PSYONIC

# ABILITY HAND™ Interface Control Document

## Introduction:

The PSYONIC Ability Hand has a number of communication interfaces allowing information to be sent to the hand from outside sources. These are used for settings changes, relaying sensor information, and various modes of motion control. The purpose of this document is to provide documentation on each interface as available in Ability Hand firmware v2.0+. This document documents the latest firmware and PSYONIC recommends that all hands are updated when new firmware is available.

The interfaces are:

1. Standard Mode (I2C)
2. Extended Mode (UART/RS485/I2C)
3. Bluetooth Low Energy
4. Analog Inputs

## Contents

# 0 Technical Specifications

**Wrist Configurations**

The Ability Hand is available in two wrist configurations: the Bolt Pattern-Research Wrist and Clinical Wrist. An overview of each configuration's features are shown below in Table 1.

*Table 1: Ability Hand Wrist Configuration Options*

| Option | Bolt mount | Quick disconnect mount | Half moon connectors | Breakout header connector | Wrist rotator compatible |
|---|---|---|---|---|---|
| Bolt Pattern-Research Wrist | ✔ | | | ✔ | |
| Clinical Wrist | | ✔ | ✔ | | ✔ |

# 1 Electrical Specification

*Table 2: Electrical Characteristics (Mk. 9.5)*

| Symbol | Ratings | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| +V | Standard Operating Voltage | 6.0 | | 12.0 | V |
| $I_q$ | Quiescent Current Consumption | - | 74 | - | mA |
| $I_{max}$ | Peak Current Consumption | - | - | 14 | A |
| AN1, AN2 | Analog Input Voltage | 0 | - | 5 | V |

**Wiring for physical interfaces**

A 6-wire interface to the hand is provided through a Molex 0874380643 connector for Research Wrists and through a slip ring electronic quick disconnect (EQD) for Clinical Wrists. The EQD allows industry standard half moon connectors to be used for all signals. Research wrists have these connections also available on a 0.1" breakout header.

*Table 3: Signal pinouts*

| Name | Clinical wrist half moons | Research wrist Molex 0874380643 | Breakout 0.1" header |
|---|---|---|---|
| +V | 3 pin PWR (red) | Pin 1 | Pin 2, 4 |
| GND | 3 pin PWR (black) | Pin 2 | Pin 1, 6, 8, 10 |
| AN1 | 3 pin electrode | Pin 3 | Pin 3 |
| AN2 | 3 pin electrode | Pin 4 | Pin 5 |
| SDA | 2 pin COM (green) | Pin 5 | Pin 7 |
| SCL | 2 pin COM (yellow) | Pin 6 | Pin 9 |

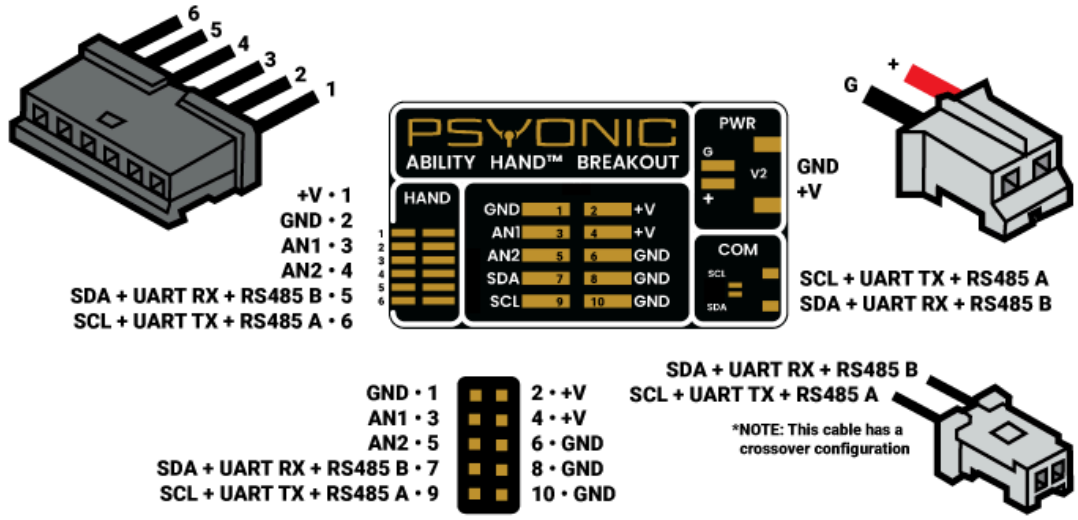
*Table 4: Functional description*

| Name | Signal | Function |
|---|---|---|
| +V | Power | System power |
| GND | Ground | System ground |
| AN1 | Analog Input 1 | Analog control of the hand |
| AN2 | Analog Input 2 | Analog control of the hand |

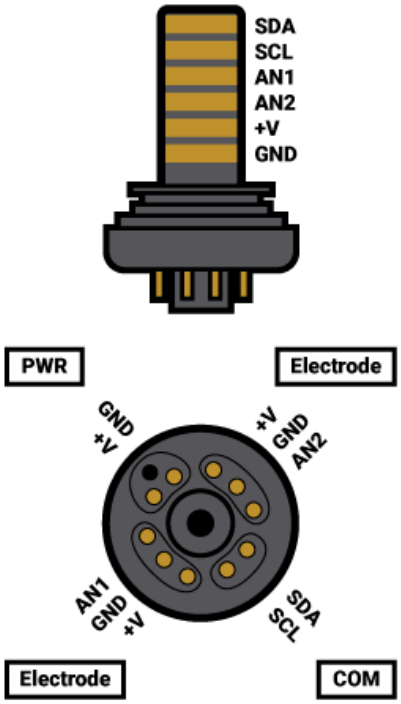
| | | |
|---|---|---|
| SDA | SDA/UART RX/RS485 B | Digital control of the hand* |
| SCL | SCL/UART TX/RS485 A | Digital control of the hand* |

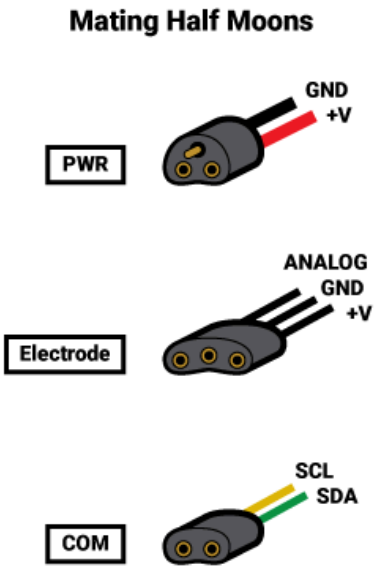
**Can support both 3.3V I2C and UART. Newer hardware revisions of the hand also support 5V level RS485 on those connections. The communication mode for these signals is stored in flash memory, and must be selected and configured using bluetooth command options.*

**Pinout for Research Wrist Option:**




**Pinout for Clinical Wrist Option (Electronic Quick Disconnect):**


SDA
SCL
AN1
AN2
+V
GND
PWR
Electrode
GND
+V
+V
GND
AN2
AN1
GND
+V
SDA
SCL
Electrode
COM


Mating Half Moons
GND
+V
PWR
ANALOG
GND
+V
Electrode
SCL
SDA
COM

# 2 Standard Communication Interface

**Introduction**
This communication interface is intended for use of the Ability Hand in clinical applications. The hand has 2 different packet structures and 4 different control modes. The first covers the 'industry standard' *Grip Control Mode* interface, consisting of a 3-byte transmission and 7-byte reception, which allows the master device to select a grip index and speed, and read back some basic status information. The second packet structure covers the *Finger Control Mode* interface, which allows for individual control of the fingers and direct touch sensor readings. This has a 25-byte transmission packet and a reception packet ranging from 24-64 bytes in length.

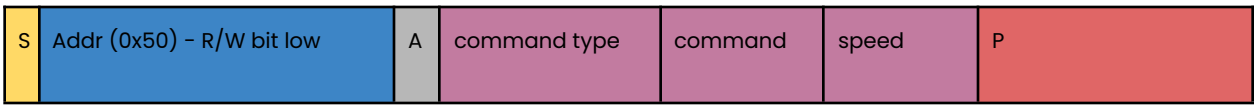
**Important Note:** I2C is accessed through pins 5 and 6 of the cable harness for the hand. I2C mode must be enabled using the relevant bluetooth flash setting.

## 2.1 I2C Specification

The hand supports both standard (100kHz) and fast (400kHz) I2C modes. When operating in 'fast mode', the cable length of the hand should be limited to preserve packet integrity. In general, it is recommended to keep the I2C cable length as short as possible between the master and slave devices. The slave address of the hand is always 0x50. The hand has built in 10k pullup resistors, so external I2C pullup resistors are not necessary for a functional interface.

## 2.2 Grip Control Mode

*Packet Structure*:
**Transmissions: (From Master to Hand)**




1. *Hand slave address*:
   0x50
2. *Command type*:

| Value | Command | Description |
|---|---|---|
| 0x1D | HAND_SET_GRASP | Normal control mode. Hand will parse data bytes for grip control |
| 0x48 | HAND_STATUS | Handshaking. Hand will reply with 0xDEADBEEF |

3. *Grip Commands*: *(Valid for packets starting with 0x1D)*

| Value | Command | Description |
|---|---|---|
| 0x00 | Open | Opens the hand. Open configuration will vary depending on the last grip. |
| 0x01 | Power | All digits flexed with the thumb rotated internally. |
| 0x02 | Key | Also known as lateral grasp. When key grasp is opened, only the thumb flexor extends and all other digits remain flexed. |
| 0x03 | Pinch | Precision pinch with index finger and thumb. |
| 0x04 | Tripod Opened | 3-jaw chuck where the pinky and ring fingers are extended. |
| 0x05 | Sign of the Horns | \m/ |
| 0x06 | Cylinder | Thumb will touch the middle and ring fingers |
| 0x07 | Mouse Grasp | Grip which allows the use of a computer mouse |
| 0x08 | Power/Key Switch | When the thumb is rotated internally, the hand will close into a power grasp. When the thumb is rotated externally, the hand will close into a key grasp |
| 0x09 | Point | Index finger point with the thumb rotated internally and flexed. |

| | | |
|---|---|---|
| 0x0A | Rude Point | Middle finger point |
| 0x0B | Hook Grasp | Static finger hooked configuration |
| 0x0C | Relax Grip | Puts the hand in a ‘relaxed’ configuration. |
| 0x0D | Sleeve Grip | Puts the thumb out of the way so the hand can be guided through the sleeve of a shirt or jacket |
| 0x0E | Peace Grip | Makes an approximation of a ‘peace’ hand sign |
| 0x0F | Tripod Closed | 3-jaw chuck where the ring and pinky are flexed |
| 0x10 | Hangloose Grip | Makes the ‘hangloose’ hand gesture |
| 0x11 | Handshake Grip | Halfway between a key grip and a power grip |
| 0x12 | Fixed Pinch Grip | Pinch where the thumb doesn’t move while opening |
| 0x13 | User Grip 7 | User defined grip |
| 0x14 | User Grip 8 | User defined grip |
| 0x15 | User Grip 9 | User defined grip |
| 0x16 | User Grip 10 | User defined grip |
| 0x17 | User Grip 11 | User defined grip |
| 0x18 | Trigger Grip | For grabbing/pulling trigger on spray bottles |
| 0x19 | User Grip 12 | User defined grip |
| 0x1A | User Grip 13 | User defined grip |
| 0x1B | User Grip 14 | User defined grip |
| 0x1C | User Grip 15 | User defined grip |
| 0x1D | User Grip 16 | User defined grip |
| 0x1E | User Grip 17 | User defined grip |
| 0x1F | User Grip 18 | User defined grip |
| 0x20 | Finger Wave | If enabled (enabled by default) this grip will make the fingers move periodically and out of phase in a ‘waggle’ pattern. If disabled, this is a slot for another user defined grip. |

*Note: all ‘User defined grips’ will result in no motion if selected, until modified using the PSYONIC mobile app or otherwise through the BLE interface

4. *Speed*:

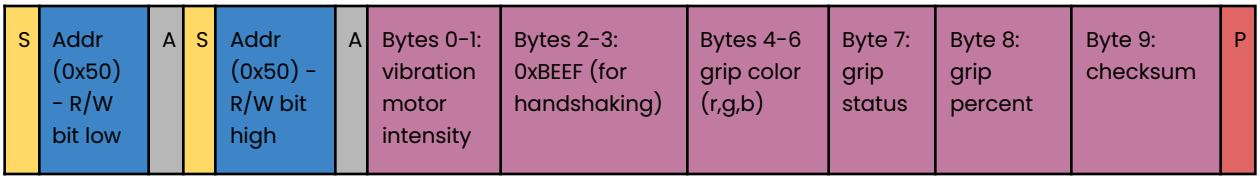
4.1 When this byte is 0, the hand will stop.

4.2 When this byte is between 1 and 254, the **finger period** will vary linearly between 2 seconds and .29 seconds. A value of 1 will set the finger period to 2 seconds, and a value of 254 will set the finger period to .29 seconds. Finger period is defined as the allotted time for a finger to move through its motion path, and multiple periods can elapse over the course of a given grip.

