241

R2_NEGATIVE_3 = 3251

R2_NEGATIVE_4 = 3258

R2_NEGATIVE_5 = 3263

R2_NEGATIVE_6 = 3268

R2_NEGATIVE_7 = 3274

R2_NEGATIVE_8 = 3282

R2_NEGATIVE_9 = 3291

R2_POSITIVE_1 = 3302

R2_POSITIVE_10 = 3309

R2_POSITIVE_11 = 3318

R2_POSITIVE_12 = 3326

R2_POSITIVE_13 = 3340

R2_POSITIVE_14 = 3353

R2_POSITIVE_15 = 3358

R2_POSITIVE_16 = 3364

R2_POSITIVE_17 = 3369

R2_POSITIVE_18 = 3375

R2_POSITIVE_19 = 3388

R2_POSITIVE_2 = 3394

R2_POSITIVE_20 = 3403

R2_POSITIVE_21 = 3410

R2_POSITIVE_22 = 3422
```

R2_POSITIVE_23 = 3434

R2_POSITIVE_3 = 3439

R2_POSITIVE_4 = 3446

R2_POSITIVE_5 = 3449

R2_POSITIVE_6 = 3454

R2_POSITIVE_7 = 3460

R2_POSITIVE_8 = 3471

R2_POSITIVE_9 = 3478

R2_SAD_1 = 3484

R2_SAD_10 = 3495

R2_SAD_11 = 3518

R2_SAD_12 = 3526

R2_SAD_13 = 3536

R2_SAD_14 = 3543

R2_SAD_15 = 3553

R2_SAD_16 = 3561

R2_SAD_17 = 3570

R2_SAD_18 = 3593

R2_SAD_19 = 3600

R2_SAD_2 = 3608

R2_SAD_20 = 3612

R2_SAD_21 = 3619

R2_SAD_22 = 3632

R2_SAD_23 = 3639

R2_SAD_24 = 3649

R2_SAD_25 = 3661

R2_SAD_3 = 3686

R2_SAD_4 = 3693

R2_SAD_5 = 3703

R2_SAD_6 = 3739

R2_SAD_7 = 3755

R2_SAD_8 = 3782

R2_SAD_9 = 3790

R2_SCREAM = 3797

R2_SCREAM_2 = 3810

R2_SHORT_OUT = 3825

R2Q5_ALARM_1 = 3864

R2Q5_ALARM_2 = 3869

R2Q5_CHATTY_1 = 3875

R2Q5_CHATTY_2 = 3891

R2Q5_HEY_1 = 3896

R2Q5_HEY_2 = 3901

R2Q5_NEGATIVE_1 = 3907

R2Q5_POSITIVE_1 = 3914

R2Q5_POSITIVE_2 = 3920

R2Q5_SAD_1 = 3925

R2Q5_SHUTDOWN = 3929

BB9E_EXTRA_1 = 3941

BB9E_EXTRA_2 = 4021

BB9E_EXTRA_3 = 4246

BB9E_EXTRA_4 = 4568

BB9E_EXTRA_5 = 4929

BB9E_EXTRA_6 = 5156

BB9E_EXTRA_7 = 5315

BB9E_HEAD_TURN_1 = 5441

BB9E_HEAD_TURN_2 = 5483

BB9E_HEAD_TURN_3 = 5513

class Animations(*value*)

Bases: IntEnum

An enumeration.

CHARGER_1 = 0

CHARGER_2 = 1

