PM2D

Concept2 PM2+ Ergometer interface daemon

Design and API documentation
version 0.10 - working version -
September 20, 2002

Table of Contents

Overview

The PM2+ unit is an -optional- unit for the Concept2 ergometer. It samples various data from the ergometer and the rower, and has a serial interface to report these data to a computer. Several PM2+ units can be daisy-chained, allowing one computer to collect data from many units.

The serial handling for the interface is not really simple. Due to the asynchronous nature of the protocols involved, any program should be able to handle exceptions like missing bytes or unresponsive units.

pm2d is a daemon for POSIX-compliant Unix systems. It offers a HTTP based interface to user level programs, while handling all serial communication to the actual PM2+ units. pm2d is not a complete rower handling system by itself, but it separates the complexity of interfacing the units from a presentation-like program.

pm2d offers two modes of operation: direct issue of interface commands, and autonomous mode. The direct issue mode will take a request from a HTTP interface, send it to the PM2+ units, and report the result and the command output (if applicable). In the autonomous mode, pm2d will send appropriate setup commands to the units, collect data from the PM2+ units from the beginning of a session up to the end, and report the complete set of data to a presentation program (or partial results, if the request is received before the session has ended).

Key highlights:

- pm2d can handle all PM2+ commands currently known.
- pm2d can collect data from complete sessions, as well as process single commands.
- pm2d supports multiple serial interfaces, each of which can connect multiple PM2+ units[1].
- Interfacing to pm2d can be in any programming language that supports TCP/IP.
- Presentation programs can run on the same system, but they can also run on another system. Trough the use of HTTP as transport protocol, presentation programs could also run over the Internet.
- Multiple presentation programs can use the data of the same session, at the same time. One could be an online monitor, the other could store the results in a database, a third could collect data from several rowing sessions (e.g. racing over the Internet).

Legal stuff

1 In some future version, this can easily be extended to a master-slave concept, that allows multiple pm2d pollers to report to one pm2d master.

RPM Distribution

The current release of pm2d is pm2d-1.0.1. The corresponding RPM is pm2d-1.0.1-1.i386.rpm, and can be downloaded from the project site on Sourceforge.net.

The rpm will install:

- Copy this document, in your standard documentation tree (probably something like /usr/share/doc/packages/pm2d/pm2d.pdf)
- Install the executable daemon, as /usr/bin/pm2d
- Create a log directory /var/log/pm2d

Interfacing to pm2d

A program can interface to pm2d using the HTTP protocol. By default[2], pm2d will run on port 8880.

A typical command interaction would be as follows:

- connect to port 8880
- send a HTTP header

  The HTTP header can contain any number of additional lines. This makes it possible to do simple debugging (connect to pm2d using a browser). pm2d will skip most headers regardless of their content[3].
- receive a HTTP reply
- If the HTTP reply contains return code 200 (OK), then skip the header and read the returned data.

A sample C code snippet to do the interfacing to pm2d:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
...
{
  struct sockaddr_in srv;
  int port;
  int s;
  int bytes;
  char buf[1024];

strcpy(buf, “GET /pm2-read-meters HTTP/1.1\r\n\r\n”);          // this is the request code
length=strlen(buf);
  hostnm=gethostbyname("pm2-server.yourdomain.com");          // alter this to your server domain
  port=8880;
// port the server runs on
  srv.sin_family=AF_INET;
  srv.sin_port=htons(port);
  srv.sin_addr.s_addr=*((unsigned long *) hostnm->h_addr);
  s=socket(AF_INET, SOCK_STREAM,0);
  connect(s, (struct sockaddr *) &srv, sizeof(srv));
  write(s, buf, length);
  while ((bytes=read(s, buf, sizeof(buf)))>=0) {
     if (bytes>0) {
        buf[bytes]='\0';
        printf(buf);              // add processing here, pick up return code and result from
header
     }
     if (bytes==0) break;
```

2 The port number is configurable from the pm2d command line
3 The only exceptions are those header lines associated with session-keepalive

```
  }
  close(s);
}
```

A standard set of return headers from pm2d looks as follows:

```
HTTP/1.1 200 OK

Date: Thu Mar  7 22:15:28 2002

Server: pm2d Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 Pragma: no-cache
Content-Type: text/html


