Interface Control Document
RW-0.003 Tiny Reaction Wheels
Doug Sinclair, Cordell Grant, May 27, 2020
Rev 1.0

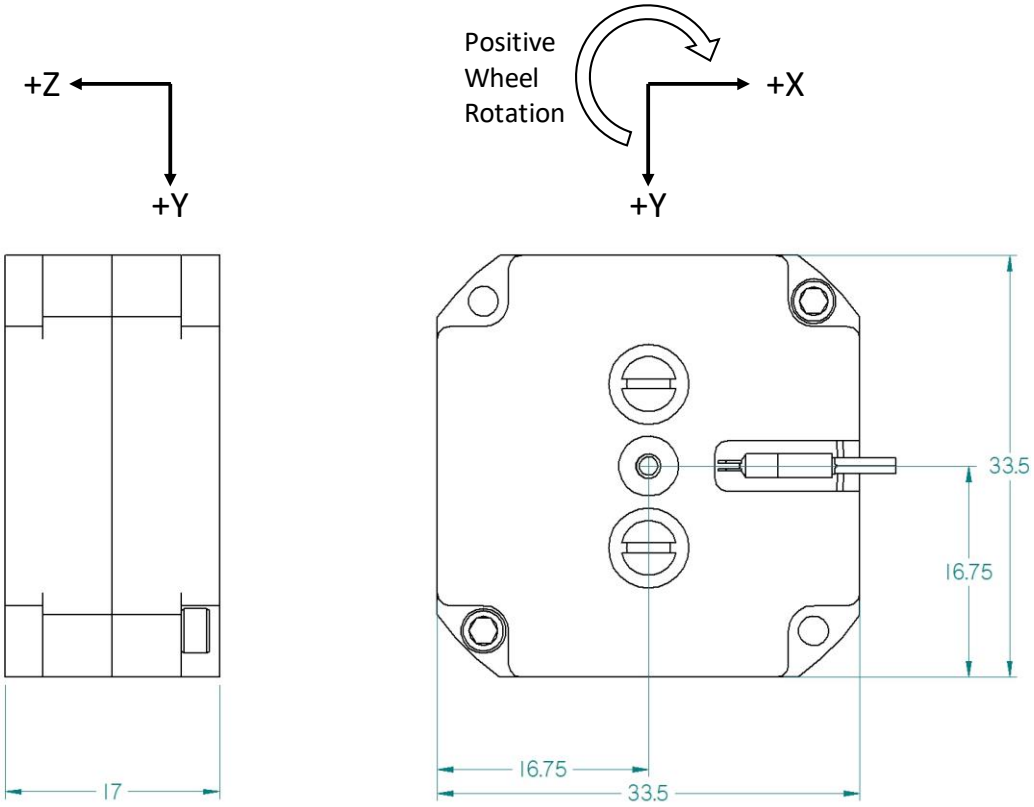
# 1 Scope

This document details the mechanical, electrical and software interfaces for the smallest Sinclair Interplanetary reaction wheels. At present these include:

- RW-0.003-8-I2C

# 2 Mechanical

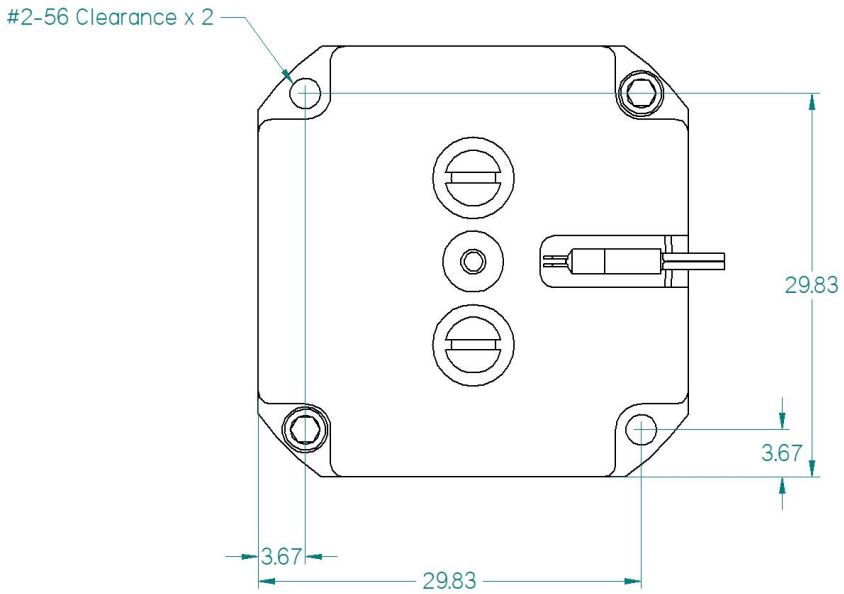
## *2.1 Overall Dimensions*



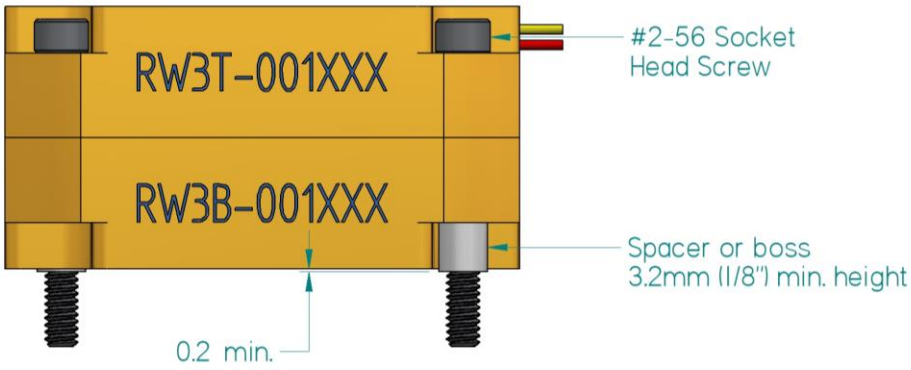
The axes for the reaction wheel are defined as shown in the above figure. The rotation arrow shows the direction of wheel rotation that is considered positive wheel speed. Rotation in the opposite direction is considered negative wheel speed.

## *2.2 Mounting Holes*

The wheel is mounted using the two unused corner holes as shown below. These holes are sized for #2-56 screws.




Of critical importance when mounting is that the wheel be supported beneath the two mounting points using a spacer or boss having a height of at least 3.2mm (1/8”). This ensures that the bottom of the wheel is not in direct contact with the spacecraft structure, which can cause warping of the wheel when the mounting screws are tightened. Such warping can impede proper function of the wheel.



## 2.3 Mass Properties

The mass of the complete wheel assembly is approximately 47.5g. The mass center is close to the geometric center.

## 2.4 Remove Before Flight

There are no necessary remove-before-flight elements. The bearings are covered with Kapton circles. These may be removed at the user's option, if the bearings are otherwise protected (such as mounting the wheel flush against a plate).

# 3 Environmental

## 3.1 Storage

The wheel must be stored in a clean environment to keep dust out of the bearings. The humidity must be kept low to prevent corrosion of the steel rotor. The wheel may be stored in a sealed bag with desiccant.

## 3.2 Thermal

**Table 1: Allowable Temperature Range**

| | |
|---|---|
| Survival Temperature | -40°C to +125°C |
| Operating Temperature (short term) | -40°C to +100°C at interface |
| Operating Temperature (long term) | -20°C to +70°C at interface |

Table 1 shows the allowed temperature range for the wheel. Short term operating temperatures are permitted for periods of hours to days, while long term operating temperatures are permitted for the many years of a mission.