```
CHARGER_3 = 2

CHARGER_4 = 3

CHARGER_5 = 4

CHARGER_6 = 5

CHARGER_7 = 6

EMOTE_ALARM = 7

EMOTE_ANGRY = 8

EMOTE_ATTENTION = 9

EMOTE_FRUSTRATED = 10

EMOTE_DRIVE = 11

EMOTE_EXCITED = 12

EMOTE_SEARCH = 13

EMOTE_SHORT_CIRCUIT = 14

EMOTE_LAUGH = 15

EMOTE_NO = 16

EMOTE_RETREAT = 17

EMOTE_FIERY = 18

EMOTE_UNDERSTOOD = 19

EMOTE_YES = 21

EMOTE_SCAN = 22

EMOTE_SURPRISED = 24

IDLE_1 = 25

IDLE_2 = 26

IDLE_3 = 27

WWM_ANGRY = 31

WWM_ANXIOUS = 32

WWM_BOW = 33

WWM_CONCERN = 34

WWM_CURIOUS = 35

WWM_DOUBLE_TAKE = 36

WWM_EXCITED = 37
```

WWM_FIERY = 38

WMM_FRUSTRATED = 39

WWM_HAPPY = 40

WWM_JITTERY = 41

WWM_LAUGH = 42

WWM_LONG_SHAKE = 43

WWM_NO = 44

WWM_OMINOUS = 45

WWM_RELIEVED = 46

WWM_SAD = 47

WWM_SCARED = 48

WWM_SHAKE = 49

WWM_SURPRISED = 50

WWM_TAUNTING = 51

WWM_WHISPER = 52

WWM_YELLING = 53

WWM_YOOHOO = 54

MOTOR = 55

extended_sensors = {'gyroscope': {'x': ToySensor(bit=33554432, min_value=-20000.0, max_value=20000.0, modifier=None), 'y': ToySensor(bit=16777216, min_value=-20000.0, max_value=20000.0, modifier=None), 'z': ToySensor(bit=8388608, min_value=-20000.0, max_value=20000.0, modifier=None)}, 'r2_head_angle': {'r2_head_angle': ToySensor(bit=67108864, min_value=-162.0, max_value=182.0, modifier=None)}}

get_battery_voltage_state(*proc=None*)

generic_raw_motor(*index: GenericRawMotorIndexes*, *mode: GenericRawMotorModes*, *speed*, *proc=None*)

perform_leg_action(*leg_action: R2LegActions*, *proc=None*)

set_head_position(*head_position: float*, *proc=None*)

get_head_position(*proc=None*)

set_leg_position(*leg_position: float*, *proc=None*)

get_leg_position(*proc=None*)

get_leg_action(*proc=None*)

enable_leg_action_notify(*enable: bool*, *proc=None*)

enable_head_reset_to_zero_notify(*enable: bool*, *proc=None*)

add_head_reset_to_zero_notify_listener(*listener: Callable*)

remove_head_reset_to_zero_notify_listener(*listener: Callable*)

set_audio_volume(*volume*, *proc=None*)

get_audio_volume(*proc=None*)

stop_all_audio(*proc=None*)

3.4.6 R2-Q5

class spherov2.toy.r2q5.**R2Q5**(*toy*, *adapter_cls*)

Bases: *R2D2*

toy_type = ToyType(display_name='R2-Q5', prefix='Q5-', filter_prefix='Q5', cmd_safe_interval=0.12)

class Animations(*value*)

Bases: IntEnum

An enumeration.

CHARGER_1 = 0

CHARGER_2 = 1

CHARGER_3 = 2

CHARGER_4 = 3

CHARGER_5 = 4

CHARGER_6 = 5

CHARGER_7 = 6

EMOTE_ALARM = 7

EMOTE_ANGRY = 8

EMOTE_ATTENTION = 9

EMOTE_FRUSTRATED = 10

EMOTE_DRIVE = 11

EMOTE_EXCITED = 12

EMOTE_SEARCH = 13

EMOTE_SHORT_CIRCUIT = 14

EMOTE_LAUGH = 15

EMOTE_NO = 16