4.3 When this byte is 255, the finger period will be set to .2 seconds.

**Receptions: (From Hand to Master)**

1. HAND_SET_GRASP mode (byte 0 of last transmission was 0x1D):




Bytes 0-1 comprise a 16-bit integer that maps to the maximum touch sensor value, for use in controlling our vibration motor.

Byte 7 will return 0xFF when the grip is in progress, and for closed grips, the grip command of the last requested grip from the master when the grip is completed. For opened grips, due to legacy support of an old COAPT protocol, Byte 7 may sometimes be zero, and sometimes be a non-zero value not equal to the current grasp command. A value not equal to 0xFF will indicate that the hand has completed its motion.

Byte 8 will contain a concatenation of a grip percentage, and a motor driver power status bit.

Bit 7: if all 6 motor drivers are disabled (hand in idle state), will have value 1. If any one motor driver is active, will have value 0

Bits 6-0: Grip completion percentage (7bit; i.e. 127 corresponds to 100%)

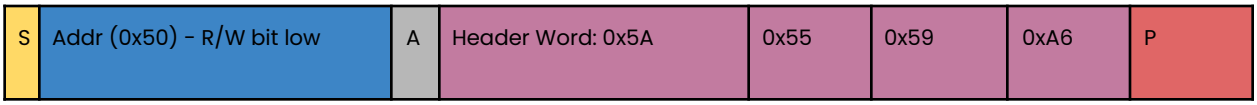
Byte 9 will contain a checksum of the entire packet. The 8-bit signed sum of bytes 0-9 will be equal to 0 if there were no communication errors.

## 2.3 Tap Notification

*Packet Structure*:
**Transmissions: (From Master to Hand)**

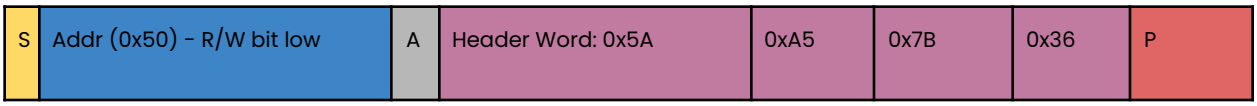
Tap messages are to be sent by a valid PSYONIC IO board. The full 4 byte message is sent once per tap, and each 4 byte word corresponds to a different tap type.

*Single Tap*:



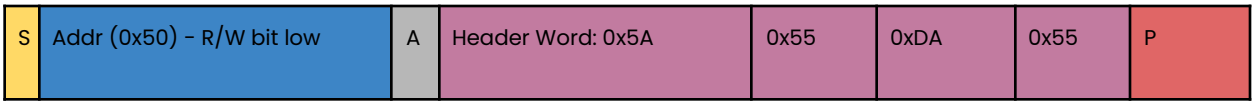

Default action: no action

*Double Tap*:




Default action: the hand will switch to the next grip in the direct control grip set (4 grips, circular buffer).

*Triple Tap*:




Default action: the hand will enter 'freeze mode'. Direct control inputs are temporarily ignored while in

freeze mode, and the hand will indicate freeze mode by blinking the requested RGB led color.

# 3 Extended Mode API Interface[1]

This interface is intended for use by researchers for control of the hand with high levels of access to the hardware resources on the hand. It is supported in 3.3V level I2C, UART, and 5V level RS485[2]. Communication modes (I2C, UART, RS485) must be selected with a flash setting, configurable over BLE. By default, the hand is configured for I2C.

The Extended Mode API interface is entered when the hand receives a valid transmission with a correct address, format header, and checksum[3] in the protocol current selected by the flash setting. The hand will leave 'normal' mode and be controlled via the parsed data in one of the formats described in following sections[4]. The hand will operate based on the API command for a maximum of 300ms, if no new command is received within 300ms it will automatically exit API mode. The time to enter API mode is negligible, but care should be taken to ensure turnaround time between valid transmissions does not exceed 300ms in order to prevent the hand from re-entering normal mode which may be accompanied by undesired finger movements.

## 3.1 Communication Protocol Specifications

### 3.1.1 I2C Specification

I2C API Mode is available over the SCL/SDA pins from the hand (see Table 3). The Extended API supports I2C in 100kHz (normal) and 400kHz (fast) modes. In fast mode, cable length must be considered to ensure data integrity. The hand has internal 10kΩ pull-up resistors on the I2C lines, external resistors are not required. In I2C mode the ability hand is always at address 0x50.

Following are sample I2C Transmission Packets, full details of the API are in section 3.2.

Table 5: Sample I2C Transmission Packet Format (Master to Hand)

| - | | | Byte 0 | Bytes 1 - 12 | Byte 13 | - |
|---|---|---|---|---|---|---|
| S | Addr (0x50) - R/W bit low | A | Format header | Payload. One value per motor (6x total values), either voltage, position, velocity, or current (i.e. torque) request. | Checksum | P |

Table 6: Sample I2C Reception Packet Format (Hand to Master)

[1] Extended Mode API available from software version 2.0 and higher
[2] RS485 is only available on hardware version 10.0 and newer
[3] In fast mode for UART & RS485 the checksum is not checked prior to entering API mode
[4] Special Read Only API Commands described in section 3.3.2 do not enter API control mode

| - | | | | | | Byte 0 | Bytes 1 to n - 2 | Byte n - 1 | Byte n | - |
|---|---|---|---|---|---|---|---|---|---|---|
| S | Addr (0x50) - R/W bit low | A | S | Addr (0x50) - R/W bit high | A | Format Header | Payload. Hand data in format specified by header. Length will be 36 or 69 bytes for a total length (**n**) of 39 or 72 bytes. | Bitmask indicating finger enable/disable status | Checksum | P |

### 3.1.2 UART/RS485 Specification

The ability hand supports UART, and hardware versions 10.0 and greater support 5V level RS485 on pins 5-6 (SCL/SDA) of the main cable harness. The design of the protocol is for a multi-drop RS485 network, but it is compatible with direct UART (at appropriately reduced baud rates).

The extended interface for UART or RS485 can be entered by first configuring the hand for either extended mode 3.3V level UART or 5V level RS485 using the appropriate bluetooth controlled flash setting, then sending a properly formatted data frame using the selected protocol.

#### 3.1.2.1 Configuration

The settings used for UART and RS485 are configurable using flash settings. The following parameters can be modified and saved to the hands flash memory:

1. Slave address (default: 0x50)
2. Baud rate[5] (default: 460800)
3. Reply behavior[6] (default: standard mode)

When the hand is configured in UART API extended mode it will listen for transmission packets from a UART/RS485 master addressed to its slave address at the specified baud rate. The Ability Hand supports any 8-bit slave address except 0x00 and the following enumerated baud rates.

[5] Baud Rates above 460800 are only available for RS485, not UART.
[6] Multiple Reply Behaviors only available over UART. RS485 always uses standard behavior.

Table 7: UART/RS485 Enumerated Baud Rates

| **Enum Value** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|
| **Baud Rate** | 1200 | 2400 | 4800 | 9600 | 14400 | 19200 | 28800 | 31250 | 38400 |
| **Actual Baud**[7] | 1205 | 2396 | 4808 | 9598 | 14414 | 19208 | 28829 | 31250 | 38462 |
| | | | | | | | | | |
| **Enum Value** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** |
| **Baud Rate** | 56000 | 57600 | 76800 | 115200 | 230400 | 250000 | 460800 | 921600 | 1000000 |
| **Actual Baud** | 55944 | 57762 | 76923 | 115942 | 230400 | 250000 | 470588 | 941176 | 1000000 |

The baud rate and slave address can be modified via the Bluetooth Word Setting API_CONFIG0 (section 4.4.2). The transmission behavior can also be modified for increased bandwidth with Bluetooth Binary Setting 33 (section 4.4.3).

#### 3.1.2.2 Transmission Behavior and Format

The following table describes the general format of each UART transmission packet. The 0th byte is the hand address, the 1st byte is the format header. The next sequence of bytes corresponds to a unique command for each of the 6 motors in the hand, which are used based on the format header. The final byte is a checksum of all preceding bytes, including the address.

*Table 8: General format for transmissions (to the hand)*

| Byte 0 | Byte 1 | Bytes 2-13 | Byte 14 |
|---|---|---|---|
| Hand slave address (default 0x50) | Format header | Payload. One value per motor (6x total values), either voltage, position, velocity, or current (i.e. torque) request. | Checksum |

The general behavior of the hand in UART/RS485 mode is that it waits to receive a transmission frame addressed to it and then issues a response. The hand uses idle-line detection to group sequences of bytes. If the hand's UART RX line is left idle (high level) for 15 bits at the current baud rate, the frame is considered completed, and the hand will begin issuing its response. The next byte received after an idle event is considered to be the start (byte 0) of the next data frame. The UART on the hand is fully asynchronous; bytes can be received and transmitted at the same time.

[7] The Actual Baud Rate produced by the MCU on the Ability Hand.

#### 3.1.2.3 Standard Response Behavior and Format

The standard response behavior for UART and RS485 is that the hand will reply with a response frame immediately after a data frame with a matching slave address and checksum is received. In this mode, transmission messages can be a minimum of 3 bytes long (address, format header, and checksum). Any truncation of the message is allowed as long as the last byte in the frame has a properly computed checksum.

*Table 9: General format for receptions (from the hand)*

| Byte 0 | Bytes 1 to n - 2 | Byte n - 1 | Byte n |
|---|---|---|---|
| Format Header | Payload. Hand data in format specified by header. Length will be 36 or 69 bytes for a total length (n) of 39 or 72 bytes. | Bitmask indicating finger enable/disable status | Checksum |

#### 3.1.2.4 UART Fast Response Mode

Over UART, an additional fast mode is available. This mode can be enabled over bluetooth via a Binary Setting (see section 4.6.3). Fast Mode the hand will issue a reply[8] immediately after a matching address byte and valid format header is sent by the master[9]. This mode provides a simple way of increasing the bandwidth. Achieving the same bandwidth in standard mode is possible with precise timing, but more difficult to implement. Fast mode also decreases the probability that the hand will not issue a reply due to single event errors during the transmission of a large master data frame as a valid checksum is not required.

[8] In fast mode the format of the response will lag by one data frame. When the hand is initialized in fast mode the first response will always be in the default format (format 1).
[9] Only a matching address and format header are required to generate a response. However, the hand will still reject transmissions with an invalid data frame.

## 3.2 Temperature Protection

The API has built-in motor protection subroutines designed to protect the motors from commands that would otherwise damage them due to excessive heating. For instance, if API commands were accepted directly, a command such as 'move all fingers with max torque forever' would result in a sustained stall condition almost immediately. This protection is accomplished by leveraging the backdrivability of the fingers, estimating the temperature through heuristic functions, and indirectly measuring the motor temperature through ambient sensing inside the palm. The API indicates that it is not directly passing a user motion request value through the 'Overtemperature Limit Status' word in the API reply frame (3.4.7).

### 3.2.1 Halt-on-Collision

The Ability hand can detect collisions with objects through motor loading. This detection is performed by subtracting the back-emf voltage from the applied synchronous rotor voltage ($V_{stall}$). If $V_{stall}$ is small, the loading on the motor is small and it is considered 'not stalled'. If $V_{stall}$ is large, the motor is highly loaded and it is considered 'stalled'. A stalled condition in this context is considered to be equivalent to a 'collision'.

By default, a finger will be considered in a 'collision' state when $V_{stall}$ = 3.0V. When this occurs, motion in the direction the finger was moving when the stall was detected is prohibited until that finger receives a command to move in the opposite direction. This behavior is held for only digits that are non-backdrivable (index, middle, ring, pinky, thumb flexor), because non-backdrivability mechanically allows a static force to be maintained by the actuator without expending any electrical energy in the motor. Backdrivable actuators require electrical energy to maintain a static force, so this rule is not applicable to the thumb rotator.

A limitation of this feature is that if an object being manipulated by the hand shifts post-collision, the fingers cannot be commanded to close in on the object further until the finger first receives a command to move in the reverse direction. Additionally, because the fingers use worm gear drives, this method can be subject to 'false positive' collision detections post-stall, due to the fact that ratcheting action from gripping an object can cause high loading when moving in the opposite direction.

To overcome these limitations, two commands can be modified in the miscellaneous API section of the hand's nonvolatile storage. The first, WORD_API_POSTCONTACT_VOLTAGE (008) will change the voltage cap applied to motion after a collision is registered. By default this value is 0V (stopped). Increasing to 1-2V will allow a small amount of continuous current draw in the motor post-stall, which can prevent unwanted failure to move in the fingers.

The second, WORD_COLLISION_VOLTAGE (007) will directly set the $V_{stall}$ collision threshold. Increasing this value will increase grip strength of the Ability Hand when operating in API mode.

**Warning:** care should be taken when modifying miscellaneous words 007 and 008 from their default values, as secondary protection measures may begin to kick in if too much waste heat is allowed to build in the motor.