## 3.3 Pressure

The wheel will operate in sea-level atmosphere and in hard vacuum. It has not been qualified to operate at high altitude atmospheres, and should not be powered during ascent unless additional testing is performed to show that there is no danger of arcing.

All materials meet the standard outgassing requirements of TML < 1%, CVCM < 0.1%.

## 3.4 Vibration

The wheel is designed to survive typical launch environments.
[Qualification environment TBD]

# 4 Electrical

## 4.1 Polarized Nano

The wheel is fitted with a 4 contact polarized nano connector. The pinouts for these connectors are liable to confusion. For clarity, use mating connector A79600-001 from Omnetics. The wire colours of that mating connector are:

**Table 2: Polarized Nano Connector**

| Wire | Name |
|---|---|
| Black | Power In/Out |
| Brown | Ground |
| Red | SDA |
| Orange | SCL |

### 4.2 Programming Header

There is a single plated through hole on the PCB that is accessed with a clip-lead during fabrication. It carries the signal "C2CK", and is used for initial software load.

# 5 Signals

### 5.1 Power In/Out

| | |
|---|---|
| Absolute Maximum | -0.3 V to +11.0 V |
| Operating Range | +5.0 V to +8.8 V |
| Front-end Capacitance | 220 nF |
| Total Capacitance | 43 uF |
| ESD Protection | Front end capacitor ESD qualified per HBM-AEC Q200-002. |

The Power In/Out signal is used to power the reaction wheel. Power normally flows into the wheel, but during regeneration it is possible for power to flow out.

At +8.82 V nominal the overvoltage protection feature will reset the wheel from application to bootloader software. See the fault protection section for more information.

### 5.2 Ground

The ground signal is the return for the power signal, and is the reference for all of the other signals. Ground is connected to chassis through a bleed resistor (TBD) and RF capacitor (TBD).

### 5.3 SDA, SCL

| | |
|---|---|
| Absolute Maximum | -0.5 V to +5.6 V |
| Input High Voltage | > 2.3 V |
| Input Low Voltage | < 1.0 V |
| Output Low Voltage | < 0.9 V @ 3 mA |
| Pullup Current | 330 uA @ 0 V |
| ESD Rating | IEC 61000-4-2 model<br>±30 kV contact discharge<br>±30 kV air discharge |

SDA and SCL comprise an $I^2C$ communications bus. The signals are not loaded if the wheel is powered down. The SCL signal is also used during factory programming.

## 5.4 C2CK

| | |
|---|---|
| Absolute Maximum | -0.5 V to +5.6 V when wheel is powered<br>-0.5 V to +2.5 V when wheel unpowered |
| Input High Voltage | > 2.2 V |
| Input Low Voltage | < 0.77 V |
| Pullup Current | 3.3 mA @ 0 V |
| ESD Rating | IEC 61000-4-2 model<br>±30 kV contact discharge<br>±30 kV air discharge |

C2CK is used during factory programming.

## 5.5 Power Architecture

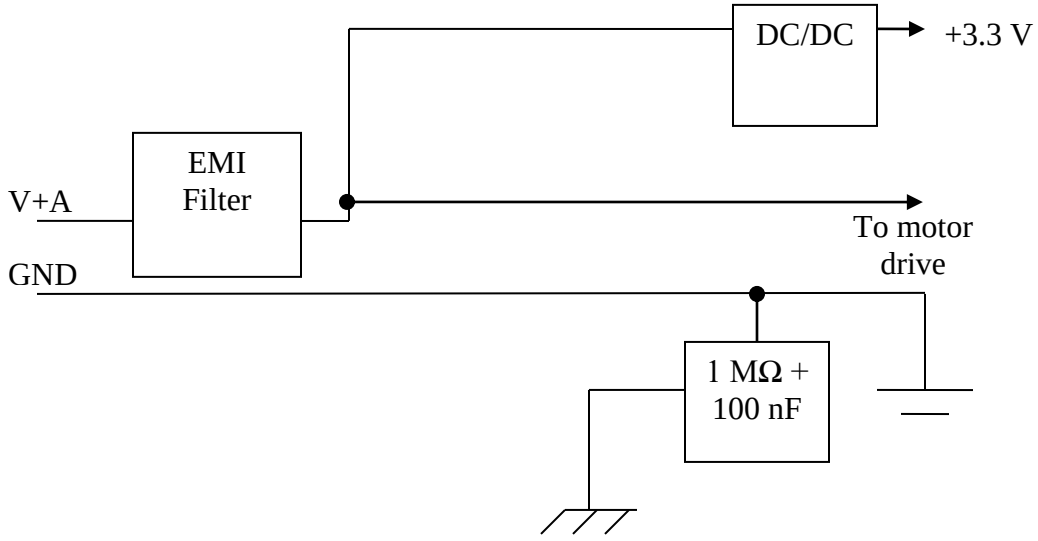



**Figure 1: Power Architecture**

The wheel has a simple power architecture with only one power input. All of the voltage rails are produced by DC/DC conversion, giving greater efficiency.

## 5.6 Regenerative Braking

The wheel makes use of regenerative braking when slowing the rotor under moderate torque. This will result in the wheel consuming a net negative amount of power, pushing current back out onto the spacecraft power bus. The spacecraft power system design must be able to deal with this.

In an emergency, if the power line becomes disconnected from the power system (such as if turned off via a relay switch) regeneration will increase the voltage at the wheel until the ~8.82 V safety threshold is reached. This will cause the wheel to reset and cease regeneration.

# 6 Protocol Layer 2 (Data Link Layer)

## 6.1 $I^2C$

The wheel uses a standard $I^2C$ communications bus. It acts as a bus slave, but will perform clock-stretching as needed.

If SCL is held low for 25 msec the interface will reset, as described in the SMBus specification.

## 6.2 Asynchronous Serial

Asynchronous serial is supported by the reaction wheel hardware, but not implemented by its software at this time. If used, the SDA signal becomes TX and the SCL signal becomes RX.

# 7 Protocol Layer 3 (Network Layer)

NSP is the Nanosatellite Protocol, originally developed at UTIAS/SFL for use on the CanX nanosatellites. This in turn is descended from the Simple Serial Protocol (SSP) used by UTIAS/SFL and Dynacon on the MOST and CHIPSAT spacecraft as well as the Dynacon reaction wheels in the wider market.

The reaction wheel uses NSP messages for all communication.

## 7.1 SLIP Encoding

NSP messages are encoded for transmission on asynchronous or $I^2C$ serial channels use SLIP framing, as described in RFC 105. This is required in order to indicate the beginning and end of NSP messages.

**Table 3: SLIP Framing Special Characters**

| | |
|---|---|
| FEND | 0xC0 |
| FESC | 0xDB |
| TFEND | 0xDC |
| TFSEC | 0xDD |

Whenever FEND would occur within the message it is replaced by two bytes: FESC TFEND. Whenever FESC would occur within the message it is replaced by FESC TFESC.

## 7.2 $I^2C$ Encapsulation

NSP messages are encapsulated for transmission over $I^2C$ links. The NSP address of a device is also used as its I2C address – note that I2C addresses are only 7 bits long, while NSP implemented on other links will accept up to 8 bits.

