Raspberry Pi Build HAT serial protocol

Working with the Build HAT firmware serial protocol

Colophon

build-date: 2021-10-19
build-version: githash: 91fd59e-clean

Legal Disclaimer Notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI (TRADING) LTD ("RPTL) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPTL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPTL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPTL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPTL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPTL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPTL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPTL's Standard Terms. RPTL's provision of the RESOURCES does not expand or otherwise modify RPTL's Standard Terms including but not limited to the disclaimers and warranties expressed in them.

Introduction

The BuildHAT is a board that provides an interface between a Raspberry Pi host and up to four LEGO LPF2 (LEGO Power Functions version 2) devices. Supported LPF2 devices include a wide range of actuators and sensors. Firmware running on the HAT deals with the hard real-time requirements of the LPF2 devices, including monitoring for connection and disconnection events and interrogating devices to determine their capabilities and properties.

The HAT communicates with the Raspberry Pi host over the 'command port', a 115200 baud serial interface with eight bits per character, no parity and one stop bit ('8N1'). There is no flow control. The command port protocol is entirely in plain text, and it is perfectly possible to simply run a terminal emulator on the host and interact manually with the HAT. When experimenting the HAT in this way it is convenient to enable echo mode so that you can see what you are typing: see the description of the `echo` command below. See also the `plimit` command, which must be sent before many operations will work correctly.

This document describes the commands available over that interface.

NOTE

A Python library is provided that provides a higher-level interface to the functions of the HAT: in most cases it will be preferable to use that library rather than the lower-level commands described here.

Port and Device Basics

The firmware numbers the ports from 0 to 3. There is the notion of the 'current port', set using the `port` command. Many commands implicitly address the current port.

Each port may have a device connected to it. A device may be 'active', which means that it communicates with the HAT using a serial interface, or it may be 'passive'. Passive devices include some lights and motors, although most types of motor are active devices. Active devices can offer some feedback to the HAT: for example, an active motor might contain position or speed sensors to allow it to be controlled precisely.

An active device has up to sixteen 'modes'. A mode can be thought of as a small memory buffer in the device, much like the concept of a 'characteristic' in Bluetooth terminology. Some modes are intended to be written to, to control the device; some are intended to be read from, to extract sensor readings for example.

When a device is plugged in to the HAT, the HAT emits a 'connected' message followed by the information it has about the device. For a passive device this will just be an identifying code number. For an active device this will be an identifying code number followed by other information including the connection baud rate, software and hardware version numbers and a list of the available modes. See the `list` command below for more detail.

When a device is unplugged from the HAT, a 'disconnected' message is emitted.

Device Power

The LPF2 connector supplies power to a connected device in two ways. The first of these is a normal digital logic power supply at +3.3V which is always present. If a device attempts to draw too much current from this supply then a 'Port power fault' message is emitted for as long as the fault persists.

The second power source is at about +7.2V and intended for driving motors and other relatively high-power devices. This is supplied by a separate dedicated driver IC for each port. If a fault is detected on this supply then a 'Motor power fault' message is emitted repeatedly. This fault condition is latched in hardware and must be cleared explicitly after the cause of the fault has been resolved: see the `clear_faults` command below.

The motor driver ICs can output PWM waveforms of either polarity to allow speed and direction control of motors.

On-board Controllers

The firmware implements an independent controller for the motor power output on each port of the HAT. The controller can be in one of two modes: 'direct PWM' (the default) and 'PID'. In addition, each controller has an associated setpoint, which can be constant or varying: see the `set` command below.

In direct PWM mode the setpoint, which must be in the range from –1 to +1, directly controls the power output. This is useful in simple motor applications, for driving lights, and for certain devices that use 'motor power' for other purposes. Such devices usually need to have power enabled very shortly after connection is established, and usually need to be powered in reverse: i.e., `set -1`.

In PID mode a proportional-integral-differential controller reads a value from a sensor: typically this will be a speed or angle sensor on a motor being controlled. This value is called the 'process variable'. The PID controller adjusts the output power to attempt to have the process variable track the setpoint closely. Using this you can, for example, attempt to run a motor at constant speed under varying load, or move a motor to a given position. See the `pid` command below for more details.

Command Summary

Any command can be abbreviated to its shortest unique prefix. Multiple commands can be given on one line separated by semicolons.

help, ?

