

Command:RDSBYTE (RB)

MS2000 or RM2000 syntax

Shortcut	RB
Format	RDSBYTE [axis] = [status_byte]...

Tiger syntax

Shortcut	RB
Format	RDSBYTE [axis] = [status_byte]...
Type	Axis-Specific

Requests the TG-1000 and MS-2000 to respond with the Status Byte (see definition below). Similar to [RDSTAT](#) but returns the raw byte.

Note: If you don't see the serial response, it may be because the reply is in raw bytes, see the [Python script](#) below for an example of how to parse the data. Or use the [RDSTAT](#) command without any modifier, which returns the same byte in decimal form.

Status Byte Format

76543210 → bit position (0 is LSB)

00001010 → bit pattern for hex 0x0A, decimal 10

MSB: :LSB → most significant bit - least significant bit

The number is one byte, which can be broken down into 8 bits that represent the following internal flags:

Bit 0	0 = No commanded move is in progress. 1 = A commanded move is in progress. This bit is synonymous with the STATUS command. If the bit is set, then STATUS returns B, otherwise STATUS returns N.
Bit 1	0 = The axis is disabled. 1 = The axis is enabled. Axes can be re-enabled using one of the following: High Level command MC <axis>+, cycles the clutch switch for the Z-axis, Low Level Start Motor command (hex 47), or a system reset.
Bit 2	0 = Motor is inactive (off) 1 = Motor is active (on).
Bit 3	0 = Joystick/Knob disabled 1 = Joystick/Knob enabled
Bit 4	0 = Motor not ramping 1 = Motor ramping
Bit 5	0 = Ramping down 1 = Ramping up

Bit 6	Upper Limit Switch: 0 = Open 1 = Closed
Bit 7	Lower Limit Switch: 0 = Open 1 = Closed

Reply

:<byte as hexadecimal> in the examples below

<0x3A><byte as hexadecimal><0x0D><0x0A> as raw bytes

The reply should be read as raw bytes, a common pattern for the Status Byte is 0x0A, which is <LF> when interpreted as ASCII. There is a Python script at the bottom of this page as an example.

The total length of the reply will be the number of axes sent in the command plus 3.
For example RB X Y Z will reply with 6 bytes.

The response :<0x0A><0x0A> below is [58, 10, 10, 13, 10] in raw bytes.
58 = ': ', 13 = '\r', 10 = '\n' are always a part of the response.

MS-2000 Example

```
RB X
:<0x8A>
RB X Y
:<0x8A><0x0A>
RB X Y Z
:<0x8A><0x0A><0x0A>
```

TG-1000 Example

```
1RB X Y
:<0x8A><0x0A>
```

The X-axis example value of 0x8A means the following:

Bit7: 1 - X Axis is at its lower limit
 Bit6: 0 - X Axis upper limit switch open
 Bit5: 0 - Ramping down, if ramping
 Bit4: 0 - Motor not ramping
 Bit3: 1 - Joystick/Knob is enabled
 Bit2: 0 - Motor power is off
 Bit1: 1 - X Axis is enabled
 Bit0: 0 - No commanded move is in progress

Note: Motor power can be on while a commanded move is

not in progress and the stage appears not to be moving. This happens when the motor is either making a final adjustment to a commanded move or when it is applying a force to maintain the stage position.

Python Helper Script

Python Status Byte

```
# /// script
# dependencies = ["pyserial>=3.5"]
# ///
import serial
from enum import Flag, auto

class StatusByte(Flag):
    """
    The Status Byte returned by the RDSBYTE (RB) command.
    The value parameter should be an 8-bit int, 0-255.
    00000001 <- Bit 0 is 1, commanded move in progress.
    """
    COMMANDED_MOVE = auto()
    AXIS_ENABLED = auto()
    MOTOR_ACTIVE = auto()
    JS_KNOB_ENABLED = auto()
    MOTOR_RAMPING = auto()
    MOTOR_RAMPING_UP = auto()
    AT_UPPER_LIMIT = auto()
    AT_LOWER_LIMIT = auto()

def main() -> None:
    # use an empty string for MS2000 (card_address = "")
    card_address = "1"

    # which axes to query (for a single axis use: axes = ["X"])
    # axes_byte_len is the number of bytes that need to be read for
    RDSBYTE
    axes = ["X", "Y"]
    axes_str = " ".join(axes)
    axes_byte_len = len(axes) + 3 # 3 bytes for ':', '\r', and '\n'

    with serial.Serial("COM5", 115200, timeout=1) as serial_port:
        # query the controller for the status byte of each axis
        command = f"{card_address}RB {axes_str}\r"
        serial_port.write(bytes(command, encoding="ascii"))
        response = serial_port.read(axes_byte_len)

        # report and check for errors
        print(f"Send: \"{command[:-1]}\r\"")
```

```

    print(f"Recv: \"{response}\" (interpreted as ASCII)")
    print(f"Number of bytes to read: {axes_byte_len}\n")
    if b"N" in response:
        print("Error in response...")
        return

    # view as bytes
    response_bytes = [byte for byte in response]
    print(f"{response_bytes = } (raw bytes)\n")

    # get the status byte for each axis and skip the first byte
    (':')
    status_bytes = [response[i] for i in range(1, len(axes) + 1)]
    print(f"{status_bytes = } (decimal)")

    # create a dictionary that maps uppercase axis names to status
    bytes
    status_bytes_dict = {axis.upper(): StatusByte(status_byte) for
    (axis, status_byte) in zip(axes, status_bytes, strict=True)}
    print(f"{status_bytes_dict = }\n")

    # check the status of each axis
    for axis in axes:
        status_byte = status_bytes_dict.get(axis.upper())
        # check a single flag
        is_at_upper_limit = StatusByte.AT_UPPER_LIMIT in
status_byte
        # check multiple flags
        is_js_and_axis_enabled = StatusByte.JS_KNOB_ENABLED |
StatusByte.AXIS_ENABLED in status_byte
        # check if one flag is True and the other is False
        is_motor_on_and_not_cmd_move = StatusByte.MOTOR_ACTIVE in
status_byte and StatusByte.COMMANDED_MOVE not in status_byte
        print(f"{axis} Axis Status: {status_byte.value:08b}") #
print the status_byte in binary
        print(f"{status_byte = }")
        print(f"{is_at_upper_limit = }")
        print(f"{is_js_and_axis_enabled = }")
        print(f"{is_motor_on_and_not_cmd_move = }\n")

    if __name__ == "__main__":
        main()

```

[commands](#), [tiger](#), [ms2000](#)

From:

<https://docs.asiimaging.com/> - **Applied Scientific Instrumentation**

Permanent link:

<https://docs.asiimaging.com/commands/rdsbyte>

Last update: **2026/03/12 17:25**