### 3.2.2 Temperature Heuristic Shutdown

The Ability Hand will use an internally calculated temperature heuristic as a secondary mechanism to monitor motor use and prevent overtemperature conditions. If the heuristic temperature exceeds a threshold value, the motor will be shut down until a sufficient cooling window has elapsed.

### 3.2.3 Ambient Temperature Shutdown

The Ability Hand uses a tertiary safety mechanism to monitor motor temperature indirectly with a physical measurement. Since there is no direct motor stator sensor available, the junction temperature of each motor controller microcontroller is used to estimate the ambient temperature of the palm. If the ambient temperature exceeds a factory-calibrated value, an overtemperature condition will be indicated with a 'triple beep' sound and shutdown. Motion will be re-enabled when the temperature has cooled to a lower, factory calibrated temperature, indicated with a 'long beep' sound.

## 3.3 Extended Mode Data Format Overview

### 3.3.1 Control Modes

There are four motor control modes available:

1. Position
2. Velocity
3. Torque ($I_q$)[10]
4. Voltage ($V_q$)

There are three response variants available that include different data combinations:

1. Finger position, current, touch sensors
2. Finger position, rotor velocity, touch sensor

[10] Torque Control Mode available from firmware version 2.3.

3. Finger position, current, rotor velocity

Additionally, a read only mode is available to receive data over UART/I2C. In this mode the hand is not controlled by UART/I2C and will respond to other control inputs such as Bluetooth or Analog Inputs. See section 3.3.3 for more details on read only mode.

Control modes are selected with a 'format header' in the transmit message, which is the first non-address byte. This byte determines both the format the hand will use to parse the data, and the content of the response it will issue. In the event that the hand is in API mode and receives a valid message (address & checksum) with a format header that does not specify a response mode[11], the hand will generate a response message with the last format header received. The default response format is variant 1. In addition to the control modes, there are miscellaneous options which can be enabled or disabled using the format header.

*Table 10: Extended API Control Mode Format Headers*

| Control Mode | Reply Mode | Format Header (Hexadecimal) |
|---|---|---|
| Position mode | Variant 1 | 0x10 |
| | Variant 2 | 0x11 |
| | Variant 3 | 0x12 |
| Velocity mode | Variant 1 | 0x20 |
| | Variant 2 | 0x21 |
| | Variant 3 | 0x22 |
| Torque mode | Variant 1 | 0x30 |
| | Variant 2 | 0x31 |
| | Variant 3 | 0x32 |
| Voltage mode | Variant 1 | 0x40 |
| | Variant 2 | 0x41 |
| | Variant 3 | 0x42 |

[11] This includes messages with format headers for miscellaneous commands, with unknown format headers, or with no format header. If this is the first API message sent the default response mode will be used

### 3.3.2 Read Only Mode

Read Only mode is available to get data from the hand without affecting the hand. Read Only commands do not trigger the hand to enter API control mode[12]. This allows data to be streamed from hand while it is controlled by another method, such as over Bluetooth or Analog Inputs.

| Mode | Format Header Value (Hexadecimal) |
|---|---|
| Read Only mode, reply variant 1 | 0xA0 |
| Read Only mode, reply variant 2 | 0xA1 |
| Read Only mode, reply variant 3 | 0xA2 |

### 3.3.3 Miscellaneous Format Header Commands

A number of miscellaneous commands are available to either trigger a one-time action or change volatile settings which reset on each power cycle. These commands trigger the hand to enter API mode and will sustain the hand in API control mode by resetting the 300ms timeout.

Miscellaneous messages can be sent without the data payload of command messages. For convenience, a data payload of up to 12 bytes can be included - this data will be ignored by the hand. The payload must have a valid checksum in the last byte.

[12] Note that a message with a Read Only format header also does not trigger the hand to exit API control mode. Read Only messages will sustain the hand in API control mode by resetting the 300ms timeout upon reception.

*Table 11: Miscellaneous Payload Format Headers*

| Description | Format Header Value (Hexadecimal) |
|---|---|
| Upsample Thumb Rotator[13] Enable | 0xC2 |
| Upsample Thumb Rotator Disable | 0xC3 |
| Exit API Control Mode[14] | 0x7C |

### 3.3.3 Extended Register Access

Access to various volatile settings for the Ability Hand is possible through the extended register access interface. The message format for this interface is as follows:

*Register Write*:

| Byte 0 | Byte 1 | Bytes 2-5 | Bytes 6-9 | Byte 10 |
|---|---|---|---|---|
| Hand slave address (default 0x50) | 0xDE | Address | Value | Checksum |

In response to a valid write request, the hand will send a four byte success or fail message.

| Bytes 0-3 |
|---|
| 0x4E11FEA4 (Success) or 0xFA17FFFF (Fail) |

*Register Read*:

| Byte 0 | Byte 1 | Bytes 2-5 | Byte 10 |
|---|---|---|---|
| Hand slave address (default 0x50) | 0xDA | Address | Checksum |

[13] Upsampling acts on incoming position requests for the thumb rotator. This feature is to be used if new positions are being requested infrequently (on the order of 30Hz) which can cause jerky motion due to the high link inertia and lower gear ratio of the thumb rotator joint.
[14] Immediately exit API mode, skipping the 300 millisecond timeout. This can be safely called when not in API mode without causing any unexpected movements

In response to a valid read request, the hand will send a message with the following format:

| Bytes 0-3 | Bytes 4-7 | Byte 8 |
|---|---|---|
| 0x4E11FEA4 (Success) or 0xFA17FFFF (Fail) | Register Value (Success) or 0x00000000 (fail) | Checksum |

*Notes on Position Control Settings*:
The position control gains for the Ability Hand are accessible in this interface. These position controllers are implemented in fixed-point. Generally speaking each gain value has an associated 'value' and 'radix', where the true value is determined by dividing the gain by two to the power of the radix value, or 1 left-shifted by the radix. So a value of '1000' with a radix of '4' would evaluate to a true gain of 62.5. Each motor has an associated Proportional, Integral, and Derivative gain and radix term. To mitigate the potential for overflow in a fixed-point integral calculation, each PID controller has an 'integral division' value which is applied at the computation of each integral summation step. The integral term also has an associated saturation value; i.e. the absolute value of the integral effort produced by the controller cannot exceed the saturation value. Finally, each controller has an overall 'radix' term, which is essentially division by the specified power of two applied to the entire controller.

Table of addresses:

| Address | Description | Read Only Status: |
|---|---|---|
| 0x0 | Reserved | n/a |
| 0x1 | Index Proportional fixed point gain value (voltage) | r/w |
| 0x2 | Index Proportional fixed point gain radix (voltage) | r/w |
| 0x3 | Index Integral fixed point gain value (voltage) | r/w |
| 0x4 | Index Integral fixed point gain radix (voltage) | r/w |
| 0x5 | Index Derivative fixed point gain value (voltage) | r/w |
| 0x6 | Index Derivative fixed point gain radix (voltage) | r/w |
| 0x7 | Index Integral division value (voltage) | r/w |

| | | |
|---|---|---|
| 0x8 | Index Integral saturation value (voltage) | r/w |
| 0x9 | Index output right-shift (voltage) | r/w |
| 0xa | Index output saturation (voltage) | r |
| 0xb | Middle Proportional fixed point gain value (voltage) | r/w |
| 0xc | Middle Proportional fixed point gain radix (voltage) | r/w |
| 0xd | Middle Integral fixed point gain value (voltage) | r/w |
| 0xe | Middle Integral fixed point gain radix (voltage) | r/w |
| 0xf | Middle Derivative fixed point gain value (voltage) | r/w |
| 0x10 | Middle Derivative fixed point gain radix (voltage) | r/w |
| 0x11 | Middle Integral division value (voltage) | r/w |
| 0x12 | Middle Integral saturation value (voltage) | r/w |
| 0x13 | Middle output right-shift (voltage) | r/w |
| 0x14 | Middle output saturation (voltage) | r |
| 0x15 | Ring Proportional fixed point gain value (voltage) | r/w |
| 0x16 | Ring Proportional fixed point gain radix (voltage) | r/w |
| 0x17 | Ring Integral fixed point gain value (voltage) | r/w |
| 0x18 | Ring Integral fixed point gain radix (voltage) | r/w |
| 0x19 | Ring Derivative fixed point gain value (voltage) | r/w |
| 0x1a | Ring Derivative fixed point gain radix (voltage) | r/w |
| 0x1b | Ring Integral division value (voltage) | r/w |
| 0x1c | Ring Integral saturation value (voltage) | r/w |
| 0x1d | Ring output right-shift (voltage) | r/w |
| 0x1e | Ring output saturation (voltage) | r |
| 0x1f | Pinky Proportional fixed point gain value (voltage) | r/w |

| | | |
|---|---|---|
| 0x20 | Pinky Proportional fixed point gain radix (voltage) | r/w |
| 0x21 | Pinky Integral fixed point gain value (voltage) | r/w |
| 0x22 | Pinky Integral fixed point gain radix (voltage) | r/w |
| 0x23 | Pinky Derivative fixed point gain value (voltage) | r/w |
| 0x24 | Pinky Derivative fixed point gain radix (voltage) | r/w |
| 0x25 | Pinky Integral division value (voltage) | r/w |
| 0x26 | Pinky Integral saturation value (voltage) | r/w |
| 0x27 | Pinky output right-shift (voltage) | r/w |
| 0x28 | Pinky output saturation (voltage) | r |
| 0x29 | Thumb Flexor Proportional fixed point gain value (voltage) | r/w |
| 0x2a | Thumb Flexor Proportional fixed point gain radix (voltage) | r/w |
| 0x2b | Thumb Flexor Integral fixed point gain value (voltage) | r/w |
| 0x2c | Thumb Flexor Integral fixed point gain radix (voltage) | r/w |
| 0x2d | Thumb Flexor Derivative fixed point gain value (voltage) | r/w |
| 0x2e | Thumb Flexor Derivative fixed point gain radix (voltage) | r/w |
| 0x2f | Thumb Flexor Integral division value (voltage) | r/w |
| 0x30 | Thumb Flexor Integral saturation value (voltage) | r/w |
| 0x31 | Thumb Flexor output right-shift (voltage) | r/w |
| 0x32 | Thumb Flexor output saturation (voltage) | r |
| 0x33 | Thumb Rotator Proportional fixed point gain value (voltage) | r/w |
| 0x34 | Thumb Rotator Proportional fixed point gain radix (voltage) | r/w |
| 0x35 | Thumb Rotator Integral fixed point gain value (voltage) | r/w |
| 0x36 | Thumb Rotator Integral fixed point gain radix (voltage) | r/w |
| 0x37 | Thumb Rotator Derivative fixed point gain value (voltage) | r/w |

| | | |
|---|---|---|
| 0x38 | Thumb Rotator Derivative fixed point gain radix (voltage) | r/w |
| 0x39 | Thumb Rotator Integral division value (voltage) | r/w |
| 0x3a | Thumb Rotator Integral saturation value (voltage) | r/w |
| 0x3b | Thumb Rotator output right-shift (voltage) | r/w |
| 0x3c | Thumb Rotator output saturation (voltage) | r |
| 0x3d | Unique ID Word 1 | r |
| 0x3e | Unique ID Word 2 | r |

## 3.4 Extended API Data Format

### 3.4.1 Position Data

Position is sent and received as a signed 16-bit integer (int16_t) in little endian format. The conversion ratio to represent finger positions (in degrees, with 0 degrees representing the fully extended position) is as follows:

$P_{digial} = \Theta_{degrees} * 32767 / 150$

Positive finger angles cause all joints but the thumb rotator to flex inwards towards the palm. The thumb rotator direction is reversed; negative angles from 0 cause it to flex away from its fully extended position. This relationship holds for velocity, voltage, and torque requests as well.

### 3.4.2 Finger Velocity Data To Hand

Velocity is sent as a signed 16-bit integer (int16_t) in little endian format. Velocities are represented in fixed point format with the following format:

$\omega_{digial} = \omega_{degrees/sec} * 32767 / 3000$

Positive velocities cause the fingers and thumb flexor to flex inwards toward the palm and negative voltages cause them to flex outwards toward the fully open positions. This is reversed for the thumb rotator.

In velocity control mode, the hand will track the requested velocity in finger degrees per second. Note that this differs from the velocity reported by the hand, which is the rotor velocity in radians per second. Therefore, the motor rotor velocity will differ by the gear ratio, units, and fixed point-conversion factor.

### 3.4.3 Rotor Velocity Data From Hand

Velocity data from the hand represents the motor rotor velocity in radians per second: The conversion is given below:

$\omega_{digital} = \omega_{rad/sec} * 4$

Note that the gear ratio of the finger must be used to convert this number into the rotational speed of the actual finger joint.

The gear ratios for each finger are provided in the table below:

| Finger | Gear Ratio |
|---|---|
| Index | 649:1 |
| Middle | 649:1 |
| Ring | 649:1 |
| Pinky | 649:1 |
| Thumb Flexor | 649:1 |
| Thumb Rotator | 162.45:1 |

### 3.4.4 Voltage Data

Voltage is only available as a control mode, the hand does not report voltage data. Voltage is sent as a signed 16-bit integer (int16_t) in little endian format. The requested ‘voltage’ is interpreted as a percentage of the power supply voltage, and can be thought of as a PWM

duty cycle. The range of input values is (-3546, 3546), where 3546 is 100% duty in the forward direction of the motor, and -3546 is 100% duty in the negative direction of the motor. Voltage/duties are enforced using the 'sinusoidal control' commutation method.