Prints a synopsis of the available commands.

echo <0|1>

Disables (default) or enables echo of characters received over the command port.

version

Prints the version string for the currently-running firmware.

port <port>

Sets the current port, used implicitly by many other commands.

vin

Prints the voltage present on the input power jack.

ledmode <ledmode>

Sets the behaviour of the HAT's LEDs.

ledmode	Effect
−1	LEDs lit depend on the voltage on the input power jack (default)
0	LEDs off
1	orange
2	green
3	orange and green together

list

Prints a list of all the information known about the LPF2 devices connected to the HAT. Typical output is as follows.

```
P0: connected to active ID 40
type 40
```

```
    nmodes =3
    nview  =3
    baud   =115200
    hwver  =00000004
    swver  =11000000
    M0 LEV O SI = PCT
      format count=1 type=0 chars=1 dp=0
      RAW: 00000000 00000009    PCT: 00000000 00000064    SI: 00000000 00000009
    M1 COL O SI = PCT
      format count=1 type=0 chars=2 dp=0
      RAW: 00000000 0000000A    PCT: 00000000 00000064    SI: 00000000 0000000A
    M2 PIX O SI =
      format count=9 type=0 chars=3 dp=0
      RAW: 00000000 000000AA    PCT: 00000000 00000064    SI: 00000000 000000AA
    M3 TRANS SI =
      format count=1 type=0 chars=1 dp=0
      RAW: 00000000 00000002    PCT: 00000000 00000064    SI: 00000000 00000002
       speed PID: 00000000 00000000 00000000 00000000
    position PID: 00000000 00000000 00000000 00000000
  P1: no device detected
  P2: connected to passive ID 8
  P3: connected to active ID 30
  type 30
    nmodes =5
    nview  =3
    baud   =115200
    hwver  =00000004
    swver  =10000000
    M0 POWER SI = PCT
      format count=1 type=0 chars=4 dp=0
      RAW: 00000000 00000064    PCT: 00000000 00000064    SI: 00000000 00000064
    M1 SPEED SI = PCT
      format count=1 type=0 chars=4 dp=0
      RAW: 00000000 00000064    PCT: 00000000 00000064    SI: 00000000 00000064
    M2 POS SI = DEG
      format count=1 type=2 chars=11 dp=0
      RAW: 00000000 00000168    PCT: 00000000 00000064    SI: 00000000 00000168
    M3 APOS SI = DEG
      format count=1 type=1 chars=3 dp=0
      RAW: 00000000 000000B3    PCT: 00000000 000000C8    SI: 00000000 000000B3
    M4 CALIB SI = CAL
      format count=2 type=1 chars=5 dp=0
      RAW: 00000000 00000E10    PCT: 00000000 00000064    SI: 00000000 00000E10
    M5 STATS SI = MIN
      format count=14 type=1 chars=5 dp=0
      RAW: 00000000 0000FFFF    PCT: 00000000 00000064    SI: 00000000 0000FFFF
    C0: M1+M2+M3
       speed PID: 00000BB8 00000064 00002328 00000438
    position PID: 00002EE0 000003E8 00013880 00000000
```

For each port the HAT ports the type and ID of any device connected. For active devices it also gives the number of 'modes' and 'views', the baud rate of the HAT's connection to the device, and the device's hardware and software version numbers. Next comes a list of the available modes, any possible combi modes, and any recommended PID parameters.

The first line for each mode gives its name (such as `LEV`) and unit, such as `PCT` or `DEG`. The second line gives the number of

data items in the mode (usually 1) , its type (0=signed char, 1=signed short, 2=signed int, 3=float), a hint as to a suitable number of characters to use to display its value, and a suitable number of decimal places. The third line gives minimum and maximum values for the data in that mode in various units.

clear_faults

Clears any latched motor power fault.

coast

Switches the motor driver on the current port to 'coast' mode, that is, with both outputs floating.

pwm

Switches the controller on the current port to direct PWM mode.

off

Same as `pwm; set 0`.

on

Same as `pwm; set 1`.

pid <pidparams>

Switches the controller on the current port to PID mode. The `pidparams` specify from where the process variable is to be fetched and the gain coefficients for the controller itself. They are, in order, as follows.

