Neosensory Developer API

Last Update: 04-21-2021

Introduction

The Neosensory Developer API enables anyone to stream custom vibrations to the Buzz and other Neosensory products over Bluetooth Low Energy (BLE). One can also connect over USB (using a Serial terminal application with a baud rate of 9600), but BLE is better to work with for practical purposes as it enables the device to be worn. To work with BLE, the developer is generally responsible for performing all BLE tasks relating to scanning, pairing, connecting, etc. There are a number of platform-dependent libraries that can help with this. Neosensory is also preparing Software Development Kits (SDKs) to help streamline this process.

JSON Formatting

The API produces JSON formatted responses, with the following structure:

{"status_code": <status code>, "message": <message> or "data": { <JSON data structure> } or "error": <message>}

with the following status codes:

Status Codes	Description
200	Success
400	Bad Request/Error

Access and Usage of the API and SDK

The Developer API and SDK are currently free for anyone to access and use subject to the acceptance of Neosensory's Developer Terms and Conditions. Using the Developer API requires an explicit acceptance command (see API documentation below).

Bluetooth Low Energy

Neosensory uses the Generic Attribute Profile (GATT) interacting with its hardware.

Advertising data:

- Friendly name : string "<PREFIX><MACADDR>" eg. "BuzzF98D6D6B0234"
- Appearance: Generic Watch

Relevant UUIDs:

UART over BLE Service **(UUID: 6E400001-B5A3-F393-E0A9-E50E24DCCA9E)**

Characteristics:

- UART RX **(UUID: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E)**
    - Write without response and write
- UART TX **(UUID: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E)**
    - Notify

Motor Control

The "motors vibrate" command allows direct control over the Buzz's motors. It takes as an argument a base64 encoded string representing "motor control frames". Each motor control frame consists of 4 bytes with each byte representing the amplitude of a single motor. Neosensory Buzz can queue up to 64 motor control frames. It plays them with a set interval of 16 ms between frames. The previous frame persists until the next frame is available in the queue. Depending on your system constraints and BLE parameters negotiated with Buzz, you may not be able be to transmit perfectly a "motors vibrate" command periodically every 16 ms. To achieve a smooth playback, you can transmit several motors control frames in a single "motors vibrate" command. To send multiple control frames, the 4 byte motor control frames need to be concatenated sequentially before encoding with base64. If your device has successfully negotiated the largest BLE MTU size with Buzz, you can send up to 42 frames in a single "motors vibrate" command. If the motor queue was full, it will return an error message saying so.

Future Enhancement: Buzz will respond with the frame index at which it stopped filling its queue along with the error message currently reported

Developer API Commands

Command	**Minimum Firmware Level**	**Description**
`auth as developer`	< 1.5.4.132	Request developer authorization. Returns the message "Please type 'accept' and hit enter to agree to Neosensory Inc's Developer Terms and Conditions, which can be viewed at https://neosensory.com/legal/dev-terms-service"
`accept`	< 1.5.4.132	After successfully calling `auth as developer`, use the `accept` command to agree to the Neosensory [Developer Terms and Conditions](https://neosensory.com/legal/dev-terms-service). Successfully calling this unlocks the following commands: `audio start`, `audio stop`, `motors_clear_queue`, `motors start`, `motors_stop`, `motors vibrate`.
`audio start`	< 1.5.4.132	(Re)starts the device's microphone audio acquisition. This command requires successful developer authorization, otherwise, the command will fail.
`audio stop`	< 1.5.4.132	Stop the device's microphone audio acquisition. This should be called prior to transmitting motor vibration data. This command requires successful developer authorization, otherwise, the command will fail.
`device battery_soc`	< 1.5.4.132	Obtain the device's battery level in %. This command does not require developer authorization.
`device info`	< 1.5.4.132	Obtain various device and firmware information. This command does not require developer authorization.
`motors clear_queue`	< 1.5.4.132	Clear any vibration commands sitting the device's motor FIFO queue. This should be called prior to streaming control frames using `motors vibrate`.
`motors start`	< 1.5.4.132	Initialize and start the motors interface. The motors can then accept motors vibrate commands.
`motors stop`	< 1.5.4.132	Clear the motors command queue and shut down the motor drivers.