```
EMOTE_RETREAT = 17
EMOTE_FIERY = 18
EMOTE_UNDERSTOOD = 19
EMOTE_YES = 21
EMOTE_SCAN = 22
EMOTE_SURPRISED = 24
IDLE_1 = 25
IDLE_2 = 26
IDLE_3 = 27
WWM_ANGRY = 31
WWM_ANXIOUS = 32
WWM_BOW = 33
WWM_CONCERN = 34
WWM_CURIOUS = 35
WWM_DOUBLE_TAKE = 36
WWM_EXCITED = 37
WWM_FIERY = 38
WMM_FRUSTRATED = 39
WWM_HAPPY = 40
WWM_JITTERY = 41
WWM_LAUGH = 42
WWM_LONG_SHAKE = 43
WWM_NO = 44
WWM_OMINOUS = 45
WWM_RELIEVED = 46
WWM_SAD = 47
WWM_SCARED = 48
WWM_SHAKE = 49
WWM_SURPRISED = 50
WWM_TAUNTING = 51
WWM_WHISPER = 52
```

WWM_YELLING = 53

WWM_YOOHOO = 54

3.4.7 RVR

class spherov2.toy.rvr.RVR(*toy*, *adapter_cls*)

Bases: ToyV2

toy_type = ToyType(display_name='Sphero RVR', prefix='RV-', filter_prefix='RV', cmd_safe_interval=0.075)

class LEDs(*value*)

Bases: IntEnum

An enumeration.

RIGHT_HEADLIGHT_RED = 0

RIGHT_HEADLIGHT_GREEN = 1

RIGHT_HEADLIGHT_BLUE = 2

LEFT_HEADLIGHT_RED = 3

LEFT_HEADLIGHT_GREEN = 4

LEFT_HEADLIGHT_BLUE = 5

LEFT_STATUS_INDICATION_RED = 6

LEFT_STATUS_INDICATION_GREEN = 7

LEFT_STATUS_INDICATION_BLUE = 8

RIGHT_STATUS_INDICATION_RED = 9

RIGHT_STATUS_INDICATION_GREEN = 10

RIGHT_STATUS_INDICATION_BLUE = 11

BATTERY_DOOR_REAR_RED = 12

BATTERY_DOOR_REAR_GREEN = 13

BATTERY_DOOR_REAR_BLUE = 14

BATTERY_DOOR_FRONT_RED = 15

BATTERY_DOOR_FRONT_GREEN = 16

BATTERY_DOOR_FRONT_BLUE = 17

POWER_BUTTON_FRONT_RED = 18

POWER_BUTTON_FRONT_GREEN = 19

POWER_BUTTON_FRONT_BLUE = 20

POWER_BUTTON_REAR_RED = 21

POWER_BUTTON_REAR_GREEN = 22

POWER_BUTTON_REAR_BLUE = 23

LEFT_BRAKELIGHT_RED = 24

LEFT_BRAKELIGHT_GREEN = 25

LEFT_BRAKELIGHT_BLUE = 26

RIGHT_BRAKELIGHT_RED = 27

RIGHT_BRAKELIGHT_GREEN = 28

RIGHT_BRAKELIGHT_BLUE = 29

UNDERCARRIAGE_WHITE = 30

ping(*data*, *proc=None*) → bytearray

get_api_protocol_version(*proc=None*) → ApiProtocolVersion

send_command_to_shell(*command: bytes*, *proc=None*)

add_send_string_to_console_listener(*listener: Callable*)

remove_send_string_to_console_listener(*listener: Callable*)

get_supported_dids(*proc=None*)

get_supported_cids(*s*, *proc=None*)

get_main_app_version(*proc=None*)

get_bootloader_version(*proc=None*)

get_board_revision(***, *proc=Processors.PRIMARY*)

get_mac_address(***, *proc=Processors.PRIMARY*)

get_stats_id(***, *proc=Processors.PRIMARY*)

get_processor_name(*proc=None*)

get_boot_reason(***, *proc=Processors.PRIMARY*)

get_last_error_info(***, *proc=Processors.PRIMARY*)

write_config_block(***, *proc=Processors.PRIMARY*)

get_config_block(***, *proc=Processors.PRIMARY*)

set_config_block(*metadata_version*, *config_block_version*, *application_data*, ***, *proc=Processors.PRIMARY*)

erase_config_block(*j*, ***, *proc=Processors.PRIMARY*)

get_swd_locking_status(*proc=None*)

get_manufacturing_date(*, *proc=Processors.PRIMARY*)

get_sku(*, *proc=Processors.PRIMARY*)

get_core_up_time_in_milliseconds(*, *proc=Processors.PRIMARY*)

get_event_log_status(*proc=None*)

get_event_log_data(*j*, *j2*, *proc=None*)

clear_event_log(*proc=None*)

enable_sos_message_notify(*enable: bool*, *, *proc=Processors.PRIMARY*)

add_sos_message_notify_listener(*listener: Callable*)

remove_sos_message_notify_listener(*listener: Callable*)

get_sos_message(*, *proc=Processors.PRIMARY*)

clear_sos_message(*, *proc=Processors.PRIMARY*)

get_out_of_box_state(*, *proc=Processors.PRIMARY*)

enable_out_of_box_state(*enable: bool*, *, *proc=Processors.PRIMARY*)

enter_deep_sleep(*s*, *, *proc=Processors.PRIMARY*)

sleep(*, *proc=Processors.PRIMARY*)

force_battery_refresh(*, *proc=Processors.PRIMARY*)

wake(*, *proc=Processors.PRIMARY*)

get_battery_percentage(*, *proc=Processors.PRIMARY*)

get_battery_voltage_state(*, *proc=Processors.PRIMARY*)

add_will_sleep_notify_listener(*listener: Callable*)

remove_will_sleep_notify_listener(*listener: Callable*)

add_did_sleep_notify_listener(*listener: Callable*)

remove_did_sleep_notify_listener(*listener: Callable*)

enable_battery_voltage_state_change_notify(*enable: bool*, *, *proc=Processors.PRIMARY*)