Note that, as with the other control modes, the thumb rotator will flex inwards when given a *negative* input, whereas the remaining joints will flex inwards when given a *positive* valued input.

### 3.4.5 Current (Torque) Data

Current values are sent as a signed 16-bit integer (int16_t) in little endian format. The requested value is interpreted as a motor rotor current request, in a fixed-point representation based on the current measurements taken with the analog instrumentation on the hand board. When this mode is active, the hand performs field-oriented control on each motor to track the requested rotor current. The rotor current returned in reply variants 1 and 3 is expressed in the same units.

The equation that relates fixed-point rotor currents to currents in amps depends on the specific hardware version of the hand. The conversion ratio C in the equation below can be requested from the hand over bluetooth, using the command ***Rv.*** See section 4.5 for more details.

$I_{digital} = I_{amps} * C$

The motor torque constant is provided below:

$k_{\tau} = 1.49$ mNm/A

The fingertip forces can be calculated using the above equations, the kinematic information provided in the universal robot description format (URDF) file included with the API, and the description of the 4 bar linkage parameters included elsewhere in this document.

Due to the extremely high bandwidth of FOC (30-40kHz) there will be effectively no difference between the requested motor current and measured rotor current, so it is recommended to use reply variant 2 or 3 when in this mode.

It is also important to note that there is passive, electromagnetic damping that occurs as a consequence of the motor dynamics when using voltage as a control input. This damping effect is removed as a consequence of active torque tracking, and as a result explicit damping is required in most cases to close a stable control loop on position when using torque as an input.

It is strongly recommended to use the rotor velocity term in variants 2 and 3 for damping, as this measurement will provide a substantially lower noise velocity estimate than a velocity measurement computed from the finger angles.

### 3.4.6 Touch Sensor Data

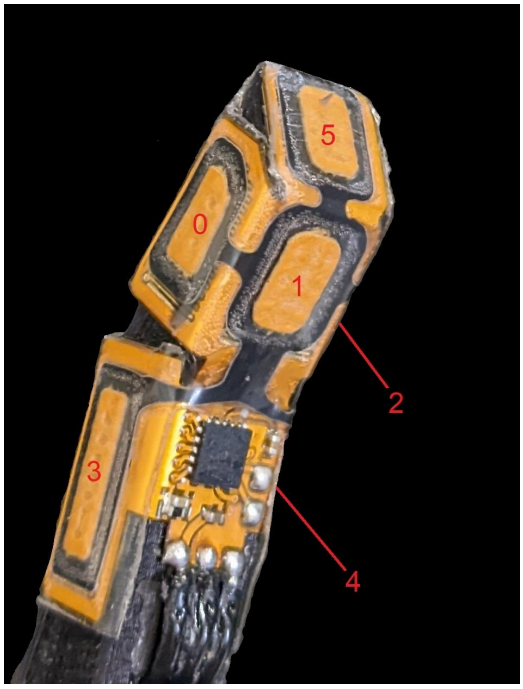
Touch Data is packed into 45 total bytes as 12 bit unsigned integers. For hands that do not have all touch sensors installed, a value of 0 will be sent in their place. Note that each word is broken up and packed with little-endianness. The endian-ness of the receiving system should be considered when unpacking this range of bytes.

The following C code will decode this range of bytes when run on a little-endian system.

```
/*
 *  Load packed 12 bit values located in an 8 bit array
 *  into an unpacked (zero padded) 16 bit array
 */
void unpack_8bit_into_12bit(uint8_t* arr, uint16_t* vals, int valsize)
{
	for(int i = 0; i < valsize; i++)
		vals[i] = 0; //Clear the buffer before loading with |=

	for(int bidx = valsize * 12 - 4; bidx >= 0; bidx -= 4)
	{
		int validx = bidx / 12;
		int arridx = bidx / 8;
		int shift_val = (bidx % 8);
		vals[validx] |= ((arr[arridx] >> shift_val) &0x0F) << (bidx % 12);
	}
}
```

The unpacked touch data corresponds to sites 0-6 per finger. I.e. the first 6 values are for the index, the next 6 are for the middle finger, and so on for the ring, pinky, and thumb. The site locations are detailed in the figure below:




The fingertip sensors are FSR based. To extract the FSR resistance from a raw FSR ADC value $D$, use the following equations:

1.) $V = D\frac{3.3}{4096}$

2.) $R = \frac{33000}{V} + 10000$

The conductance of an FSR is linearly dependent on the force exerted on the sensor. Therefore the force equation as a function of resistance is:

3.) $F_{sensor} = \frac{C_1}{R} + C_2$

Where $F_{sensor}$ is the force exerted on the sensor and R is the changing resistance of the sensitive element[15].

## 3.4.7 Overtemperature Limit Status

[15] PSYONIC has not performed an accurate calibration of the FSR sites at this time. Placeholder values C1 = 121591.0 and C2 = 0.878894 can be used to obtain a rough approximation of the force-resistance relationship. These constants provide a force estimate in Newtons.

The status of the torque limiting safety function is exposed through the api in the form of a bitmask, which indicates whether a motor is directly fulfilling the requested motion command. If some limitation is being applied, the corresponding bit will be set to '1'. If no limitation is active, the bit will be set to '0'. The bit mapping can be seen in table 12.

Table 12: Torque Limit Status Bitmap Indices

| Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 | Bit 7 |
|---|---|---|---|---|---|---|---|
| Index | Middle | Ring | Pinky | Thumb Flexor | Thumb Rotator | (Ignored) | (Ignored) |

### 3.4.8 Checksums

All transmissions to the hand must have a valid checksum in the final byte to be processed. All transmissions from the hand will include a checksum in the final byte which can be used to check data integrity. The checksum is the 2s complement negation of the (8 bit, 2s complement) sum of the preceding bytes. The 8 bit 2s complement sum all bytes will be equal to 0 for an uncompromised data frame.

## 3.5 Example Extended API Data Frames

### 3.5.1 Control Commands to the Hand

All command transmissions to the hand follow the same format regardless of control mode. Note that the Read Only commands may be sent in the format of a Control Command or a Miscellaneous Command.

*Table 13: I2C Control Command to Hand*

| Byte 0 | Bytes 1-2 | Bytes 3-4 | Bytes 5-6 |
|---|---|---|---|
| Command Header | Index Finger Data | Middle Finger Data | Ring Finger Data |
| | | | |
| Bytes 7-8 | Bytes 9-10 | Bytes 11-12 | Byte 13 |
| Pinky Finger Data | Thumb Flexor Data | Thumb Rotator Data | Checksum |

*Table 14: UART/RS485 Control Command to Hand*

| Byte 0 | Byte 1 | Bytes 2-3 | Bytes 4-5 | Bytes 6-7 |
|---|---|---|---|---|
| Hand Slave Address | Command Header | Index Finger Data | Middle Finger Data | Ring Finger Data |
| | | | | |
| Bytes 8-9 | Bytes 10-11 | Bytes 12-13 | | Byte 14 |
| Pinky Finger Data | Thumb Flexor Data | Thumb Rotator Data | | Checksum |

### 3.5.2 Miscellaneous Commands to Hand

Miscellaneous Commands are sent with just the command header and checksum.

*Table 15: I2C Miscellaneous Command to Hand*

| Byte 0 | Byte 1 |
|---|---|
| Command Header | Checksum |

*Table 16: UART/RS485 Miscellaneous Command to Hand*

| Byte 0 | Byte 1 | Byte 2 |
|---|---|---|
| Hand Slave Address | Command Header | Checksum |

### 3.5.3 Hand Reply Variants 1 & 2

Reply Variants 1 and 2 are identical except for the second piece of data. Variant 1 includes the Motor Current while variant 2 includes the Motor Rotor Velocity.

*Table 17: I2C, UART, RS485 Reply Variants 1 & 2*

| Byte 0 | Bytes 1-2 | Byte 3-4 | | Bytes 5-6 | Bytes 7-8 |
|---|---|---|---|---|---|
| Format Header | Index Position | Index Current / Rotor Velocity | | Middle Position | Middle Current / Rotor Velocity |
| | | | | | |
| Bytes 9-10 | Bytes 11-12 | Bytes 13-14 | | Bytes 15-16 | Bytes 17-18 |
| Ring Position | Ring Current / Rotor Velocity | Pinky Position | | Pinky Current / Rotor Velocity | Thumb Flexor Position |
| | | | | | |
| Byte 19-20 | Bytes 21-22 | Bytes 23-24 | | Byte 25 | Byte 26 |
| Thumb Flexor Current / Rotor Velocity | Thumb Rotator Position | Thumb Rotator Current / Rotor Velocity | | bits 11-4 of index site 0 touch sensor data | bits 3-0 of index site 0 & bits 11-8 of the index site 1 |
| | | | | | |
| Byte 27 | Byte 28 | ... | Byte 69 | Byte 70 | Byte 71 |
| bits 7-0 of index site 1 | bits 11-4 of index site 2 | ... | bits 7-0 of thumb site 5 | Hot/Cold status of each finger | Checksum |

### 3.5.4 Hand Reply Variant 3

Reply Variant 3 includes the finger position, motor current, and rotor velocity omitting the touch sensor data.

*Table 18: I2C, UART, RS485 Reply Variant 3*

| Byte 0 | Bytes 1-2 | Byte 3-4 | Bytes 5-6 | Bytes 7-8 | Bytes 9-10 | Bytes 11-12 | Bytes 13-14 |
|---|---|---|---|---|---|---|---|
| Format Header | Index Position | Index Current | Middle Position | Middle Current | Ring Position | Ring Current | Pinky Position |
| | | | | | | | |
| Bytes 15-16 | Bytes 17-18 | Byte 19-20 | Bytes 21-22 | Bytes 23-24 | Bytes 25-26 | | |
| Pinky Current | Thumb Flexor Position | Thumb Flexor Current | Thumb Rotator Position | Thumb Rotator Current | Index Rotor Velocity | | |
| | | | | | | | |
| Byte 27-28 | Byte 27-28 | Bytes 31-32 | Bytes 33-34 | Bytes 35-36 | Byte 37 | Byte 38 | |
| Middle Rotor Velocity | Middle Rotor Velocity | Pinky Rotor Velocity | Thumb Flexor Rotor Velocity | Thumb Rotator Rotor Velocity | Hot/Cold status | Checksum | |

### 3.5.5 PPP byte stuffing

The default Ability Hand extended protocol uses UART idle line detection to frame groups of bytes. A data frame is considered complete if the line is held high (idle) for more than the amount of time for a single byte to complete at the given baud rate. Therefore, to support the full protocol, both systems must be capable of precisely measuring/delaying code execution on the order of microseconds or less. A microsecond or sub-microsecond delay is not possible on any traditional operating system. Therefore if a traditional OS is used (any non real time Linux distribution, macOS, any Windows version), bandwidth is throttled (due to large minimum delay times) and code for receiving Ability Hand replies is complex and unreliable (prone to framing errors).

To circumvent this, enable PPP stuffing on the Ability Hand with toggles 46 and 47. Enabling PPP stuffing (toggle 46) byte-stuffs the outgoing frames from the Ability Hand, and enabling PPP unstuffing (toggle 47) causes the Ability Hand to unstuff incoming data. The protocol used is HDLC byte stuffing, consisting of a prepended and appended frame character (0x7E), and

escape character (0x7D). A byte matching either the frame or escape character is prepended with an escape character, followed by XOR'ing that byte with the mask 0x20. Unstuffing is the reciprocal process of stuffing.

Example code implementing stuffing/unstuffing in both C and python can be found in the following repository: https://github.com/psyonicinc/PPP-stuffing

# 4 Bluetooth Low Energy

## 4.1 Introduction

The Ability Hand has bluetooth low-energy connectivity which can be used to obtain various diagnostic information, control motion, and change important settings in the flash memory of the hand. Bluetooth connections use a GATT profile. Commands are generally sent from the BLE client (i.e. a phone or PC) to the BLE server (the Ability Hand) with write operations in the form of short ASCII encoded strings. The Ability hand sends data back to the BLE client in the form of notifications.

General message structure (for sending data *to* the Ability Hand) consists of a header byte, followed by a unique message structure. There are 10 different header bytes which subdivide the categories of operation for the Ability Hand bluetooth interface.

| | |
|---|---|
| Grip Command Header (Legacy) | G |
| Grip Configuration Header | g |
| (Flash Data Storage/FDS) Write Command Header | W |
| Read Command Header | R |
| Read Status Header | S |
| Data Plotting Header | P |
| Firmware Update Header | O |
| Miscellaneous Command Header | C |
| Grip Customization Header | m |
| Individual Finger Control Header | M |

## 4.2 General Connection Information

The GATT profile UUIDs are listed as follows:

| | |
|---|---|
| Service UUID | "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" |
| RX Characteristic UUID | "6E400002-B5A3-F393-E0A9-E50E24DCCA9E" |
| TX Characteristic UUID | "6E400003-B5A3-F393-E0A9-E50E24DCCA9E" |

Write data to the RX characteristic to send it to the Ability Hand.