`motors vibrate <control frames>`	< 1.5.4.132	<control frames> is a string represented by a base64 encoded byte array where each byte represents motor intensity for each motor. Multiple motor control frames can be sent as concatenated byte arrays. The FIFO queue size on the band is 64 control frames played out every 16ms. Examples: All OFF - `motors vibrate AAAAAA==` Motor 0 - `motors vibrate /wAAAA==` Motor 1 - `motors vibrate AP8AAA==` Motor 2 - `motors vibrate AAD/AA==` Motor 3 - `motors vibrate AAAA/w==` Motor 3 ON for 16ms and then OFF - `motors vibrate AAAA/wAAAAA=` **--- for FW 1.5.4.132 and above ---** The band's response to this command is optional, and controlled by the `motors config_threshold` command. If a response arrives, below are the potential responses: **"status_code": 400 response parameters** "motor queue full": The internal motor queue is full. The depth of the motor queue can be requested using the `motors get_threshold` command. "motors are stopped": You must send `motors start` command prior to sending this command. **"status_code" 200 response parameters:** "queue_status": integer represented as char, range: 0 - 2 This parameter reports that status of the depth of the motor queue, based on the `motors config_threshold` command. 0 = Threshold not yet reached 1 = Threshold reached 2 = Threshold exceeded "curr_queue_depth": decimal integer represented as char array, range: 0 - 64

		The number of `motors vibrate` commands in the queue.
`motors config_threshold <feedback type> <threshold>`	1.5.4.132	This command controls how the band responds to the `motors vibrate` command listed above. <feedback type>: integer represented as char, range: 0 - 2 0 = Default (same behavior as firmware pre 1.5.4.132). In this configuration, the motors vibrate does not return a response unless an error occurs. Potential errors are listed above. 1 = Always respond. In this configuration, the `motors vibrate` command always returns a response. 2 = Threshold response. In this configuration, the `motors vibrate` command only returns a response if the threshold is reached or exceeded. <threshold>: decimal integer represented as char array, range: 0 - 64 This parameter controls the threshold at which the band will respond to a `motors vibrate` command if the feedback type is set to 2 (threshold response).

`motors get_threshold`	1.5.4.132	This command returns the current `motors vibrate` command queue configuration. "feedback_type": This parameter is set by the `motors config_threshold` command. Default = 0. "high_threshold": This parameter is set by the `motors config_threshold` command. Default = max_queue_depth. "max_queue_depth": This parameter is the hard coded maximum queue depth for the firmware. Default = 64.
`motors config_lra_mode <mode>`	1.5.4.132	This command sets the LRA operation mode. This setting is not persistent, and will reset to the default (open loop) if the band is reset. <mode>: integer represented as char, range: 0 - 1 0 = LRA open loop mode 1 = LRA closed loop mode **WARNING: Closed loop mode is not designed for applications where the motors will be vibrating persistently over long stretches of time. Using closed loop mode in this way may damage you or your users' devices and reduce LRA lifespan. Neosensory will not be held liable or take responsibility for any damage incurred by running the devices with closed loop mode.**
`motors get_lra_mode`	1.5.4.132	This command allows you to read the current LRA vibration mode. "lra_mode": The current LRA operation mode: 0 = LRA open loop mode (default) 1 = LRA closed loop mode

`leds set <LED 1 color> <LED 2 color> <LED 3 color> <LED 1 intensity> <LED 2 intensity> <LED 3 intensity>`	1.5.4.132	This command allows you to control each of the band's 3 LED's color and intensity. <LED X color>: hex integer represented as char array, range: 0 - 0xFFFFFF This parameter controls the RGB color of the respective LED. The upper 8-bits represent the intensity of red, the middle 8-bits represent the intensity of green, and the lower 8-bits represent the intensity of blue. For example: 0xFF0000 = red 0xFF00 = green 0xFF = blue 0xFFFFFF = white. (Web Colors) <LED X intensity>: decimal integer represented as char array, range: 0 - 50 0 = Off 50 = Max glow The parameter controls the intensity of the LED.
`leds get`	1.5.4.132	This command allows you to read the current state of the LEDs. "color": array of decimal integers: [LED 1 color, LED 1 color, LED 1 color] The parameter contains an array of the RGB color codes that were last set for each LED. "intensity": array of decimal integers: [LED 1 intensity, LED 1 intensity, LED 1 intensity] This parameter contains an array of intensities, representing the current intensity of each LED.

`config set_buttons_response <enable feedback> <allow sensitivity changes>`	1.5.4.132	<enable feedback>: integer represented as char, range: 0 - 1 0 = disabled, the band will not generate a CLI response when a button is pressed. 1 = enabled, the band will send an unsolicited CLI response each time any button is pressed. The unsolicited message format is listed below. <allow sensitivity changes>: integer represented as char, range: 0 - 1 0 = not allowed, when a user presses the +/- buttons, the microphone intensity will not be changed, and no LED activity will be generated. 1 = allowed, when a user presses the +/- buttons, the microphone sensitivity will be changed, and the LEDs will indicate the current microphone sensitivity. **Unsolicited button response format:** When this command's parameter <enable feedback> = 1, and a user presses a button, the following message will be sent: "status_code": 201 "type": "button_press" "button_val":

		1 = Plus button was pressed 2 = Power button was pressed 3 = Minus button was pressed