**Table 4: NSP Telecommand over I2C with no Reply**

| Transmitter | Data | Notes |
|---|---|---|
| OBC | START | |
| OBC | Slave address, Write | Slave ACK indicates slave receiving |
| OBC | Source address | |

| | | |
|---|---|---|
| OBC | Message Control Field | Poll bit = 0 |
| OBC | Outgoing data (0 – N bytes) | |
| OBC | 16-bit CRC | |
| OBC | FEND | |
| OBC | STOP | |

The table above shows the OBC (Onboard computer), which is also the bus master, sending a telecommand to a wheel where no reply is required. The message is SLIP framed, except that the leading FEND is omitted as it is redundant – $I^2C$ provides an out-of-band frame start signal.

The destination address is not transmitted. Instead, the transaction begins with the slave address and the write bit, in accordance with the $I^2C$ specification. Note that the CRC is computed based on the NSP message, not on the bytes transmitted over the I2C link.

**Table 5: NSP Telecommand over I2C with Reply**

| Transmitter | Data | Notes |
|---|---|---|
| OBC | START | |
| OBC | Slave address, Write | Slave ACK indicates slave receiving |
| OBC | Source address | |
| OBC | Message Control Field | Poll bit = 1 |
| OBC | Outgoing data (0 – N bytes) | |
| OBC | 16-bit CRC | |
| OBC | FEND | |
| OBC | START | |
| OBC | Slave address, Read | Slave ACK indicates slave receiving |
| Wheel | Message Control Field | Poll bit = 1 |
| Wheel | Reply data (0 – N bytes) | |
| Wheel | 16-bit CRC | |
| Wheel | FEND | OBC sends NACK to reclaim control of SDA |
| OBC | STOP | |

The table above shows the OBC sending a telecommand to a wheel where a reply is required. Instead of sending a STOP at the end of the telecommand, the OBC sends a repeated START. To comply with the $I^2C$ specification it must then re-send the slave address in read mode.

The slave then sends the reply message. This is SLIP framed, but as before the leading FEND is omitted. The destination and source address are also both omitted. An I2C transaction is atomic, so there can be no question of which telecommand produced which reply. Even though these fields are omitted, the CRC is still computed based on the entire NSP message in the normal fashion.

# 8 Protocol Layer 4 (Transport Layer)

## *8.1 NSP Message Format*

**Table 6: NSP Message Fields**

| Length | Field |
|---|---|

| | |
|---|---|
| 1 byte | Destination Address |
| 1 byte | Source Address |
| 1 byte | Message Control Field |
| 0 or more bytes | Data Field |
| 2 bytes | Message CRC |

Each NSP message has the format shown above. The shortest possible messages are 5 bytes (with zero data, not counting framing).

The wheel supports a maximum data length of 260 bytes, giving a total message length of 265 bytes. Note that network-layer framing may add additional bytes to the message as it is transmitted.

## 8.2 NSP Addresses

All NSP messages contain a destination and a source address. A reply message will be sent with a destination address equal to the source address of its command message. Similarly, the source address will be set equal to the destination address from the command.

The user is free to pick one or more NSP addresses for flight computers and other units that may talk to the wheel. Avoid choosing the SLIP framing characters FEND (0xC0) and FESC (0xDB), as well as the reserved address 0x00. By convention the flight computer would normally use NSP address 0x11.

The wheel pays no particular attention to the source address of commands, and will accept commands from any unit on the bus.

## 8.3 Message Control Field

**Table 7: Message Control Field**

| | |
|---|---|
| Bit 7 (MSB) | "Poll/Final" Bit |
| Bit 6 | "B" Bit |
| Bit 5 | "ACK" Bit |
| Bits 4 – 0 | Command code |

The message control field packs four values into a single byte. The command code is an enumerated value between 0x00 and 0x1F that determines how the data field should be interpreted.

The "ACK" bit is ignored on commands coming into the wheel. On telemetry reply messages sent by the wheel it is set to indicate successful execution of the command, or cleared to indicate that the command cannot be executed.

The "B" bit is copied unchanged from a command message into its reply message. The wheel does not use it internally.

The "Poll/Final" bit is interpreted differently for command and telemetry messages. For a command, the bit is "Poll". If it is set to '1' then the wheel will generate a telemetry message in reply. If it is cleared to '0' then the command will be executed, but no response telemetry message will be sent.

For a telemetry message, the bit is “Final”. If a reply consists of a single telemetry message, then the bit is set to ‘1’. If a reply is too large to fit into a single message then the final message has the bit set to ‘1’ and the others have the bit cleared to ‘0’.

### 8.4 Data Field

The interpretation of the data field is dependent on the command code in the message control field. Some command codes may have no data, some may require a certain fixed number of data bytes, and some can accept a variable data length.

### 8.5 Message CRC

Each NSP message contains a 2 byte (16-bit) CRC to guard against errors in transmission. The 16-bit CCITT polynomial is used: x^16 + x^12 + x^5 + 1. The initial shift register value is 0xFFFF. Bytes are fed into the CRC computation starting with the destination address, and concluding with the last byte of the data field. Within a byte, bits are fed in LSB first.

The following fragment of C code, courtesy of Henry Spencer, illustrates how the CRC can be computed.

```
#define POLY 0x8408 /* bits reversed for LSB-first */
unsigned short crc = 0xffff;
while (len-- > 0) {
      unsigned char ch = *bufp++;
      for (i = 0; i < 8; i++) {
            crc = (crc >> 1) ^ ( ((ch ^ crc) & 0x01) ? POLY : 0
            );
            ch >>= 1;
      }
}
```

### 8.6 Error Conditions

The wheel will ignore NSP command messages where the destination address does not correspond to its own NSP address. NSP messages with invalid CRC, invalid encapsulation, too short or too long are also ignored. In none of these cases will any reply message be generated.

If an NSP command message is in error due to an unknown command code, or if the data field is not consistent with the requirements of the command code, and if the “Poll” bit is set, then a NACK reply message will be generated. This message will be the same length as the command message, and contain the same data field. The command code will be the same, as will the “B” bit. The “ACK” bit will be cleared to ‘0’.

# 9 Protocol Layer 5 (Session Layer)

### 9.1 Operating Modes

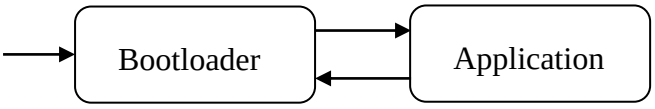



**Figure 2: Mode Transition Diagram**

Power-on starts the unit in bootloader mode.

### 9.1.1 Bootloader to Application Transition

The wheel will transition from bootloader to application mode upon receipt of an "INIT 0x00001000" command.

### 9.1.2 Application to Bootloader Transition

The wheel will transition from application mode to bootloader mode under the following conditions:

- An "INIT" command with no data is received.
- The Power In/Out pin exceeds ~8.82 V.

## *9.2 Test Scripts*

The reaction wheel contains a number of preprogrammed test scripts. These are used in the factory for initial characterization and pass/fail acceptance testing. They can also be used by customers to verify the health of the wheel during integration and on-orbit.

The exact contents of the test scripts is not documented here, to avoid the danger that it might become out of sync with the actual software. The *rw-bit-term* program should be used to record and interpret test script output. It is automatically synced to the wheel onboard software.