Notifications must be enabled for the TX characteristic to receive data from the Ability Hand. The Ability Hand transmits all data as notifications.

## 4.3 Grip Commands

### 4.3.1 Grip Commands (Legacy) (‘G’)

This section describes how the app can move the hand to a limited number of grip configurations with a predefined speed. This command format is used by the demo mode screen in the PSYONIC App.

Set a grip: ***‘G--’***

- Byte 0: ***‘G’***. Prefix for this mode
- Byte 1: The grip index. ASCII format. List defined below:
    - ***‘0’***: Open the hand
    - ***‘1’***: Power
    - ***‘2’***: Key
    - ***‘3’***: Pinch
    - ***‘4’***: Chuck
    - ***‘5’***: \m/
    - ***‘6’***: Cylinder Grip
    - ***‘7’***: Mouse Grip
    - ***‘Z’***: Power/Key Mode switch
    - ***‘9’***: Point
    - ***‘A’***: Rude Point
    - ***‘8’***: Hook grasp
    - ***‘C’***: Relax
    - ***‘D’***: Sleeve Grasp
    - ***‘B’***: Peace Grasp
    - ***‘F’***: Chuck Grasp
    - ***‘E’***: Hang Loose Grasp
    - ***‘G’***: Handshake Grasp
    - ***‘H’***: Pinch (fixed thumb flexor location)
    - ***‘I’***: Unassigned
    - ***‘J’***: Unassigned
    - ***‘K’***: Unassigned
    - ***‘L’***: Unassigned
    - ***‘M’***: Unassigned
    - ***‘N’***: Trigger grip
    - ***‘O’***: Unassigned
    - ***‘P’***: Unassigned

        - ***'Q'***: Unassigned
        - ***'R'***: Unassigned
        - ***'S'***: Unassigned
        - ***'T'***: Unassigned
        - ***'U'***: Unassigned
        - ***'V'***: If enabled with binary setting, does 'waggle grip'. If not, loads an unassigned grip (no motion)
        - Any other value: Open the hand
    - Byte 2: The speed. RAW BINARY FORMAT. Can use ASCII commands; a value of ***'A'*** will move the hand with a calm, slow speed. Fastest human typable speed will use the character ***'~'.*** (It may be possible, depending on platform, to use extended ascii. ALT+254 is the second fastest, ALT+255 is the fastest).
    - For the PSYONIC App:
        - 0: stop the hand mid grasp
        - 1-254: proportional speed (maximum .3s)
        - 255: max speed (.2s)
    - Example command: ***'G3A'***. The hand will make a pinch grasp with low speed.

### 4.3.2 Grip Commands ('g')

Grips can also be requested using a command-line friendly/typable format. This command set indexes directly into the Ability Hand's grip configuration list (section 4.3.3) with a typed, user requested grip period. The structure of this command set is as follows:

- **gxx:xx.xx**

Where g is the header, bytes 1-2 are for the grip index, byte 3 is a colon separation character, and the remaining (arbitrary length) bytes are a typed floating point format number corresponding to the desired *period* in seconds that should elapse for each sub-motion of the selected grip.

Note that the index **-1** can be used as a shorthand to access the open configuration of whichever grip is currently active.

Examples:

- **g03:0.20** - execute a power grasp with a 0.2 second motion period
- **g-1:5.1234** - open the hand with a 5.1234 seconds motion period

### 4.3.3 Grip Configuration List

0 chuck grasp
1 chuck ok grasp
2 pinch grasp
3 power grasp
4 key grasp
5 handshake grasp
6 three finger trigger grasp
7 point grasp
8 relax grasp
9 sign of the horns grasp
10 rude point grasp
11 mode switch close grasp *(note: special grip which gets written to by application)*
12 Utility/Cylinder
13 Mouse
14 Hook
15 Sleeve
16 Peace
17 Hangloose
18 Pinch, thumb does not move
19 Free
20 Free
21 Free
22 Free
23 Free
24 Free
25 Free
26 Free
27 Free
28 Free
29 Free
30 Free
31 Waggle Grip, IF waggle is not disabled (We28 disables it)
32 - 63: **open configurations. Add 32 to the grip index below for the open config**

### 4.3.4 Grip Change CLI Specification (‘m’)

Before entering, select the grip you want to change using demo mode. You can modify both the grip and the open configuration of that grip.

- ‘***m***’ : enter grip modification mode.

Once in grip modification mode, you will see a status message that tells you you’re changing a configuration index. Indices 0-31 are close configurations and indices 32-63 are open configurations.

Select grip configuration idx:

- ‘***mg***--’ where ‘--’ is a typed, base 10 integer. Only integers between 0 and 63 are valid. This will load the grip configuration index to be modified. The grip configuration list is the same as for the non-legacy grip commands, detailed in section 3.3.2

Select finger:

- ‘***mf0***’ : select index finger
- ‘***mf1***’ : select middle finger
- ‘***mf2***’ : select ring finger
- ‘***mf3***’ : select pinky finger
- ‘***mf4***’ : select thumb flexor
- ‘***mf5***’ : select thumb rotator

Set selected finger setpoint:

- ‘***mq----***’ : where ---- is a typed, base 10 floating point number. Negative signs are interpreted and valid. Decimal points are interpreted and valid, although they are *not* required.
    - Example uses of this command:
        - ‘***mq30***’ : sets the current finger to 30.000 degrees
        - ‘***mq-55***’ : sets the current finger to -55.000 degrees
        - ‘***mq15.123***’ : sets the current finger to 15.123 degrees
        - ‘***mq-1.234***’ : sets the current finger to -1.234 degrees
    - Out of bounds value will be truncated to finger limits, the hand will return the value of the final position.

    - The thumb rotator can only move to negative values; all other fingers move only to positive values. All fingers can move to “0”.

Set the selected finger priority:

- ‘***mp--***’ : where -- is a *signed* (i.e. ‘***mp-1***’ is parsed as -1) base 10 integer which sets the priority of the given setpoint of the selected grip configuration. Priority determines the order in which the finger will come in. Priority 0 comes in first, until all priority 0 finger motions have finished, then priority 1, then priority 2, etc. Negative priority setpoints will never be loaded (i.e.that finger will not move or be changed).

View current finger position:

- ‘***mr***’ : print the current selected finger position to console in ascii-string format

View the current finger priority:

- ‘***mR”***’ : print the current selected finger priority to console in ascii-string format

Save the current configuration:

- ‘***ms***’ : the selected grip configuration will be saved (all setpoint and priority changes for the whole grip)

Exit grip modification mode:

- ‘***me***’: return to normal mode

## 4.4 Write Configuration CLI Specification (‘W’)

All commands in this section (prefix ***‘W’***) change filesystem data and save it. When using the CLI, a successful save will return with the string “success”. An unsuccessful save will return with “error: fs write”.

Electrode Flip Bit: ***‘WK0’ / ‘WK1’***

- Changes which electrode corresponds to what electrode connection on the coax plug. Software equivalent of swapping electrodes on the coaxial plug.

Key on Power Off Bit: ***‘WG0’ / ‘WG1’***

- When ‘0’ is written, the hand will move to a key grasp, initiated by power off. When ‘1’ is written, the hand will do nothing before power is lost. Legacy alias for MC_ENABLE_POWEROFF_GRIP_BIT

Write Hand Type: ***‘WH-’***

- Write the hand type. Not checked; if an out of bounds type is written, the ENTIRE filesystem will be restored to system default settings on the next power cycle.
    - ***‘WH0’*** : Long left hand
    - ***‘WH1’*** : Long right hand
    - ***‘WH2’*** : Short left hand
    - ***‘WH3’*** : Short right hand

Write Control Mode: ***‘W0-’***

- Change and save the new control mode.
    - ***‘W00’*** : Bluetooth control mode
    - ***‘W01’*** : Direct control mode
    - ***‘W02’*** : Linear Transducer control mode
    - ***‘W03’*** : I2C Top Rings control mode (aka Coapt Top Rings)
    - ***‘W04’*** : Single site control mode
    - ***‘W08’*** : I2C Mid Rings (aka Coapt Mid Rings)

Vibration Enable/Disable Bit ***‘W30’ / ‘W31’***

- ***‘W30’*** Enable vibration feedback
- ***‘W31’*** Disable vibration feedback

Vibration Intensity Bit ***‘WK--’***

- ***‘WK’*** header followed by raw binary value between 0-87. 87 is the maximum value parsed; it is internally thresholded by the hand.

Contact Reflex Enable/Disable Bit ***‘W50’ / ‘W51’***

- ***‘W50’*** Enable vibration feedback
- ***‘W51’*** Disable vibration feedback

Write Max Mappable Grip Period ***'WM[typed floating point number]'***

- Direct control (dual and single site) will proportionally map the grip period (in seconds) using this number. Grip period = time for the hand to transition between open and close configurations. Setting this number will determine the minimum amount of time the hand can possibly take to close using one of these modes. Numbers smaller than .2 will automatically be capped at .2 Numbers larger than about 5 seconds will seemingly result in direct control ceasing to function properly; use this feature carefully!
- Examples of valid commands:
    - ***'WM1.0'***
    - ***'WM.33'***
    - ***'WM0.4'***

Write Electrode 1 threshold ***'W6----'***

Write Electrode 2 threshold ***'W7----'***

Write Electrode 1 gain ***'W8----'***

Write Electrode 2 gain ***'W9----'***

- Prefix followed by a *4 byte floating point number*. These 4 bytes are *raw binary data*[16] in *32-bit ieee-754 floating point number (single precision)* format. Sets the corresponding threshold or gain.

Write Hand Name : ***'WJ--------'***

- Writes the variable portion of the bluetooth advertising name to the filesystem. Typed string format
- Example of use: ***'WJ20ABH025'***

Write Mode Switch Toggles: ***'WSxxxxxxx'***

- ***'WS'*** header followed by 7 digit data. Each can take ***'0'***, ***'1'***, or ***'x'*** (any character OTHER than ***'0'*** or ***'1'*** is equivalent to ***'x'*** ). As with other binary toggle settings, ***'0'*** is ENABLE and ***'1'*** is DISABLE.
- Data order:
    - Position 0: Open on Open (***'WS0xxxxxx'*** / ***'WS1xxxxxx'***)
    - Position 1: Double Impulse Close (***'WSx0xxxxx'*** / ***'WSx1xxxxx'***)
    - Position 2: Double Impulse Open (***'WSxx0xxxx'*** / ***'WSxx1xxxx'***)
    - Position 3: Co Contract (***'WSxxx0xxx'*** / ***'WSxxx1xxx'***)
    - Position 4: Long Co Contract (***'WSxxxx0xx'*** / ***'WSxxxx1xx'***)
    - Position 4: Long Open(***'WSxxxxx0x'*** / ***'WSxxxxx1x'***)
    - Position 4: Long Close (***'WSxxxxxx0'*** / ***'WSxxxxxx1'***)

Write Mode Switch Grips: ***'WTxxxxxx'***

[16] These bytes are not typable. They are raw binary data, not an interpreted string.

- ***'WT'*** header followed by a 6 character data field. Data is the exact same format as the ***'W2'*** command.
- Data Format (***x*** = data, ***-*** = placeholder character. Example to show position, NOT valid commands):
    - Position 0: Double Close Grip ***'WSx-----'***
    - Position 1: Double Open Grip ***'WS-x----'***
    - Position 2: Co-Contract Grip ***'WS--x---'***
    - Position 3: Long Co-Contract Grip ***'WS---x--'***
    - Position 4: Long Open Grip ***'WS----x-'***
    - Position 5: Long Close Grip ***'WS-----x'***

Enable Binary Setting ***'We[ARG]'***

Disable Binary Setting ***'Wd[ARG]'***

- ***[ARG]*** is a typed integer string between 0 and 63 corresponding to the index of a binary setting to change
- The enable command will set the binary setting to ***'0'***, enabling that setting.
- The disable command will set the binary setting to ***'1'***, disabling that setting.
- The complementary read setting for this command is ***'RZ'.*** See the command spec for ***'RZ'*** for the complete list of binary settings
- Example command: ***'We16'*** will enable UART/RS485 communication. ***'Wd16'*** will disable UART/RS485 (returning to I2C)
- See section 4.6.3 for full list of Binary Settings

Write Direct Control Timing Information ***'WU[0-3][0-255]'***

- This command writes timing settings for double impulse pulse time, rest time, long hold hold time, and long co contract hold time.
- The first argument ***[0-3]*** is a single digit character (***'0'***, ***'1'***, ***'2'***, or ***'3'***).
    - ***'0'***: writes the double impulse pulse time
    - ***'1'***: writes the long hold time
    - ***'2'***: writes the long co contract time
    - ***'3'***: writes the rest interval time
- The second argument ***[0-255]*** is a typed string integer ranging from 0 to 255. Each value corresponds to an increment of 15mS, starting at 50mS.
- Once written, the hand will automatically follow up with one of the following responses.
    - Double Impulse = [pulse time in ms]
    - Long Hold = [pulse time in ms]
    - Co Contract = [pulse time in ms]
    - Rest Interval = [pulse time in ms]

- Example command: ***'WU040'*** will set the double impulse pulse time to 650mS. The follow up string will read **'**Double Impulse = 650'.
- The complementary read command is ***'RX[0-3]'***

Factory Reset Non-Grip Settings: ***'Wo'***

- Special command (no data field) which restores non-grip settings to factory default.