Name	Meaning
`pvport`	port to fetch process variable from
`pvmode`	mode to fetch process variable from
`pvoffset`	process variable byte offset into mode
`pvformat`	process variable format: `u1`=unsigned byte, `s1`=signed byte, `u2`=unsigned short, `s2`=signed short, `u4`=unsigned int, `s4`=signed int, `f4`=float
`pvscale`	process variable multiplicative scale factor
`pvunwrap`	0=no unwrapping; otherwise modulo for process variable phase unwrap
`Kp`	proportional gain
`Ki`	integral gain
`Kd`	differential gain
`windup`	integral windup limit

The PID controller fetches the process variable from the mode specified by the `pvport`, `pvmode`, `pvoffset`, `pvformat` parameters. Note that a suitable `select` command is required to ensure that this mode's data are available.

It then multiplies the value from the mode by `pvscale`.

The `pvunwrap` parameter allows 'phase unwrapping' of the process variable. The commonest use of this is with an angular sensor that outputs a value from −180° to +179° depending on its absolute position. When the sensor is turned continuously, the reading will jump from +179° to −180° once per revolution. Setting the `pvunwrap` parameter to 360 will cause the unwrapper to add or subtract 360° at each discontinuity to make its output continuous (and in principle infinite in range). More precisely, the output of the unwrapper is guaranteed equal to its input modulo the `pvunwrap` parameter, with the smallest possible change between one sample and the next.

`Kp`, `Ki` and `Kd` are the standard PID controller parameters. The implied unit of time in the integrator and differentiator is one second.

The output of the error integrator is clamped in absolute value to the `windup` parameter.

set <setpoint>

Configures the setpoint for the controller on the current port. The setpoint can be a (floating-point) constant or a waveform specification.

A waveform specification is one of `square`, `sine`, `triangle`, `pulse` and `ramp`, followed by four floating-point parameters.

For square, sine and triangle waveforms these parameters are the minimum value, the maximum value, the period in seconds, and the initial phase (from 0 to 1). For a pulse waveform the first three parameters are the setpoint value during the pulse, the setpoint value after the pulse, and the duration of the pulse; the fourth parameter is ignored. For a ramp waveform the first three parameters are the setpoint value at the start of the ramp, the setpoint value at the end of the ramp, and the duration of the ramp; the fourth parameter is again ignored.

When a pulse or ramp is completed, a message `pulse done` or `ramp done` is emitted.

bias <bias>

Sets a bias value for the current port which is added to positive motor drive values and subtracted from negative motor drive values. This can be used to compensate for the fact that most DC motors require a certain amount of drive before they will turn at all.

plimit <limit>

Sets a global limit to the motor drive power on all ports. For safety when experimenting the default value is 0.1; this will usually need to be increased.

select

Deselects any previously-selected mode on the current port.

select <selmode>

Selects the specified mode on the current port and repeatedly outputs that mode's data as raw hexadecimal.

select <selmode> <offset> <format>

Selects the specified mode on the current port. Repeatedly extracts a value from that mode starting at the specified offset, interpreting it according to the specified format (`u1`=unsigned byte, `s1`=signed byte, `u2`=unsigned short, `s2`=signed

short, `u4`=unsigned int, `s4`=signed int, `f4`=float) and outputs that value.

selonce

As `select` but outputs data once rather than repeatedly.

combi <index>

Deconfigures any previously-configured combi mode on the current port at the specified index.

combi <index> <clist>

Configures a combi mode on the current port at the specified index. `clist` is a list of pairs of numbers, each pair giving a mode and an offset into that mode. Note that the offset is a 'dataset' offset, i.e., is multiplied by the size of the data elements in that mode (as given by the `list` command) to obtain a byte offset.

write1 <hexbyte>*

Writes the given hexadecimal bytes to the current port, the first byte being a header byte. The message is padded if necessary, and length and checksum fields are automatically populated.

write2 <hexbyte>*

Writes the given hexadecimal bytes to the current port, the first two bytes being header bytes. The message is padded if necessary, and length and checksum fields are automatically populated.

debug <debugcode>

Not required for normal use. Note that some debug modes can generate output faster than the serial port can handle at its standard speed.

signature

Not required for normal use.

Appendix A: Examples

The following are some simple examples to illustrate how to use the above commands.