## *9.3 Byte Order*

All multi-byte values transported in the data field of NSP messages are in little-endian format. That is, the least-significant byte is stored first, and the most-significant byte is stored last.

## *9.4 Command Codes*

**Table 8: Command Codes**

| Command Code | Command | Bootloader | Application |
|---|---|---|---|
| 0x00 | PING | Yes | Yes |
| 0x01 | INIT | Yes | Yes |
| 0x02 | PEEK | Yes | Yes |
| 0x03 | POKE | Yes | Yes |
| 0x04 | DIAGNOSTIC | Yes | Yes |
| | | | |
| 0x06 | CRC | Yes | Yes |
| 0x07 | READ FILE | No | Yes |
| 0x08 | WRITE FILE | No | Yes |
| 0x09 | READ EDAC | No | Yes |

The table above shows the command codes that can be used by the host spacecraft to communicate with the wheel.

## *9.5 PING (0x00)*

The PING command is typically used during testing to verify communications. Incoming data is ignored. The reply packet contains a human-readable text string containing:

- The type of device and the manufacturer
- The name, and compile time and date of the software that is currently running on the target processor.

### 9.5.1 Command Format

| | |
|---|---|
| Bytes 0 – N | Zero or more bytes, ignored by the NSP module |

### 9.5.2 Reply Format

| | |
|---|---|
| Bytes 0 – N | Human-readable ASCII string. No NULL termination. |

## *9.6 INIT (0x01)*

The INIT command is used to change the operating mode of a wheel. In general, and INIT with data is interpreted as an address to jump to. An init with no data is interpreted as a reset or exit command. In all cases, if a reply has been requested ("Poll" bit set to '1') then the reply will be sent before the processor state is changed.

The wheel will respond to an INIT with no data by completely resetting the device, returning to bootloader mode. If it is in bootloader mode, it will respond to an INIT with 4 bytes of data by running an Application Module at the corresponding 32-bit start address. By convention, devices will ship from the factory with the supervisor processor primary application program stored at address 0x00001000. Thus, a command of INIT 0x00001000 will start the default behaviour.

### 9.6.1 Command Format

Reboot command:

| |
|---|
| No payload bytes |

Application start command:

| | |
|---|---|
| Bytes 0 – 3 | 32-bit integer address of program to start |

### 9.6.2 Reply Format

Reboot reply:

| |
|---|
| No payload bytes |

Application start reply:

| | |
|---|---|
| Bytes 0 – 3 | 32-bit integer address of program to be started |

## *9.7 PEEK (0x02)*

The PEEK command is used to read the device memory. The wheel processor has no restriction on the alignment or length of a peek.

### 9.7.1 Command Format

| | |
|---|---|
| Bytes 0 – 3 | 32-bit address to start peeking data |
| Byte 4 | Number of bytes to read. A value of 0 indicates that 256 bytes should be read. |

### 9.7.2 Reply Format

| | |
|---|---|
| Bytes 0 – 3 | 32-bit address of the start of data |
| Bytes 4 – N | One or more bytes read from the target memory |

## *9.8 POKE (0x03)*

The POKE command is used to write the device memory. The wheel will only permit a POKE into flash memory when in bootloader mode. Each 512 byte block of flash memory has a lifetime of only 20,000 write cycles. One cycle is consumed for each POKE command that accesses a particular block. This lifetime is more than sufficient for occasional software patches, but the user is cautioned that a looping sequence of POKE commands could easily wear out a block.

A single POKE command cannot span two flash blocks. This restriction comes from the SMBus 25 msec clock stretching limit – we cannot erase and program two flash blocks in this time.

### 9.8.1 Command Format

| | |
|---|---|
| Bytes 0 – 3 | 32-bit address to start poking data |
| Byte 4 – N | 1 - 256 bytes to write to the target memory |

### 9.8.2 Reply Format

| | |
|---|---|
| Bytes 0 – 3 | 32-bit address where data write began |
| Bytes 4 – N | 1 – 256 bytes written to the target memory |

## *9.9 DIAGNOSTIC (0x04)*

The DIAGNOSTIC command gathers error count data from the wheel.

### 9.9.1 Command Format

| | |
|---|---|
| Byte 0 | Address of the diagnostic channel to read, as an 8-bit integer |

### 9.9.2 Reply Format

| | |
|---|---|
| Byte 0 | Address of the diagnostic channel read, as an 8-bit integer |
| Bytes 1 - 4 | Diagnostic value from the addressed channel, as a 32-bit integer |

## *9.10 CRC (0x06)*

CRC command is used to calculate a checksum on an area of flash memory. The CRC uses the same 16-bit polynomial, with the same bit order, as is used for NSP messages.

The largest possible range is 0x0000 to 0xFBFF. This takes 17 msec to compute. Thus the computation is always compatible with the SMBus 25 msec clock stretch limit.

#### 9.10.1.1 Command Format

| | |
|---|---|
| Bytes 0 – 3 | Address of the first byte to CRC as 32-bit integer |
| Bytes 4 – 7 | Address of the last byte to CRC as 32-bit integer |

#### 9.10.1.2 Reply Format

| | |
|---|---|
| Bytes 0 – 3 | Address of the first byte in CRC as 32-bit integer |
| Bytes 4 – 7 | Address of the last byte in CRC as 32-bit integer |
| Bytes 8 – 9 | CRC result as 16-bit integer |

## *9.11 READ FILE (0x07)*

The Read File command returns one or more "files", which are four consecutive bytes of EDAC protected memory. A read from address 0 is a special case, and an additional mode byte is returned.

### 9.11.1 Command Format

| | |
|---|---|
| Bytes 0 | EDAC address divided by 4 (0 – 255). 0 for mode, 1 – 255 for normal. |

### 9.11.2 Reply Format

| | |
|---|---|
| Bytes 0-4 or 0-5 | File Reply structure. The first byte of the structure determines its type. |

#### 9.11.2.1 Mode Reply Structure

| | |
|---|---|
| Byte 0 | 0 |
| Byte 1 | Command type read from EDAC |
| Bytes 2 - 5 | Command value read from EDAC |

#### 9.11.2.2 Normal Reply Structure

| | |
|---|---|
| Byte 0 | Non-zero EDAC address divided by 4 (1 – 255) |
| Bytes 1 - 4 | EDAC data bytes read from memory |

## *9.12 WRITE FILE (0x08)*

The Write File command stores one "files", which are four consecutive bytes of EDAC protected memory. A write to address 0 is a special case, and an additional mode byte is stored.

If a Write File command fails due to improper formatting then no modification to EDAC memory is made.

#### 9.12.1 Command Format

| | |
|---|---|
| Bytes 0-N | One File Store structures. The first byte of each structure determines its type. |

##### 9.12.1.1 Mode Store Structure

| | |
|---|---|
| Byte 0 | 0 |
| Byte 1 | Command type to store |
| Bytes 2 - 5 | Command value to store |

##### 9.12.1.2 Normal Store Structure

| | |
|---|---|
| Byte 0 | Non-zero EDAC address divided by 4 (1 – 255) |
| Bytes 1 - 4 | Data bytes to write to EDAC memory |

#### 9.12.1 Reply Format

| | |
|---|---|
| Bytes 0-4 or 0-5 | File Reply structures. The first byte of each structure determines its type. |

##### 9.12.1.1 Mode Reply Structure

| | |
|---|---|
| Byte 0 | 0 |
| Byte 1 | Command type read from EDAC |
| Bytes 2 - 5 | Command value read from EDAC |

##### 9.12.1.2 Normal Reply Structure

| | |
|---|---|
| Byte 0 | Non-zero EDAC address divided by 4 (1 – 255) |
| Bytes 1 - 4 | EDAC data bytes read from memory |

### *9.13 READ EDAC (0x09)*

The Read EDAC command returns bytes from EDAC memory. The read process is atomic.