Factory Reset Grip Settings: ***'Wj'***

- Special command (no data field) which restores grip settings to factory default.

Recalibrate Thumb Rotator Encoder

- On the next power-up, the thumb rotator joint will obtain a new encoder offset by stalling it to its fully extended position. Ensure the thumb has a clear path to extend without colliding with other objects before issuing this command + restart.

Write contact reflex grip period: ***'WWx.xxx'***

- String format to write the contact reflex period setting. The value written will be the period in seconds the hand will take to achieve its grip when the contact reflex is active. A period of 1000 will stop the hand. A period of .2 is the maximum possible speed for that grip.

Write Miscellaneous 32 bit word: ***'Wmxxx:xxxx'/'Wnxxx:xxxx'***

- General command format: Header ***(Wm/Wn)***, *exactly* 3 characters worth of address (valid inputs are 000-215), colon argument separator, followed by the typed numerical argument (which will be loaded into the specified word).
- ***'Wmxxx:xxxx'*** takes an argument string (after the colon) as a floating point string. An example: ***'Wm001:1.234'*** will set miscellaneous memory index 1 to 1.234f.
- ***'Wnxxx:xxxx'*** takes an argument string (after the colon) as a *hexadecimal* string. An example: ***'Wn002:0xDEADBEEF'*** will load the value 0xDEADBEEF into memory index 2.
- This feature is not intended to be used as general storage. Each word is potentially used to change the behavior of various features on the hand, changing currently unused words may result in undesired behavior when a future update utilizes this word. This command is created to be a 'general purpose' command which reduces the scope of individual command strings and thus overall developer effort. The list of words for this command can be found in section 4.6.
- IMPORTANT NOTE: the byte order for miscellaneous words is little-endian. For example, the correct command for setting the baud rate to 1Mbps while keeping the other API configuration settings as default is '**Wn001:0x00120000'**. For convenience, it is recommended to use the **Wp** command for individual byte setting.

Write individual byte to Miscellaneous word: ***'WpXXX:X:0xXX'***

- General command format: Header ***Wp***, *exactly* 3 characters worth of address (valid inputs are 000-215), colon argument separator, byte address (0,1,2 or 3), followed by a typed numerical argument (single byte hexadecimal value)
- Example: ***Wp002:1:0xFE*** writes the value 0xFE to byte index 1 of miscellaneous word 2.
- Intended to provide a simpler interface to the miscellaneous hexadecimal settings than the ***Wn*** command

## 4.5 Read Configuration CLI Specification (‘R’)

Commands in this section (with the prefix ***‘R’***) read various filesystem data and send them using the nRF BLE UART service.

| **Cmd** | **Name / Description** | **Output** |
|---|---|---|
| R1 | Control Input Type | • 0 : Bluetooth control mode<br>• 1 : Direct control mode<br>• 2 : Linear Transducer control mode<br>• 3 : I2C Top Rings control mode (aka Coapt Top Rings)<br>• 4 : Single site control mode<br>• 5 : Direct control mode (flipped electrodes)<br>• 6 : Linear Transducer Control mode (flipped electrodes)<br>• 7 : Single site control mode (flipped electrodes)<br>• 8 : I2C Mid Rings (aka Coapt Mid Rings) |
| R4 | Read Vibration Enabled | • 0: vibration is enabled.<br>• 1: vibration is disabled |
| RC | Read Hand Name | • Human readable hand name (string).<br>• Postamble portion of the bluetooth advertising name. |
| RD | Read Electrode 1 threshold | • *32 bit ieee 754 single-precision floating point format* electrode threshold/gain.<br>• Not human readable. |
| RE | Read Electrode 2 threshold | |
| RF | Read Electrode 1 gain | |
| RG | Read Electrode 2 gain | |
| RL | Read Electrode Flip Bit | • 0: electrodes are flipped. |

| | | |
|---|---|---|
| | | • 1: electrodes are not flipped |
| RO | Read Power Off Option | • 0 : hand will move to key when powered off (with a PSYONIC power switch and connected comm).<br>• 1: hand will do nothing when powered off |
| RS | Read Max Mappable Grip Period | Sends the header ***'PER'*** followed by an ASCII floating point string indicating the value of the mappable grip speed. Example: ***'PER1.234'***. |
| RV | Read Mode Switch Toggle | Sends the header ***'MOD:'*** followed by a 7 character string corresponding to the ***'0'*** or ***'1'*** ascii toggle settings sent with the complementary write command. The order is the same, stated as follows:<br>• Position 0: Open on Open<br>• Position 1: Double Impulse Close<br>• Position 2: Double Impulse Open<br>• Position 3: Co Contract<br>• Position 4: Long Co Contract<br>• Position 5: Long Open<br>• Position 6: Long Close |
| RW | Read Mode Switch Grips | Sends the bytes written to the mode switch grip fields. Does not contain a header. Byte order is as follows:<br>• Byte 0: Double Impulse Close Byte<br>• Byte 1: Double Impulse Open Grip<br>• Byte 2: Co Contract Grip<br>• Byte 3: Long Co Contract Grip<br>• Byte 4: Long Open Grip<br>• Byte 5: Long Close Grip |
| RX[0-3] | Read Timing Information | • Hand will reply with TIM[number] where the number is the time in ms. Will range from 50 to 3875.<br>• The argument (0-3) corresponds to the timing parameter index, as specified in the |

| | | |
|---|---|---|
| | | complementary write command, ***'WU[0-3][0-255]'***. |
| RZ[arg] | Read Binary Configuration Setting | • The ***[ARG]*** field is a typed integer string between 0-63, corresponding to the binary setting to be read.<br>• The hand will reply with the string: **BIN[ARG]=[VAL]**, where **[ARG]** is the setting number and **[VAL]** is the value of that setting (***'0'*** for enable and ***'1'*** for disable).<br>• Example: ***'RZ16'***. The hand will reply with ***'BIN16=0'*** if UART/RS485 mode is enabled.<br>• The complementary write commands for this command are ***'We[ARG]'*** and ***'Wd[ARG]'***, which enable or disable the setting of index ***[ARG]***, respectively.<br>• See section 4.6.3 for list of binary settings. |
| Ra | Read Contact Reflex Period | Returns human readable contact reflex grip period (period the hand has when reflex is activated) |
| Rc | Read hand version hash | Returns a 7-8 character hexadecimal hash corresponding to the specific firmware version number each device is running. |
| Rj | Read motor drivers version hash | |
| Rk | Read power switch version hash | |
| Rq | Read prepared motor driver version hash | Returns the version hash the current firmware has prepared to deploy to the motor drivers |
| Rmxxx | Read 32-bit Word Setting (Floating Point) | • General command format: Header ***(Rm/Rn)***, *exactly* 3 characters worth of address (valid inputs are 000-215).<br>• The hand will reply with the string: **mem[xxx]=val** where xxx is the word address and val is the value of that word in the specified format |
| Rnxxx | Read 32-bit Word Setting (hexadecimal integer) | |

| | | |
|---|---|---|
| Rtxxx:x | Read individual byte of a miscellaneous word setting | First argument is the word index. Argument after the colon is a single character (0,1,2 or 3), indicating the byte index of the word to read. Values outside of the range are modded with 4. Example: **Rt004:1** |
| Rv | Read the current conversion ratio | String format floating point conversion ratio that is used to convert the motor driver current value to Amps and vice versa |
| Ru | Perform encoder health check and read result | Performs a motor encoder health check (1-2 seconds) and reports the result. IMPORTANT: this check may return a false negative if performed with the fingers in a loaded configuration. Best results occur when the hand is in a vertical position, after re-zeroing. |
| RT | Read motor check result | Returns two values: the hexadecimal motor phase check status, recorded on motor powerup, and the current overall motor health status. Index aligned words of 6-12 characters. All pass result is: **abababababab** and **000000.** |
| R# | Read Unique ID | This command returns the Unique[17] ID from the microcontrollers in the hand. Each hand has 7 Unique IDs, 1 from the central Nordic microcontroller and 6 from the auxiliary STM microcontrollers. The response format is as follows:<br>- **nrf:[16 hexadecimal char nrf id]**<br>- **ID[#}:[24 hexadecimal char stm id]**<br>The ID[#] will be repeated for each STM, with # being a value 0-5. |

## 4.6 List of Configurable Settings

Only the settings listed here should be written and only in their specified format. Unlisted values may have internal uses and external writes to these can result in unexpected or undefined behavior.

[17] The ID on the central Nordic controller is not guaranteed to be unique, it is randomly generated. However, it is mathematically very unlikely that two hands have the same ID. The STM IDs are programmatically generated by ST and are guaranteed to be unique among parts of the same part number.

### 4.6.1 List of Word Settings: Floating point format

The table below shows all memory indices that are interpreted by the hand firmware as floating point values. Only these settings should be written to using the floating point variant of the write command. Note that all word settings share the same memory bank and indexing list. There is no protection from writing in the wrong format.

Table 19: Floating Point Format Bluetooth Word Settings

| **Memory Index:** | **Label:** | **Description:** |
|---|---|---|
| 000 | WORD_IDLE_TIMEOUT | Amount of time in seconds that must elapse with no EMG activity while the hand is open for the hand to enter a relaxed configuration |
| 007 | WORD_COLLISION_VOLTAGE | Voltage at which the motor must stall before a collision is considered to have occurred. Relevant to API mode only. 0 loads default setting (3.0V) |
| 008 | WORD_API_POSTCONTACT_VOLTAGE | Voltage threshold a motor is allowed post-collision, in the direction it was moving pre-collision. Default is 0 (i.e. motion stopped after collision). |
| 009-016 | Reserved | Reserved for internal use. Do not modify |
| 017 | Wakeup word (Index, Middle, Ring, Pinky) | Bytewise initialization word for each finger channel to indicate sensor type. Byte 0 is for index, byte 1 for middle, byte 2 for ring, byte 3 for pinky |
| 018 | Wakeup word (thumb flexor, thumb rotator) | Same as 017. Byte 0 is for thumb flexor, byte 1 is for thumb rotator. |

### 4.6.2 List of Word Settings: Hexadecimal format

The table below shows all memory indices that are interpreted by the hand firmware as floating point values. Only these settings should be written to using the hexadecimal variant of the write command.

Table 20: Hexadecimal Format Bluetooth Word Settings

| **Index:** | **Label:** | **Description:** |
|---|---|---|
| 001 | WORD_API_CONFIG0 | Uses byte-wise values to set baud rate, transmission frame format, and slave node ID for the UART/RS485 API protocol. **NOTE:** 0x00000000 will load *default settings*. Only change this word if you need to modify the hand's default settings.<br><br>**Byte 0:** Unused<br>**Byte 1:** Slave Address<br>**Byte 2:** Enumerated Baud Rate (see Section 3.1.2.1) |

| | | |
|---|---|---|
| | | **Byte 3:** Unused |
| 003 | WORD_HARDWARE_PROFILE | ENUM for Hand Hardware Version<br>**0:** Invalid<br>**1:** Pre-v10.x<br>**2:** v10<br><br>Changing the value of this word has no effect on hand operation. The contents of this word is overwritten on every power cycle. |
| 004 | WORD_USER_POWER_LIMIT | User Setting for Power Limiting the Ability Hand. For use cases where the hand is operating from a limited power source. This value is a percent out of 256 (0xFF). Valid inputs:<br><br>**0x00:** Disables user power limit - default<br><br>**0x20 - 0xFF:** Set user power limit 10% to 100%.<br>Values from 0x01 to 0x1F are interpreted as a 10% limit.<br>If a value higher than 0xFF is sent only the last byte is considered.<br><br>**Note that power limiting does not apply in API Control Mode* |
| 005 | WORD_HOT_THRESHOLDS_OFFSETS | **Byte 0:** Hot threshold for overtemp monitoring<br>**Byte 1:** Cold threshold for overtemp monitoring<br>**Bytes 2-3:** Calibration offset for index and middle in celsius. Do not modify |
| 006 | WORD_TEMP_OFFSETS_CTD | **Bytes 0-3:** Calibration offsets for ring, pinky, thumb flexor, thumb rotator |