Using a motor from the SPIKE Prime set

Plug the motor into port 0. Then send

```
port 0
```

to address port 0,

```
plimit 1
```

to remove the power limit, and

```
set triangle 0 1 10 0
```

to generate a triangle wave setpoint. The motor will accelerate to full speed and decelerate back to zero continuously with a period of ten seconds.

Now try

```
port 0
combi 0 1 0 2 0 3 0
```

to select a combi mode with index zero containing data from modes 1, 2 and 3,

```
select 0
```

to select this combi mode, and

```
plimit 1 ; bias .4
```

to remove the power limit and set a reasonable bias value for the motor. You can now set up a position PID controller that reads a 2-byte value from offset 5 in this combi mode, which is the absolute position of the motor in degrees, from −180° to +179°. We scale this by 1/360=0.0027777778 to get a position in revolutions from −0.5 to +0.5, unwrap the phase with a modulo of 1:

```
pid 0 0 5 s2 0.0027777778 1 5 0 .1 3
```

Here the PID parameters are Kp=5, Ki=0 and Kd=0.1. The integral windup limit is set arbitrarily at 3.

Now if you send

```
set square 0 1 3 0
```

the motor will alternately rotate one revolution clockwise and one revolution anticlockwise with a period of three seconds.

Using the colour sensor from the SPIKE Prime set

```
port 0 ; plimit 1 ; set -1 ; select 0
```

will turn on the sensor's light and continuously report a number corresponding to the colour detected. The sensor has many other modes.

Using the ultrasonic distance sensor from the SPIKE Prime set

```
port 0 ; plimit 1 ; set -1 ; select 1
```

will power up the sensor and continuously report a number corresponding to the distance to the object in front of the sensor in millimetres.

To control the four LEDs on the sensor use the following sequence

```
select 5 ; write1 c5 pp qq rr ss
```

where `pp`, `qq`, `rr` and `ss` are four hexadecimal numbers from 0 to 0x64 (100 decimal) that control the brightnesses of the individual LEDs.

Using the force sensor from the SPIKE Prime set

`port 0 ; select 0`

will continuously report a number corresponding to the force detected.

Using the 3x3 colour light matrix from the SPIKE Essential set

```
port 0 ; plimit 1 ; set -1
```

will turn on the light matrix. If the device does not receive power shortly after connection is established it will disconnect.

```
select 0 ; write1 c0 p
```

where p is a number from 0 to 9, will light the matrix in bar-graph style according to the value of p.

```
select 1 ; write1 c1 p
```

where p is a hexadecimal number from 0 to 0x0a will light the matrix in a solid colour according to the value of p: 0=off; 1=red; 2=magenta; 3=blue; 4=cyan; 5=pale green; 6=green; 7=yellow; 8=orange; 9=red, 0a=white.

The most flexible mode is mode 2, where you can specify the colour of each LED individually.

```
select 2 ; write1 c2 12 23 34 45 12 23 34 45 12
```

gives a random pattern of dull colours and

```
write1 c2 67 72 78 82 89 92 9a a4 aa
```

gives some random bright ones. The first hex digit (from 0 to a) specifies the brightness and the second hex digit (from 0 to a) the basic colour. So for example

```
write1 c2 1a 2a 3a 4a 5a 6a 7a 8a 9a
```

gives shades of white and

```
write1 c2 a1 a2 a3 a4 a5 a6 a7 a8 a9
```

gives all the basic colours at full brightness.

Mode 3 allows you to specify transitions.

```
write1 c3 1
```

enables row-by-row animated transitions, while

```
write1 c3 2
```

enables a fade to black and fade back up.

Appendix B: Passive ID Codes

ID (decimal)	Device
1	System medium motor
2	System train motor
3	System turntable motor
4	general PWM/third party
5	button/touch sensor
6	Technic large motor (some have active ID)
7	Technic XL motor (some have active ID)
8	simple lights
9	Future lights 1
10	Future lights 2
11	System future actuator (train points)

Appendix C: Active ID Codes

ID (hex)	Device
25	colour and distance sensor
26	Medium linear motor
2E	Technic large motor
2F	Technic XL motor
30	SPIKE Prime medium motor
31	SPIKE Prime large motor
3D	SPIKE Prime colour sensor
3E	SPIKE Prime ultrasonic distance sensor
3F	SPIKE Prime force sensor
40	SPIKE Essential 3x3 colour light matrix
41	SPIKE Essential small angular motor

Raspberry Pi Trading Ltd