#### 9.13.1 Command Format

| | |
|---|---|
| Bytes 0 – 1 | EDAC address to start reading |
| Byte 2 | Number of bytes to read. A value of 0 indicates that 256 bytes should be read. |

#### 9.13.2 Reply Format

| | |
|---|---|
| Bytes 0 – 1 | EDAC address where reading started |
| Bytes 2 – N | The data bytes read from EDAC memory |

# 10 Protocol Layer 6 (Presentation Layer)

## 10.1 Fault State

Once in application mode, the reaction wheel may exist in or out of fault state. It starts out of fault state. Fault state is entered by the following conditions:

- Motor drive FET temperature exceeding ~160 °C.
- Motor drive current exceeding ~2 A.
- Rotor speed exceeding LIMIT_SPEED2

In the fault state the motor is not driven. Fault state is exited by the following condition:

- Command to IDLE

## 10.2 Memory Map

**Table 9: Supervisor Memory Map**

| Address Range | Function |
|---|---|
| 0x00000000 – 0x00000FFF | Bootloader program memory |
| 0x00001000 – 0x0000F3FF | Program memory (flash) |
| 0x0000F400 – 0x0000F7FF | Stored parameters (flash) |
| 0x0000FA00 – 0x0000FBFF | Bootloader program memory |
| 0x0000FC00 – 0x0000FFBF | Reserved area, access forbidden |
| 0x0000FFC0 – 0x0000FFCF | 128-bit UUID |
| 0x01000000 – 0x010000FF | 256 B IRAM (RAM) |
| 0x02000000 – 0x02000FFF | 4 kB XRAM (RAM) |
| 0x03000080 – 0x030000FF | 128 B SFR (RAM) Bank 00h |
| 0x03100080 – 0x031000FF | 128 B SFR (RAM) Bank 10h |
| 0x03200080 – 0x032000FF | 128 B SFR (RAM) Bank 20h |
| 0x03300080 – 0x033000FF | 128 B SFR (RAM) Bank 30h |

The supervisor memory can be directly accessed with PEEK and POKE commands, and CRCs calculated with CRC commands. It is represented as a single 32-bit memory space, sparsely populated.

The first 4 kB of program memory contain the bootloader. These are protected against POKEs so that the bootloader cannot be accidentally changed. The next 57 kB contains the supervisor application program. A sequence of POKE commands in bootloader mode can be used to load new application programs.

The bootloader memory cannot be read by the application program, and so PEEK or CRC commands to those regions will fail if not in bootloader mode.

The processor has two RAM areas. There is little need for a user to touch these.

There are four banks of Special Function Registers (SFRs). These should not be POKEd without knowing exactly what is going on. Even PEEKing some of these registers can have unexpected side effects.

## 10.3 Diagnostics

The diagnostics contain a series of read-only integers that relate to the health of the wheel.

**Table 10: Diagnostic Channels**

| Diagnostic Channel | Name |
|---|---|
| 0x00 | Reset Reason |
| 0x01 | Reset Count |
| 0x02 | Framing Errors |
| 0x03 | Runt Packets |
| 0x04 | Oversize Packets |
| 0x05 | Bad CRC |

### 10.3.1 Reset Reason

The reset reason is an enumerated type, describing the reason for the most recent reset of the wheel processor.

**Table 11: Reset Reason Codes**

| Reset Reason Code | Meaning |
|---|---|
| 0 | Power cycle. The wheel has either been freshly turned on, or the input voltage has drooped below the minimum operating voltage. |
| 1 | Realtime clock. This should not happen, as no realtime clock is fitted to this hardware. |
| 2 | Flash error. An illegal attempt has been made to read or write flash memory. |
| 3 | Comparator reset. This occurs if the Power In/Out pin exceeds 8.82 V. |
| 4 | Watchdog reset. The default application program does not use the watchdog timer, but if it somehow does get turned on this is the reset that it would generate. |
| 5 | Missing clock. The clock source for the processor stopped ticking. |
| 6 | Pin reset. The external C2CK signal has been pulled low. This should not happen to an integrated wheel. |
| 7 | Software reset. The most likely cause is that an INIT command has been received with no data, forcing a reset. |

### 10.3.2 Reset Count

The reset count contains the number of wheel processor resets since the last power cycle reset. Immediately after a power cycle the reset count will read as 0. After the first non-power-cycle reset it will read 1.

### 10.3.3 Framing Error

A framing error is declared if an NSP message is incorrectly encapsulated on the communications link. This would be any time a FESC character is seen that is not immediately followed by TFESC or TFEND.

### 10.3.4 Runt Packet

A runt packet is a NSP message that is less than 5 bytes long. Such a fragment cannot be a properly formed NSP message since it cannot contain a source and destination address, control field, and CRC.

### 10.3.5 Oversize Packet

An oversize packet is one that has too many bytes in the data field. Packets that are too long cannot fit into the allocated message buffers and so they must be rejected. See section 8.1 for the length constraints.

### 10.3.6 Bad CRC

This count is incremented every time a properly formatted (in length and framing) NSP message is received where the CRC field does not match with the computed CRC, and where the first byte is equal to the NSP address of the wheel.

## *10.4 EDAC Memory*

The wheel supports 1024 bytes of EDAC protected memory. These are implemented using software-based triple-redundant storage into conventional SRAM cells. EDAC memory can be read with READ EDAC and READ FILE commands, and written with WRITE FILE commands. The MODE_STORE command will save EDAC memory into non-volatile flash memory.

**Table 12: EDAC Memory Contents**

| EDAC Address | File Address | Function | Format |
|---|---|---|---|
| 0x000 – 0x003 | 0x00 | Command Value | Command Dependent |
| 0x004 – 0x007 | 0x01 | GROUND | Volts (IEEE-754 float) |
| 0x008 – 0x00B | 0x02 | VDD | Volts (IEEE-754 float) |
| 0x00C – 0x00F | 0x03 | TEMPERATURE | °C (IEEE-754 float) |
| 0x010 – 0x013 | 0x04 | LDO | Volts (IEEE-754 float) |
| 0x014 – 0x017 | 0x05 | VSENSE | Volts (IEEE-754 float) |
| 0x054 – 0x057 | 0x15 | SPEED | Rad/sec (IEEE-754 float) |
| 0x058 – 0x05B | 0x16 | MOMENTUM | N-m/sec (IEEE-754 float) |
| 0x060 – 0x063 | 0x18 | SEU_COUNT | counts (IEEE-754 float) |
| 0x064 – 0x067 | 0x19 | FAULT_STATE | Enum in IEEE-754 float |
| 0x06C – 0x06F | 0x1B | HALL_DIGITAL | Binary value in IEEE-754 float |