## 4.6.3 List of Binary Settings

Table 21: Binary Bluetooth Settings

| **Setting Name** | **Index** | **Default** | **Description** |
|---|---|---|---|
| MC_VIBRATION_EN_BIT | 0 | 1 | Enable/disable vibration when touch sensors are pressed |
| MC_CONTACT_REFLEX_EN_BIT | 1 | 0 | Enable/disable contact reflex, triggered by touch sensors. |
| MC_NO_ADAPTER_ENCODER | 2 | 0 | Enable/disable check for adapter encoder. If enabled and an adapter encoder is detected, the adapter encoder is used to control that finger position. |
| MC_ENABLE_POWEROFF_GRIP_BIT | 3 | 0 | When enabled, the hand performs a power off grip upon reception of a signal to power off |
| *Reserved* | 4 | X | *Reserved* |

| | | | |
|---|---|---|---|
| MC_ELECTRODE_FLIP_BIT | 5 | 0 | When enabled, the open and close electrode input roles are swapped. |
| MC_IS_32_TO_1_BIT | 6 | 0 | Enable to flag the motor planetary gearboxes as 32.45:1. Default to 0 for safety, must be manually enabled for most hands |
| MC_ENABLE_OPEN_ON_OPEN_MS | 7 | 1 | Enable the open on open mode switch strategy for direct control. |
| MC_ENABLE_DI_CLOSE_MS | 8 | 0 | Enable the 'double impulse close' direct control mode switching strategy. |
| MC_ENABLE_DI_OPEN_MS | 9 | 0 | Enable the 'double impulse open' mode switching strategy |
| MC_ENABLE_CC_SHORT_MS | 10 | 0 | Enable the short co-contract signal for mode switching |
| MC_ENABLE_CC_LONG_MS | 11 | 0 | Enable the long co-contract mode switch signal |
| MC_ENABLE_LONG_OPEN | 12 | 0 | Enable long open mode switch strategy |
| MC_ENABLE_LONG_CLOSE | 13 | 0 | Long close mode switch strategy |
| MC_NO_FS_LOADING | 14 | 0 | If enabled, finger positions are not loaded from the filesystem on startup, even if saved positions are available |
| MC_DISABLE_AUTO_LP_MODE | 15 | 0 | If enabled, the I2C signal to bring the hand *out* of the low power state is ignored, thus placing the hand in a permanent low power state (unless overruled by the MC_FORCE_HIGH_POWER_MODE toggle) |
| MC_COMM_IS_UART | 16 | 0 | Enables the UART/RS485 peripheral on the COMM lines. I2C will no longer function. |
| MC_SLIP_DETECTION_EN | 17 | 0 | Enable slip detection feature on the hand. |
| MC_ENABLE_TAP_FREEZE | 18 | 0 | Enable IMU triple tapping for 'freeze' grip |
| MC_FIRST_OVER_CTL_ENABLE | 19 | 1 | Enables 'first over' strategy for direct control. |
| MC_ENABLE_TAP_MODESWITCH | 20 | 0 | Enable IMU double tapping to change grips (in control modes that use the |

| | | | |
|---|---|---|---|
| | | | griplist) |
| *Reserved* | 21 | X | *Reserved* |
| MC_DISABLE_DC_FILTER | 22 | 0 | When enabled, the low pass filter on direct control electrode signals is not used. |
| *Reserved* | 23 | X | *Reserved* |
| *Reserved* | 24 | X | *Reserved.* |
| MC_NO_TAP_MODESWITCH_WHILE_CLOSED | 25 | 0 | When enabled, mode switching via. IMU tapping is disabled when the hand is in a closed configuration. |
| MC_COAPT_FORCE_2INPUT_OC | 26 | 0 | When enabled, the hand attempts to prevent coapt from recognizing it by overloading the handshaking message on startup with 0 for 15 seconds. |
| MC_FORCE_HIGH_POWER_MODE | 27 | 0 | When enabled, all power related messaging is ignored and the hand will constantly be in the highest power state. |
| MC_DISABLE_FINGER_WAGGLE | 28 | 0 | When enabled, the 'finger waggle' grip behavior reverts to just another user-configurable, empty grip. |
| *Reserved* | 29 | X | *Reserved* |
| MC_ENABLE_ULTRAFAST | 30 | 0 | Experimental feature to allow increased max finger speed for smaller ranges of motion. |
| MC_ENABLE_PULLSTRING_LOCKOUT | 31 | 0 | Enables the lockout feature of pullstring/linear transducer control mode. |
| MC_ENABLE_IDLE_RELAX | 32 | 0 | When enabled, the hand will move to a relaxed configuration when open with no electrode activity for a specifiable amount of elapsed time. |
| MC_FAST_UART | 33 | 0 | Enable Fast Mode for UART, see section 3. |
| *Reserved* | 34 | X | *Reserved* |
| MC_RS485_ENABLED | 35 | 0 | Enables RS485 (requires v10 Hardware) |
| MC_RS485_MASTER_DEMO_ENABLED | 36 | 0 | Enables a demo where the hand acts as RS485 or UART master to another Ability |

| | | | |
|---|---|---|---|
| | | | Hand, causing a 'mirror' effect |
| MC_RGB_DISABLED | 37 | | Disables the RGB light on the PSYONIC power switch during main operation (not applied during startup) |
| *Reserved* | 38 | x | *Reserved* |
| MC_DISABLE_AIRTEMP_PROTECTION | 39 | 0 | Disables ambient overtemperature motion disable feature. WARNING: disabling this may result in thermal failure of the motors in the hand |
| MC_DISABLE_FAST_STARTUP | 40 | 0 | When set to 1, torque threshold for stall will be increased for startup motor re-zeroing |
| *Reserved* | 41-45 | X | *Reserved. Do not change* |
| MC_ENABLE_PPP_STUFFING | 46 | 0 | Enable PPP stuffing of extended mode hand reply frame (hand TX is byte-stuffed) |
| MC_ENABLE_PPP_UNSTUFFING | 47 | 0 | Enable PPP unstuffing of extended mode receive frame (hand unstuffs RX) |
| *Reserved* | 48-63 | X | *Reserved. Do not change* |

*__Note:__ Do not change reserved settings. These may be used internally for hand functionality.

__*Note__: there are 64 settings total. Settings not in this table can be modified with ***'We'/'Wd'***. Settings that are changed which are not in this table will have no effect at the time being, but may cause deviations from default behavior for future features.

# 4.7 Plotting Commands BLE Specification (‘P’)

The BLE Plotting Commands allow for high frequency data reporting over BLE. These commands are used by the plotting functionality of the Psyonic app. Plotting commands marked as ‘console plot’ can be send in the dev console or equivalent 3rd party app (such as nRF toolbox) and the resulting stream will be human readable text in such a console.

The other streaming formats are machine readable only. Due to the high rate of data being sent, these commands are not “command line friendly” and may overwhelm your terminal environment. These commands should be parsed by software such as the ‘plotting’ option in the PSYONIC app developer mode screen.

***‘P1’***: Disable current plot stream
***‘P0’***: Enable Touch Sensor Plotting. 6x IEEE 754 single precision (32 bit) floating point format.
***‘P2’***: Enable Direct Control Plotting. 6x IEEE 754 single precision (32 bit) floating point format.
***‘P3’***: Disable Direct Control Plotting
***‘P4’***: Enable FSR plotting

FSR plotting data format:

The data is sent in a 60 byte array of int16’s. The data should be subdivided into 5 groupings of 12 bytes each. The group index map is as follows: **index, middle, ring, pinky, thumb flexor.** Each group consists of 6x int16 values, corresponding to the sensitive sites on each finger.

**‘P41’**: FSR console plot

Prints the value of pressure sensor index 0 of each of the 5 possible sensor sites in one line as a string to the console (decimal format), followed by a bit-aligned hexadecimal status word corresponding to whether the checksum matched on each site. For clinical hands the final value should be 0x19, and for research hands it should be 0x1F.

**‘P5’**: Enable finger position plotting. 6x IEEE 754 single precision (32 bit) floating point format.
**‘P8’:** Enable finger velocity plotting. 6x IEEE 754 single precision (32 bit) floating point format.
**‘P9’:** Plot the hand’s IMU data. 7x IEEE 754 single precision (32 bit) floating point format

Value 0: Tap detection (hpf) value synthesized from IMU data
Values 1-3: Accelerometer data, raw, cast to floating point value
Values 4-6: Gyroscope data, raw, cast to floating point value

‘**PB’:** Plot the current grip percent. 2x IEEE 754 single precision (32 bit) floating point format

Value 0: close percentage (normalized 0-1)
Value 1: open percentage (equivalent to one minus the close percentage)

**‘PC’**: Plot the current finger position goal value. Relevant to clinical/grip control mode only; api value not reflected in this plot. 6x IEEE 754 single precision (32 bit) floating point format.
**‘PD’**: Plot the current progress (normalized 0-1) a given finger has progressed as a function of its starting and ending positions. 6x IEEE 754 single precision (32 bit) floating point format
**‘PE’**: Plot the motor current of each finger. 6x IEEE 754 single precision (32 bit) floating point format
**‘PF’**: Plot the adapter encoder position. 1x IEEE 754 single precision (32 bit) floating point format
**‘PG’**: Adapter encoder console plot

Prints the adapter encoder position as a string, followed by a checksum match/mismatch message

**‘PH0’**: Battery voltage plot. 1x IEEE 754 single precision (32 bit) floating point format
**‘PH1’**: Battery voltage console plot

Prints the hand battery voltage to console as a floating point string, in volts

**‘PI’**: Hand temperature console plot

Print the current internal junction temperature of each of the motor driver microcontrollers, in degrees celsius

## 4.8 Over The Air Updates (OTA) CLI Specification ('O')

The following section details how to OTA update the hand. All of these commands are blocking, the hand will not do anything else while they are running, and require a reboot upon completion regardless of outcome.

***'O0'***: OTA updates the central controller with a DFU .zip package.

1. The hand will play a song to indicate entry into OTA DFU mode.
2. The BLE advertising name will change to DFU targ.
3. Once entered, you can navigate to the DFU menu in the nRF Toolbox App
    a. Connect to the device named DFUTarg
    b. Select the DFU .zip file to update the hand with
    c. Tap 'upload'
4. The hand restarts upon successful completion of OTA DFU.

***'O1'***: Bootload the motor drivers with the firmware saved in Central Controller Memory.

- The binary data of the motor driver application code is saved in the application memory of the central motor controller. Sending this command will enter bootloading mode. The behavior of bootloading mode is detailed below, the bootloading procedure incorporates automatic retries on each step.
    - The hand will indicate the start of the bootloading procedure with the GATT notification: 'Start Driver Bootloading…'
    - For each driver the hand will provide the following GATT notifications:
        - 'starting driver #' when the bootloading for that driver begins
        - 'ch # responsive' if the driver has successfully responded to bootloader commands
        - 'driver # ok' if the driver bootloading is successful
        - 'driver # ERROR' if the driver bootloading has failed
    - When all 6 motor drivers have been flashed successfully, the notification "Bootload done OK" will be sent. The hand will then restart.
    - If any motor driver fails to bootload it can be reattempted after a power cycle. Drivers that are responsive but then fail are likely to succeed on a retry. If a driver repeatedly fails to bootload this indicates a hardware failure on the hand. The hand must be returned to PSYONIC for repair.
- If you receive the GATT notification 'Cannot Bootload: Unknown Hardware' verify that the value in the hexadecimal word setting 003 is correct (see section 4.6.2)

***'O2'***: Verify Driver Firmware

- Due to hardware limitations this command is available only on v10 hardware

- This command verifies the driver firmware by comparing the checksum of the memory on the driver with the known checksum saved on the central controller.
- On the reception of this command the hand will indicate that verification is starting with the GATT Notification: “Verifying Drivers…”
- For each driver the hand will send one of the following GATT Notifications:
    - ‘driver # ok’ if the verification is successful
    - ‘driver # error’ if the verification is unsuccessful
- If all drivers are successfully verified the hand will send the following GATT Notification and automatically restart: ‘All Drivers Verified OK’
- Checksum verification may be unsuccessful for any of the following reasons:
    - Communication Failure
    - Memory Corruption
    - Version Mismatch

***‘O3’***: Cherrypick Bootload Motor Drivers

- Due to hardware limitations this command is available only on v10 hardware
- This command is a combination of O1 and O2 commands above. The central controller will attempt to verify the firmware on each driver, if the verification fails for a driver it will then bootload that driver.
- On the reception of this command the hand will indicate that verification is starting with the GATT Notification: “Cherrypick Bootload…”
- For each driver the hand will send the following notification if the verification has been successful:
    - ‘driver # ok, not bootloading’
- If the verification is not successful, the central controller will attempt to bootload that driver. The hand will send the same “responsive”, “ok”, or “ERROR”GATT notifications as specified in command O1.
- If all drivers are successfully verified or successfully bootloader the hand will send the GATT Notification and automatically restart.
    - ‘All Drivers OK’

## 4.9 Miscellaneous/Custom Commands CLI Specification (‘C’)

***‘C0’***: Play the ‘DOOM’ theme song. \m/

***“CAxx”*** (BLE_SET_VIB): Override vibration motor intensity to input argument. Command expires after 10s and normal vibration control resumes. Input argument parsed as a base ten string number: example ***CA55***

***“CB”*** (BLE_CLEAR_VIB): Clear the vibration motor intensity. Will only affect one program loop of vibration commands mapped to touch sensor. Used to clear BLE_SET_VIB before the 10s expiry

***“CDUMP”*** (BLE_DUMP_FS): Dump the entire filesystem raw in multiple multi-byte chunks over Bluetooth. Format subject to change based on coded implementation of filesystem.