add_battery_voltage_state_change_notify_listener(*listener: Callable*)

remove_battery_voltage_state_change_notify_listener(*listener: Callable*)

get_battery_voltage_in_volts(*reading_type: BatteryVoltageReadingTypes*, *, *proc=Processors.PRIMARY*)

get_battery_voltage_state_thresholds(*, *proc=Processors.PRIMARY*)

get_current_sense_amplifier_current(*amplifier_id: AmplifierIds*, *, *proc=Processors.PRIMARY*)

get_efuse_fault_status(*efuse_id: EfuseIds*, *, *proc=Processors.PRIMARY*)

add_efuse_fault_occurred_notify_listener(*listener: Callable*)

remove_efuse_fault_occurred_notify_listener(*listener: Callable*)

enable_efuse(*efuse_id: EfuseIds*, *, *proc=Processors.PRIMARY*)

set_raw_motors(*left_mode: RawMotorModes*, *left_speed*, *right_mode: RawMotorModes*, *right_speed*, *, *proc=Processors.SECONDARY*)

reset_yaw(*, *proc=Processors.SECONDARY*)

drive_with_heading(*speed*, *heading*, *drive_flags: DriveFlags*, *, *proc=Processors.SECONDARY*)

set_control_system_type(*s*, *s2*, *, *proc=Processors.SECONDARY*)

set_component_parameters(*s*, *s2*, *f_arr*, *, *proc=Processors.SECONDARY*)

get_component_parameters(*s*, *s2*, *, *proc=Processors.SECONDARY*)

set_custom_control_system_timeout(*timeout*, *, *proc=Processors.SECONDARY*)

enable_motor_stall_notify(*enable*, *, *proc=Processors.SECONDARY*)

add_motor_stall_notify_listener(*listener: Callable*)

remove_motor_stall_notify_listener(*listener: Callable*)

enable_motor_fault_notify(*enable*, *, *proc=Processors.SECONDARY*)

add_motor_fault_notify_listener(*listener: Callable*)

remove_motor_fault_notify_listener(*listener: Callable*)

get_motor_fault_state(*, *proc=Processors.SECONDARY*)

enable_gyro_max_notify(*enable*, *, *proc=Processors.SECONDARY*)

add_gyro_max_notify_listener(*listener: Callable*)

remove_gyro_max_notify_listener(*listener: Callable*)

reset_locator_x_and_y(*, *proc=Processors.SECONDARY*)

set_locator_flags(*locator_flags: bool*, *, *proc=Processors.SECONDARY*)

get_bot_to_bot_infrared_readings(*, *proc=Processors.SECONDARY*)

get_rgbc_sensor_values(*, *proc=Processors.PRIMARY*)

magnetometer_calibrate_to_north(*, *proc=Processors.SECONDARY*)

add_magnetometer_north_yaw_notify_listener(*listener: Callable*)

remove_magnetometer_north_yaw_notify_listener(*listener: Callable*)

start_robot_to_robot_infrared_broadcasting(*far_code*, *near_code*, *, *proc=Processors.SECONDARY*)

start_robot_to_robot_infrared_following(*far_code*, *near_code*, *, *proc=Processors.SECONDARY*)

stop_robot_to_robot_infrared_broadcasting(*, *proc=Processors.SECONDARY*)

add_robot_to_robot_infrared_message_received_notify_listener(*listener: Callable*)

remove_robot_to_robot_infrared_message_received_notify_listener(*listener: Callable*)

get_ambient_light_sensor_value(*, *proc=Processors.PRIMARY*)

stop_robot_to_robot_infrared_following(*, *proc=Processors.SECONDARY*)

start_robot_to_robot_infrared_evading(*far_code*, *near_code*, *, *proc=Processors.SECONDARY*)

stop_robot_to_robot_infrared_evading(*, *proc=Processors.SECONDARY*)

enable_color_detection_notify(*enable*, *interval*, *minimum_confidence_threshold*, *, *proc=Processors.PRIMARY*)

add_color_detection_notify_listener(*listener: Callable*)

remove_color_detection_notify_listener(*listener: Callable*)

get_current_detected_color_reading(*, *proc=Processors.PRIMARY*)

enable_color_detection(*enable*, *, *proc=Processors.PRIMARY*)

configure_streaming_service(*token*, *configuration*, *proc=None*)

start_streaming_service(*period*, *proc=None*)

stop_streaming_service(*proc=None*)

clear_streaming_service(*proc=None*)

add_streaming_service_data_notify_listener(*listener: Callable*)

remove_streaming_service_data_notify_listener(*listener: Callable*)

enable_robot_infrared_message_notify(*enable*, *, *proc=Processors.SECONDARY*)

send_infrared_message(*infrared_code*, *front_strength*, *left_strength*, *right_strength*, *rear_strength*, *, *proc=Processors.SECONDARY*)