| | | | |
|---|---|---|---|
| 0x070 – 0x073 | 0x1C | CONTROL_TIME | Timer ticks (IEEE-754 float) |
| 0x080 – 0x083 | 0x20 | SPEED_P_GAIN | Amps / rad / sec (IEEE-754 float) |
| 0x084 – 0x087 | 0x21 | SPEED_I_GAIN | Amps / rad (IEEE-754 float) |
| 0x088 – 0x08B | 0x22 | SPEED_D_GAIN | Amps / rad / $sec^2$ (IEEE-754 float) |
| 0x094 – 0x097 | 0x25 | MAX_GAIN_SPEED | Rad/sec (IEEE-754 float) |
| 0x098 – 0x09B | 0x26 | MIN_GAIN_SPEED | Rad/sec (IEEE-754 float) |
| 0x0A0 – 0x0A3 | 0x28 | INERTIA | $kg\text{-}m^2$ (IEEE-754 float) |
| 0x0A8 – 0x0AB | 0x2A | GAIN_SCHEDULE1 | (IEEE-754 float) |
| 0x0AC – 0x0AF | 0x2B | GAIN_SCHEDULE2 | (IEEE-754 float) |
| 0x0B0 – 0x0B3 | 0x2C | GAIN_SCHEDULE3 | (IEEE-754 float) |
| 0x0B4 – 0x0B7 | 0x2D | GAIN_SCHEDULE4 | (IEEE-754 float) |
| 0x0BC – 0x0BF | 0x2F | CONTROL_TYPE | (IEEE-754 float) |
| 0x0C8 – 0x0CB | 0x32 | MAX_SPEED_AGE | sec (IEEE-754) |
| 0x0CC – 0x0CF | 0x33 | LIMIT_SPEED1 | Rad/sec (IEEE-754) |
| 0x0D0 – 0x0D3 | 0x34 | LIMIT_SPEED2 | Rad/sec (IEEE-754) |
| 0x0D4 – 0x0D7 | 0x35 | LIMIT_VOLTAGE | Volts (IEEE-754) |
| 0x100 – 0x103 | 0x40 | PREVIOUS_SPEED | Rad/sec (IEEE-754) |
| 0x104 – 0x107 | 0x41 | SPEED_INTEGRATOR | Amps (IEEE-754) |
| 0x108 – 0x10B | 0x42 | SPEED_LAST_ERROR | Rad/sec (IEEE-754) |
| 0x10C – 0x10F | 0x43 | ACCEL_TARGET | Rad/sec (IEEE-754) |
| 0x110 – 0x113 | 0x44 | TEST_VOLTAGE | Volts (IEEE-754) |
| 0x12C – 0x12F | 0x4B | TORQUE_T0 | Nm (IEEE-754) |
| 0x130 – 0x133 | 0x4C | TORQUE_T1 | Nm (IEEE-754) |
| 0x134 – 0x137 | 0x4D | TORQUE_T2 | Nm (IEEE-754) |
| 0x138 – 0x13B | 0x4E | TORQUE_T3 | Nm (IEEE-754) |
| 0x13C – 0x13F | 0x4F | TORQUE_T4 | Nm (IEEE-754) |
| 0x140 – 0x143 | 0x50 | VALUE_MONITOR | Varies (IEEE-754) |
| 0x144 – 0x147 | 0x51 | SFFT_STEP_TIMER | sec (IEEE-754) |
| 0x3F8 | | MODE | 8-bit enum |
| 0x3F9 | | HALL_IMPOSSIBLE | 8-bit unsigned int |
| 0x3FA | | HALL_SKIP | 8-bit unsigned int |
| 0x3FB | | CONTROL_OVERFLOW | 8-bit unsigned int |
| 0x3FC | | SFFT_STEP_NUMBER | 8-bit unsigned int |
| 0x3FD | | SFFT_TELEM_COUNT | 8-bit unsigned int |
| 0x3FE | | MODE_MONITOR | 8-bit enum |
| 0x3FF | | FRICTION_DONE | 8-bit boolean |

### 10.4.1 Command Value

Accessing file 0 causes an extra mode byte to be transferred. By writing to this file the mode of the wheel can be commanded. By reading this file the current mode can be determined. The modes are enumerated in section 10.4.32.

If this parameter is accessed through EDAC writes and reads instead of file reads and writes there is no explicit mode byte transferred. It is possible to read and write the number associated with the command, but this is not advised.

### 10.4.2 GROUND

The voltage sensed by the wheel's ADC when looking at a grounded input. Should be very close to zero. This is of very little interest outside of anomaly investigation.

### 10.4.3 VDD

The output of the wheel's +3.3 V DC/DC converter.

### 10.4.4 TEMPERATURE

The on-die temperature of the wheel's processor. This is factory calibrated, and should be accurate to within a few degrees.

### 10.4.5 LDO

The output of the wheel's +1.8 V Low Dropout regulator.

### 10.4.6 VSENSE

The voltage on the Power In/Out pin.

### 10.4.7 SPEED

This read-only parameter returns the speed of the rotor.

### 10.4.8 MOMENTUM

This read-only parameter returns the angular momentum of the rotor. It is derived from the SPEED multiplied by INERTIA.

### 10.4.9 SEU_COUNT

This parameter records the number of errors that have been found during EDAC scrubbing. Any error in a byte is considered to be a single error – no attempt is made to determine how many bits were flipped.

This parameter can be read to determine the error count. It can also be written – typically to reset it to zero.

### 10.4.10 FAULT_STATE

This is 1.0 if the wheel is in a fault state, or 0.0 otherwise.

### 10.4.11 HALL_DIGITAL

The wheel has three Hall-effect sensors (numbered 0 to 2), each one generating a binary value. The value of these three bits is encoded into this parameter.

| Hall 2 | Hall 1 | Hall 0 | HALL_DIGITAL |
|---|---|---|---|
| 0 | 0 | 0 | 0.0 |
| 0 | 0 | 1 | 1.0 |
| 0 | 1 | 0 | 2.0 |
| 0 | 1 | 1 | 3.0 |
| 1 | 0 | 0 | 4.0 |
| 1 | 0 | 1 | 5.0 |
| 1 | 1 | 0 | 6.0 |
| 1 | 1 | 1 | 7.0 |

### 10.4.12 CONTROL_TIME

This parameter not currently fully functional.

See the CONTROL_OVERFLOW file for indication of negative realtime margin.

### 10.4.13 SPEED_[P|I|D]_GAIN

These parameters set the gains for the PID closed-loop speed controller. See CONTROL_TYPE for more information on when these are read-write parameters set by the user, and when they are read-only and internally generated.

### 10.4.14 MIN_GAIN_SPEED, MAX_GAIN_SPEED

These read/write parameters bound the speed used as an input to the speed controller gain formula.

By setting these two parameters to the same value the speed dependence of the gains can be effectively disabled.

### 10.4.15 INERTIA

This read/write parameter sets the rotor inertia. It is used to scale between acceleration and torque, and momentum and speed.