***“CR”*** (BLE_RESET_HAND): Calls NVIC_SystemReset() ARM function. **Not safe** due to STM32 SPI slave peripheral hardware implementation.

***“CNxxxxxx”*** (BLE_SHOW_RGB): Display an RGB value on both the hand board RGB led and the IO board RGB led. Used to preview different colors. Input format is a 32bit/4 character, base-16/hexadecimal string. Meant to make it simple to preview ‘html’ color codes: i.e. first byte is red, second byte is green, third is blue. Example: ***“CN00FE45”*** will display the io board’s best attempt at creating 0x00FE45, a light green color.

***“Crxx”*** (BLE_READ_EEPROM): Request a read of IO board EEPROM address. There are only 100 single byte eeprom addresses accessible through this command and its complement, BLE_WRITE_EEPROM. The post-header argument is a base-10 string from 0-99. Only 2 characters of the input string are used and the rest are discarded. Hand will reply with a notification containing the value in the argument address when the operation is complete, in the format `"%.2X%.2X"` (the requested address as a 2 character hex string followed by the requested data as a 2 character hex string). Example: “***Cr31”***

***“CExx:xx”*** (BLE_WRITE_EEPROM): Request a write to IO board EEPROM address. The address field is a 2 character base-10 string (i.e. only accessible EEPROM space are addresses 0-99). The data field is parsed as a base-10 string but does not have a character limit: however it is loaded into a uin8_t, so values over 255 WILL overflow. There is no input guarding on the hand side, so software must ensure that only values between “0” and “255” are sent. Example ***CE31:207***

## 4.10 BLE Individual Finger Control (‘M’)

The fingers positions can be updated individually using the following command structure:
**“F":** Header for individual finger position. The data following this header argument will be interpreted as position targets.

**“P”:** Header to configure individual finger period. The data after this header argument will be interpreted as period arguments.

| | | | |
|---|---|---|---|
| Byte 1: Command Header | Bytes 2-3: index data (little endian) | Bytes 4-5: middle data (little endian) | Bytes 6-7: ring data (little endian) |
| Bytes 8-9: pinky data (little endian) | Bytes 10-11: thumb flexor data (little endian) | Bytes 12-13: thumb rotator data (little endian) | |

Finger order is the same as in all other instances of ordered per-finger data (index,middle,ring,pinky,thumb flexor, thumb rotator).

The fixed point representations of positions and periods can be computed with the following scaling factors and castings, depicted in c code:

```
int16_t pos_16 = (int16_t)((position * (float)0x7FFF)/(150.f));
uint16_t per_16 = (uint16_t)((period * (float)0xFFFF)/(300.f));
```

# 4.11 IO board EEPROM Map

**Button Functionality:**

This section describes the options that can be assigned to single, double and triple tapping the io board button using the commands ***"Crxx"*** and ***"CExx:xx"***, which are described in the previous section.

*Addresses*:

| Address (hex): | Address (decimal): | Function |
|---|---|---|
| 0x0A | 10 | Single tap function definition |
| 0x0B | 11 | Double tap function definition |
| 0x0C | 12 | Triple tap function definition |

| Button Function Value: | Mapped behavior |
|---|---|
| 1 | Displays battery level |
| 2 | Sends the 'freeze' signal to the hand |
| 3 | Sends the 'mode switch' signal to the hand |

Example commands which restore default behavior map of the button taps:

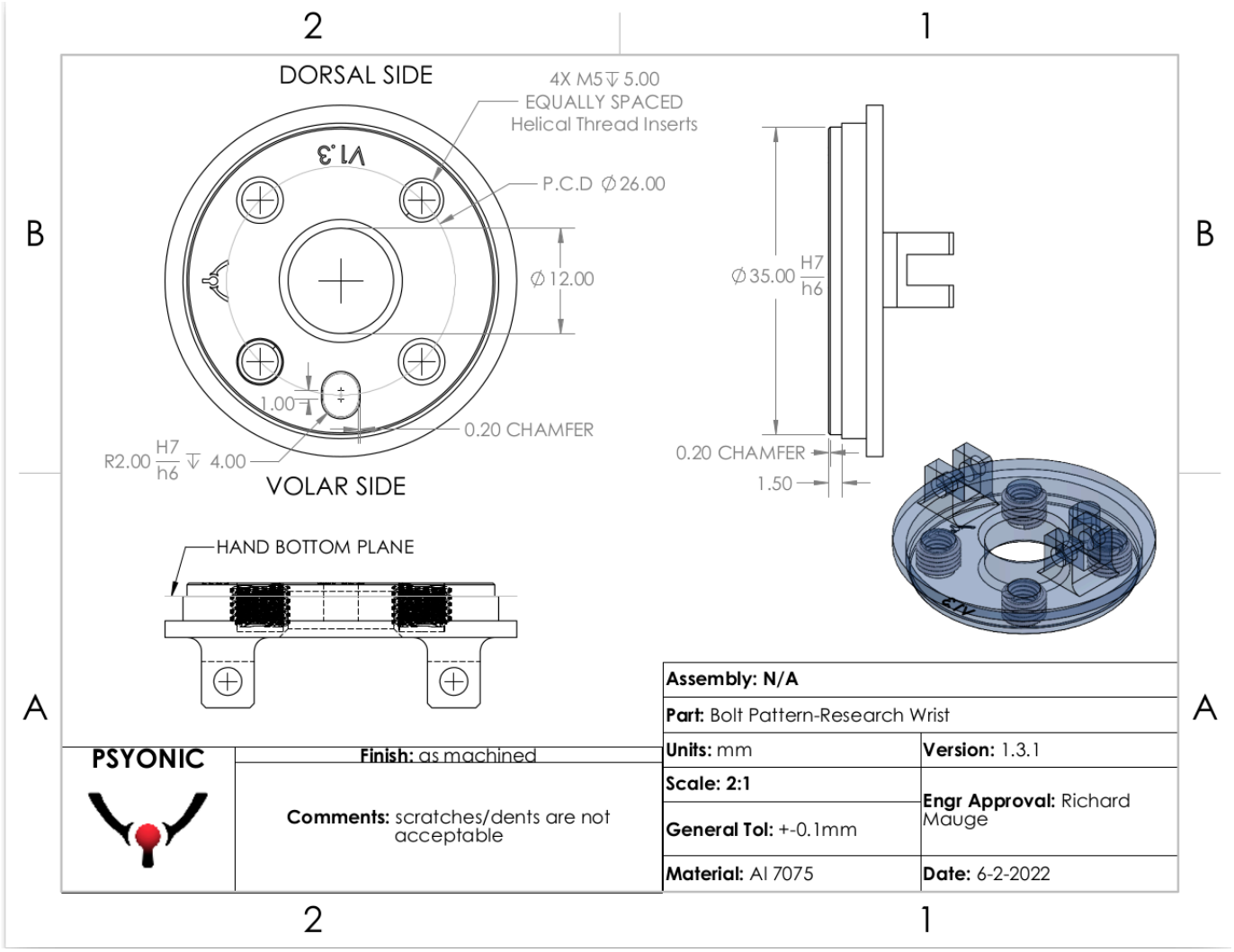
***CE10:01***

***CE11:03***

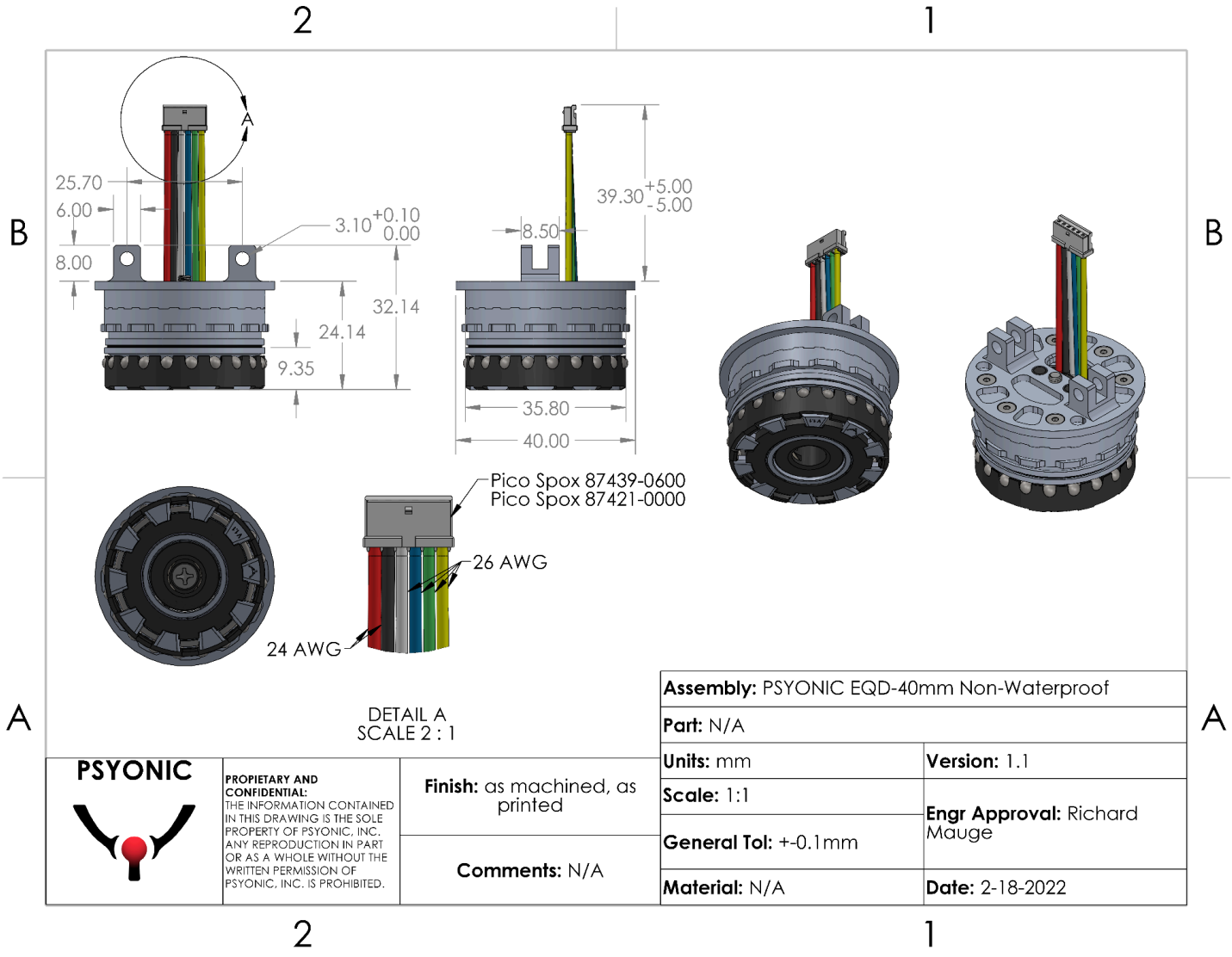
***CE12:02***

# 5 Mechanical Information

## 5.1 Bolt Pattern-Research Wrist



# 5.2 Psyonic 40mm-Non WP Electronic Quick Disconnect-Clinical Wrist



# 6 Document Control

Please contact PSYONIC for the latest version of this document.

| Version | Date | Changes |
|---|---|---|
| 1.0 | 3/25/2022 | Initial Release |
| 1.1 | 4/26/2022 | - Font/Structure change<br>- In Section 4, the hexadecimal write example erroneously used the “Wm” command instead of the correct “Wn” command<br>- New BLE OTA Commands: O2 (Verify) and O3 (Cherrypick)<br>- Updated BLE OTA Command O1 with new notification info<br>- Document the need for a power cycle on word 003 in section 4.6.2 |
| 1.2 | 6/06/2022 | - UART List Actual Baud Rates<br>- Fix Table of Contents Page Number Discrepancies<br>- Torque Control Availability Update<br>- Updated drawings and information for research wrist options |
| 1.3 | 6/07/2022 | - Correct Section 0 Configuration options to reflect research wrist options as changed in v1.2<br>- Minor wording fixes and clarifications |
| 1.4 | 8/06/2022 | - New word setting: WORD_USER_POWER_LIMIT<br>- Add some details to behavior and limitations of word settings<br>- Add new Bluetooth binary settings<br>- Document more Bluetooth Read commands and format section<br>- Updated Bolt Pattern-Research Wrist engineering drawing to more explicitly define the dowel pin clocking feature and remove irrelevant language.<br>- Add new bluetooth read command for unique ID<br>- Corrected errors in section number references |
| 1.5 | 8/11/2022 | - Corrected typos and other errors<br>- Clarification of v1.4 additions |
| 1.6 | 1/13/2023 | - Edited description of API safety rules to match new behavior<br>- Some typo corrections |
| 1.7 | 11/20/2023 | - Edited wrist drawing which had a mirror image mistake<br>- Updated toggles and added description and explanation of byte stuffing |
| 1.8 | 11/08/2024 | - Updated wording on Mk. 9.5 hand specs |

# 7 Contact PSYONIC

PSYONIC
9999 Businesspark STE B
San Diego, California 92131

1-888-PSYONIC (779-6642)  info@psyonic.io