add_motor_current_notify_listener(*listener: Callable*)

remove_motor_current_notify_listener(*listener: Callable*)

enable_motor_current_notify(*enable*, *, *proc=Processors.SECONDARY*)

get_motor_temperature(*motor_index*, *, *proc=Processors.SECONDARY*)

configure_sensitivity_based_collision_detection(*method: SensitivityBasedCollisionDetectionMethods*, *level: SensitivityLevels*, *i*, *, *proc=Processors.SECONDARY*)

enable_sensitivity_based_collision_detection_notify(*enable*, *, *proc=Processors.SECONDARY*)

add_sensitivity_based_collision_detected_notify_listener(*listener: Callable*)

remove_sensitivity_based_collision_detected_notify_listener(*listener: Callable*)

get_motor_thermal_protection_status(*, proc=Processors.SECONDARY*)

enable_motor_thermal_protection_status_notify(*enable*, *, *proc=Processors.SECONDARY*)

add_motor_thermal_protection_status_notify_listener(*listener: Callable*)

remove_motor_thermal_protection_status_notify_listener(*listener: Callable*)

set_bluetooth_name(*name: bytes*, *, *proc=Processors.PRIMARY*)

get_bluetooth_name(*, *proc=Processors.PRIMARY*)

get_bluetooth_advertising_name(*, *proc=Processors.PRIMARY*)

set_all_leds_with_32_bit_mask(*mask*, *values*, *, *proc=Processors.PRIMARY*)

set_compressed_frame_player_one_color(*r*, *g*, *b*, *, *proc=Processors.PRIMARY*)

save_compressed_frame_player_animation(*s*, *s2*, *z*, *s3*, *s_arr*, *i*, *i_arr*, *, *proc=Processors.PRIMARY*)

play_compressed_frame_player_animation(*s*, *, *proc=Processors.PRIMARY*)

play_compressed_frame_player_frame(*i*, *, *proc=Processors.PRIMARY*)

get_compressed_frame_player_list_of_frames(*, *proc=Processors.PRIMARY*)

delete_all_compressed_frame_player_animations_and_frames(*, *proc=Processors.PRIMARY*)

pause_compressed_frame_player_animation(*, *proc=Processors.PRIMARY*)

resume_compressed_frame_player_animation(*, *proc=Processors.PRIMARY*)

reset_compressed_frame_player_animation(*, *proc=Processors.PRIMARY*)

add_compressed_frame_player_animation_complete_notify_listener(*listener: Callable*)

remove_compressed_frame_player_animation_complete_notify_listener(*listener: Callable*)

assign_compressed_frame_player_frames_to_animation(*s*, *i*, *i_arr*, *, *proc=Processors.PRIMARY*)

save_compressed_frame_player_animation_without_frames(*s*, *s2*, *z*, *s3*, *s_arr*, *i*, *, *proc=Processors.PRIMARY*)

play_compressed_frame_player_animation_with_loop_option(*s*, *z*, *, *proc=Processors.PRIMARY*)

get_active_color_palette(*, *proc=Processors.PRIMARY*)

set_active_color_palette(*rgb_index_bytes*, *, *proc=Processors.PRIMARY*)

get_color_identification_report(*red*, *green*, *blue*, *confidence_threshold*, *, *proc=Processors.PRIMARY*)

load_color_palette(*palette_index*, *, *proc=Processors.PRIMARY*)

save_color_palette(*palette_index*, *, *proc=Processors.PRIMARY*)

get_compressed_frame_player_frame_info_type(*, *proc=Processors.PRIMARY*)

save_compressed_frame_player16_bit_frame(*i*, *i2*, *i3*, *i4*, *i5*, *, *proc=Processors.PRIMARY*)

release_led_requests(*, *proc=Processors.PRIMARY*)

get_current_application_id(*proc=None*)

get_all_updatable_processors(*, *proc=Processors.PRIMARY*)

get_version_for_updatable_processors(*, *proc=Processors.PRIMARY*)

set_pending_update_for_processors(*data*, *, *proc=Processors.PRIMARY*)

get_pending_update_for_processors(*, *proc=Processors.PRIMARY*)

reset_with_parameters(*strategy*, *, *proc=Processors.PRIMARY*)

clear_pending_update_processors(*data*, *, *proc=Processors.PRIMARY*)

get_factory_mode_challenge(*proc=None*)

enter_factory_mode(*challenge: int*, *proc=None*)

exit_factory_mode(*proc=None*)

get_chassis_id(*proc=None*)

enable_extended_life_test(*enable*, *, *proc=Processors.PRIMARY*)

get_factory_mode_status(*proc=None*)

property drive_control

property multi_led_control

property sensor_control

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

W