### 10.4.16 GAIN_SCHEDULE[1..4]

These four read/write parameters are used to set the speed control gains, in those cases when PROPORTIONAL_OVERRIDE is zero. First, the characteristic speed ω is determined based on the actual and setpoint speeds and on MAX_GAIN_SPEED and MIN_GAIN_SPEED.

$$\omega = MIN\left(MAX\left(\left|\omega_{actual}\right|, \left|\omega_{target}\right|, \omega_{MIN}\right), \omega_{MAX}\right)$$

The critical gain and period are modeled as a function of the characteristic speed. The four GAIN_SCHEDULE parameters are written as G1..G4.

$$Ku = G2 \cdot \omega^{G1}$$

$$Pu = G4 \cdot \omega^{G3}$$

### 10.4.17 CONTROL_TYPE

This read/write parameter is used to determine the control type, using the Ziegler-Nichols method.
The value stored in CONTROL_TYPE is truncated to an integer.
If the value is negative, then SPEED_P_GAIN, SPEED_I_GAIN and SPEED_D_GAIN are read-write parameters, set by the user.
If the value is 1, a PI controller is used:

$$Kp = 0.45 \cdot Ku$$

$$Ki = {1.2 \cdot Kp} \Big/ {Pu}$$

$$Kd = 0.0$$

If the value is 2, a PID controller is used:

$$Kp = 0.6 \cdot Ku$$

$$Ki = {2.0 \cdot Kp} \Big/ {Pu}$$

$$Kd = 0.125 \cdot KpPu$$

In the case of any other value, a P controller is used:

$$Kp = 0.5 \cdot Ku$$

$$Ki = 0.0$$

$$Kd = 0.0$$

### 10.4.18 MAX_SPEED_AGE

This read/write parameter determines which digital Hall sensor transitions are used to determine the SPEED telemetry. Transitions are discarded if they are older than MAX_SPEED_AGE in time, if a complete rotor revolution has occurred since them, or if a rotor direction reversal is detected.

MAX_SPEED_AGE is relevant at very low rotor speeds. A larger value will allow more Hall sensor transitions to be used, giving a less noisy speed estimate. However, it will also increase the latency in speed measurements which may cause closed-loop speed control modes to become unstable.

### 10.4.19 LIMIT_SPEED1

This read/write parameter sets the maximum speed that closed-loop modes will target. The magnitude of the speed target used in speed, torque, momentum and acceleration modes is clamped to this value. This is particularly significant in torque and acceleration modes – if communication with the flight computer is lost for any reason the rotor will slowly accelerate until this limit is reached.

### 10.4.20 LIMIT_SPEED2

This read/write parameter sets the absolute maximum speed that the wheel can reach. If the rotor exceeds this speed the fault state will be entered. LIMIT_SPEED2 is active in all modes, which is significant since LIMIT_SPEED1 is not effective in open-loop modes (PWM, VOLTAGE, etc).

### 10.4.21 LIMIT_VOLTAGE

This read/write parameter sets the greatest voltage that can be impressed upon the motor by the drive stages.

By setting this lower than the smallest expected bus voltage, the wheel behaviour can be made insensitive to its supply. In this way, the same performance will be achieved with the spacecraft battery discharged or charged.

Lowering this parameter will also lower the maximum possible speed of the wheel, and reduce the maximum current consumed in a slew.

### 10.4.22 PREVIOUS_SPEED

This read-only parameter contains the SPEED file from the previous control frame. It is expected that it might be used in the future to generate torque telemetry, but at present it is unused.

### 10.4.23 SPEED_INTEGRATOR

This parameter contains the closed-loop controller integrator, scaled in amps of actuation. It is technically a read/write parameter, and it is possible for the user to write this for test purposes.

### 10.4.24 SPEED_LAST_ERROR

This read-only parameter contains the controller error from the previous control frame. It is used with the differential gain term of the closed-loop controller.

### 10.4.25 ACCEL_TARGET

This parameter contains the speed setpoint used by the acceleration controller. The controller will add the acceleration to this file each frame. It is technically a read/write parameter, and it is possible for the user to write this as a way to force a new speed while remaining in acceleration/torque mode.

### 10.4.26 TEST_VOLTAGE

This read-only parameter contains test information related to motor drive voltage.
In a voltage controlled mode (such as SPEED) this contains the current motor voltage setting. It can be used to see the output of the closed-loop controller.
After a MEASURE_FRICTION or MEASURE_STICTION command, this contains the minimum voltage required to maintain motion, or the minimum voltage required to break stiction, respectively.

### 10.4.27 TORQUE_[T0..T4]

These five read-only parameters record the instantaneous torques measured in the last five control frames. T0 is the result of the most recent control frame. T4 is four frames old. The torque is computed as:

TORQUE = INERTIA * (SPEED – PREVIOUS_SPEED) * 93 Hz

Torque telemetry at low speed should be used with caution. The speed estimate is only updated when new hall sensor pulses are seen (or a very long period elapses). If there has been no hall sensor pulse in the previous control frame then SPEED == PREVIOUS_SPEED and so TORQUE == 0.

### 10.4.28 MODE

This parameter stores the wheel's current mode. It is more often accessed through file 0, where the mode and command value can be read or written simultaneously.

### 10.4.29 HALL_IMPOSSIBLE

This value counts the number of times that a transition to an "impossible" digital Hall-effect sensor configuration is seen. Impossible configurations are all "0", or all "1". This is an error condition, and would normally indicate failure of a sensor or loss of a rotor magnet. It is read/write, and can be written as zero to reset the count. The count range is 0..255. If an impossible configuration occurs with the count at 255 it will cycle back to 0.

### 10.4.30 HALL_SKIP

This value counts the number of times that a Hall-effect sensor pattern transitions to another pattern that should not be immediately adjacent. Adjacent sensor patterns are those that differ by only one bit.

### 10.4.31 CONTROL_OVERFLOW

This value counts the number of control frames where the control algorithm has not finished processing before the start of the next frame. This is an error condition, and would be expected to result in poor control. It is read/write, and can be written as zero to reset the count. The count range is 0..255. If a control overflow occurs with the count at 255 it will cycle back to 0.

### 10.4.32 SFFT_STEP_NUMBER

This value contains the step number of the current script that is being executed. It is not intended for this to be written to effect a goto behaviour.

### 10.4.33 SFFT_TELEM_COUNT

This value contains the offset from the start of the script telemetry at which the latest script telemetry will be written.

### 10.4.34 MODE_MONITOR

This value contains the effective mode command that is currently being commanded by a script. Its value outside of script execution is not updated.

### 10.4.35 FRICTION_DONE

This Boolean value is used internally by MEASURE_FRICTION and MEASURE_STICTION modes to signal when the rotor has stopped, or started, respectively.

## *10.5 Command Modes*

| Command Number | Command Name | Command Value |
|---|---|---|
| 0x00 | IDLE | Ignored |
| 0x01 | PWM | Duty cycle (-1.0 to +1.0) |
| 0x02 | VOLTAGE | Volts (-10 to +10) |
| 0x03 | SPEED | Rads/sec |
| 0x04 | PWM_H1 | Duty cycle (-1.0 to +1.0) |
| 0x05 | PWM_H2 | Duty cycle (-1.0 to +1.0) |
| 0x06 | PWM_H3 | Duty cycle (-1.0 to +1.0) |
| 0x07 | PWM_H4 | Duty cycle (-1.0 to +1.0) |
| 0x08 | PWM_H5 | Duty cycle (-1.0 to +1.0) |
| 0x09 | PWM_H6 | Duty cycle (-1.0 to +1.0) |
| 0x0A | VOLTAGE_H1 | Volts (-10 to +10) |
| 0x0B | VOLTAGE_H2 | Volts (-10 to +10) |
| 0x0C | VOLTAGE_H3 | Volts (-10 to +10) |
| 0x0D | VOLTAGE_H4 | Volts (-10 to +10) |
| 0x0E | VOLTAGE_H5 | Volts (-10 to +10) |
| 0x0F | VOLTAGE_H6 | Volts (-10 to +10) |
| 0x10 | ACCEL | Rads/sec$^2$ |
| 0x11 | MOMENTUM | N-m-sec |
| 0x12 | TORQUE | N-m |
| 0x13 | BURNIN | Final test step # |
| 0x14 | SFFT | Final test step # |
| 0x15 | LIFE | Final test step # |
| 0x16 | STORE_FILES | 0.0 or 1.0 |
| 0x17 | DEFAULT_FILES | 0.0 or 1.0 |
| 0x18 | PWM_P0 | Duty cycle (-1.0 to +1.0) |
| 0x19 | PWM_P1 | Duty cycle (-1.0 to +1.0) |
| 0x1A | PWM_P2 | Duty cycle (-1.0 to +1.0) |
| 0x1B | MEASURE_FRICTION | Volts/sec |
| 0x1C | MEASURE_STICTION | Volts/sec |
| 0x1F | SOAK | Final test step # |
| 0x20 | REPEAT | Ignored |
| 0x21 | COMPLETE | Ignored |

| | | |
|---|---|---|
| 0x24 | AUX1 | Final test step # |
| 0x25 | AUX2 | Final test step # |

### 10.5.1 IDLE

In IDLE mode the motor drive is turned off. If it is spinning, the rotor is free to slow down under friction.

### 10.5.2 PWM

In PWM mode the motor is driven with a constant duty cycle. The command may be between -1.0 and 1.0. This is interpreted as a duty cycle between 0.0 and 1.0, in either the positive or negative direction.

PWM mode does not use closed-loop current or speed control, so it is not of great use in spacecraft fine control. However it does allow for extremely high torques (and very high power consumption!), so it may be used open-loop during slew maneuvers.

### 10.5.3 VOLTAGE

In VOLTAGE mode the motor is driven with a constant voltage. The desired voltage is divided by the VSENSE telemetry measurement to determine the PWM duty cycle. This mode is open-loop, in a manner similar to PWM mode, but has feed-forward compensation against supply voltage variations.

### 10.5.4 SPEED

In SPEED mode the rotor speed is servoed to the command value. The closed-loop speed controller outputs a voltage setpoint, which is in turn used by the voltage controller.

### 10.5.5 PWM_H[1..6]

In these modes the digital Hall-effect sensors are overridden, and the binary code is set to the H1..H6 value. Other than that, the mode is identical to PWM mode. It allows a particular PWM duty cycle to be driven onto a particular motor phase regardless of the rotor position. The rotor will typically not spin in these modes, but will oscillate about a particular electrical angle.

### 10.5.6 VOLTAGE_H[1..6]

In these modes the digital Hall-effect sensors are overridden, and the binary code is set to the H1..H6 value. Other than that, the mode is identical to VOLTAGE mode. It allows a particular voltage to be driven onto a particular motor phase regardless of the rotor position. The rotor will typically not spin in these modes, but will oscillate about a particular electrical angle.

### 10.5.7 ACCEL

When not in ACCEL mode, the ACCEL_TARGET file is set to SPEED. In ACCEL mode, the acceleration command is added to ACCEL_TARGET each control frame. ACCEL_TARGET is then used as the setpoint for the speed mode controller.

### 10.5.8 MOMENTUM

In MOMENTUM mode, the SPEED controller is used with a setpoint equal to the commanded MOMENTUM divided by the INERTIA file.

### 10.5.9 TORQUE

In TORQUE mode, the ACCEL controller is used with a setpoint equal to the commanded TORQUE divided by the INERTIA file.

### 10.5.10 BURNIN

The BURNIN mode starts a test script intended to bring the bearing lubricant to a steady-state initial condition. Details are TBD.

### 10.5.11 SFFT

The SFFT mode starts a test script to fully evaluate the health of an integrated reaction wheel. It concentrates on speed capacity, torque capacity, EMF characterization and friction characterization. Closed-loop control performance is not investigated.

The test will run for a number of minutes before terminating. All of the result data is stored in the parameter file. Consult the factory for automated software that will generate pass/fail reports.

### 10.5.12 LIFE

The LIFE mode starts a test script intended for long-term operation on a life-test reaction wheel. Details are TBD.

### 10.5.13 STORE_FILES

If the STORE_FILES mode is entered with a value of exactly 1.0, all of the parameters will be stored to non-volatile flash memory. The mode value will be set to 0.0, to indicate that the write has occurred and to prevent multiple writes. Whenever the wheel resets it will start with the stored parameters.

This mode does not drive the motor, and is equivalent in that way to IDLE.

### 10.5.14 DEFAULT_FILES

If the DEFAULT_FILES mode is entered with a value of exactly 1.0 the stored parameters in non-volatile flash memory are erased. The mode value will be set to 0.0, to indicate that the erasure has occurred and to prevent multiple erasures. Whenever the wheel resets it will start with default parameters. This command has no effect on the parameters currently in the wheel parameter file, only on the parameters after the next reset.

This mode does not drive the motor, and is equivalent in that way to IDLE.

### 10.5.15 PWM_P[0..2]

The PWM_P[0..2] modes allow the duty cycle of a particular motor phase (0..2) to be set. Only the one phase is driven, and none of the phases is connected to ground.

### 10.5.16 MEASURE_FRICTION

The MEASURE_FRICTION mode should be entered with the wheel spinning, driven in a voltage-based mode (VOLTAGE, SPEED, etc). The drive voltage is slowly reduced, at a number of volts/second dictated by the mode value. Eventually the voltage will be so low that the rotor will stop.
As soon as the rotor speed reads 0.0 rad/sec, the TEST_VOLTAGE parameter will stop updating. Thus, by reading TEST_VOLTAGE the lowest voltage consistent with rotation can be determined. The HALL_DIGITAL parameter can also be read, to determine the electrical angle at which the rotor stopped.

### 10.5.17 MEASURE_STICTION

The MEASURE_STICTION mode should be entered with the wheel stopped, driven in a voltage-based mode (VOLTAGE, SPEED, etc). The drive voltage is slowly increased, at a number of volts/second dictated by the mode value. The sign of the mode value indicates whether the voltage should become positive or negative.
As soon as the rotor speed becomes non-zero, the TEST_VOLTAGE parameter will stop updating. Thus, by reading TEST_VOLTAGE the lowest voltage consistent with breaking stiction can be determined.

### 10.5.18 SOAK

The SOAK mode starts a test script intended to facilitate the 120 hour high-temperature burn-in test. It is expected that the electronics unit will be connected to a stator, but there will not be a rotor. The mode drives current through each of the motor phases in turn and logs analog telemetry.
Not presently implemented.

### 10.5.19 REPEAT

This mode exists as a flag within a script, indicating that the script should return to the first step. It is not a mode that can be usefully commanded by a user.

### 10.5.20 COMPLETE

This mode exists as a flag within a script, indicating that the script should terminate. It is not a mode that can be usefully commanded by a user.

### 10.5.21 AUX1

The AUX1 mode starts a test script to investigate closed-loop control performance through zero speed crossings.

The test will run for a number of minutes before terminating. All of the result data is stored in the parameter file. Consult the factory for automated software that will generate pass/fail reports.

### 10.5.22 AUX2

The AUX2 mode starts a test script to investigate closed-loop control performance to step acceleration commands at different wheel speeds.

The test will run for a number of minutes before terminating. All of the result data is stored in the parameter file. Consult the factory for automated software that will generate pass/fail reports.