0,0,c0,0.000000
```

Note the empty line between the cache-control header and the 0,0,c0... part; this empty line signals the end of the headers and the start of the payload data. Counting the number of lines to reach the start of the payload is not recommended, since the number of returned header lines may change in a future version.

The headers contain one useful data item: the Date: line. This is the date and time of processing the command. Other lines currently only serve the purpose of disabling caching in browsers and proxies as much as possible.

All data output lines from pm2d are in the following format:

<poller id>,<rower id within poller>,<data specific to command>

Note that several output lines will be returned if more than one PM2+ unit is connected, each line identified by poller id and rower id.

The use of HTTP also allows you to test the interface using a regular browser. Just enter a request in the URL bar (like http://your-server.com:8880/pm2-get-meters), and take a look at the output.

OS support

pm2d should build on most POSIX compliant Unix systems. However, I'm building on Linux (2.4.18 kernel). Most important for porting is the pthreads library.

There is no configure script yet, nor is there a make file.

Basic functionality has been tested and verified on FreeBSD 4.3, SuSE Linux 7.2, and systems loosely based on these distributions.

On Linux, compile as gcc -o pm2d -lpthread pm*.c

On FreeBSD, compile as gcc -o pm2d -pthread pm*.c

Solaris/Intel: compile as gcc -o pm2d -pthread pm*.c,

or cc -o pm2d -lsocket -lpthread pm*.c if you are using Sun's compilers.

Unknown interface commands

There are a couple of interface commands for which there are no commands in pm2d. Mainly this is because I do not know how they work (or what they do).

These are:

cmd	*notes*	*context*
0xe4 <rower>	seems to be some kind of wakeup. However, this can done just as well by any other command?.	e:row sends this to all rowers at program start.
0xe5 <rower>	same as e4	e:row sends this to active units on program close
0x93 <??>	something to do with place display during races, tests produce strange results sometimes. More testing necessary.	e:row does use this function, but results are not visible if the heart rate pickup is connected (and receiving signal)
0x96 <??>	no idea what this is supposed to do	e:row inserts 0x96 00 00 just after setup of race or workout
0xe2, 0xc3	forgot in which contexts I saw these	
0xc1, 0xc2	something to do with unit numbering	
<rower id>	every byte under 0x80 seems to be a command by itself. It returns what looks like a 4-byte integer, containing possibly the speed of the erg wheel. What units are used is not clear.	

pm2d autonomous commands

The autonomous commands deal with high level commands to the PM2 unit or units.

The specifications for these commands are being developed. Only some of them have been implemented in pm2d. Use this documentation only as an illustration of what functionality will soon be offered, because exact message layouts may change during the implementation process.

Design issues to be resolved

- Time stamping of results: Best would be to have each poller synchronized to a verified NTP time source. But what if one of them isn't? Maybe always use stamp of master poller?
- What to poll exactly, and how often? Obviously all strokes, but maybe meters in between? Even with SPM of 30, stroke data could be up to 2 secs old, so a close race might be reported the wrong way. --> Solution for now: poll stroke related data only after a stroke, and other data more often. Keep separate timers; one of last poll, one of last stroke.
- Should there be a distinct set of setup commands for autonomous mode? Or should setup be handled by the direct commands?

Retrieve report

This command retrieves data for a workout or race from pm2d.

pm2d format (command)

GET /pm2d-retrieve-report/ HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<%.0f meters>, <%d spm at stroke>, <%d stroke speed in watts>, <%d:%02d stroke speed in minutes:seconds>, <%d:%02d time at rower of last stroke>, <%d heart rate at stroke>

Notes

One line of output is generated for each stroke on each rower since the start of the workout. If the workout is not yet complete, data will be reported up to the last completed measurement.

Data is kept by pm2d until the start of a new workout. Also, the data are logged to the /var/log/pm2d directory after the completion of a workout.

Retrieve current workout data

This command retrieves data on a workout in progress from pm2d. Only the last measured values will be reported.

pm2d format (command)

GET /pm2d-retrieve-lastdata/ HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<meters %f>, <spm %d>, <watts %d>, <speed %d:%02d>, <time at rower of last stroke %d:%02d>, <time at rower %d:%02d>, <hr %d>

Notes

One line of output is generated for each rower.

Some data items (like spm, speed, etc) are reported as of the last measured stroke (at the reported rower time). Other items are current as of the last poll operation.

pm2d direct commands

This section contains descriptions of all commands implemented in pm2d. Every command description contains how to invoke the command, what result is returned, as well as a description of the interface between pm2d and the unit.

PM2 direct commands allow an interfacing program to let pm2d handle the serial interface. Instead of the complexity of serial communication, an interfacing program can use a simple(r) interface.

Most programs should, however, use the autonomous commands. *These commands offer an even higher level interface, in which pm2d handles not only the serial communication, but the control of PM2+ units from the start of a race or workout up to the end.*

Also, any command in the direct section causes pm2d to instantly send a command to a unit, and (depending on the command) wait for a response from the unit. Directly interfacing from a number of monitor programs to one pm2d controlling several PM2+ units, could easily overload the serial interface, the pm2d program, or the PM2+ units. Autonomous commands, however, do buffering of commands and thus cause much less interactions on the serial interface. Presentation programs should, for this reason, always use the autonomous command set.

If, for whatever reason, you decide to interface using direct commands, or even write your own serial handler based on the documentation included in this section, I would suggest you use the same output formats as pm2d does. If all of us standardise on these formats, we can all use each other's programs.

Every response is preceded by a HTTP header. The Date: field in the header contains the date and time stamp of the command run.

All replies for commands that return direct data have a common prefix of <poller id>,<rower id>. The amount of lines should equal the number of units under control of the pm2d that is processing the command.

Format descriptions are following the POSIX standard for printf format characters. Most used are:

%d	decimal
%02d	decimal 2 positions, left padded with leading zeroes as necessary
%f	floating point
%02x	hex 2 positions, left padded with leading zeroes as necessary

Typical command scenarios

Most setup commands are somewhat dependent on the order in which they are issued. The most useful scenario's are described here.

Scenario for starting a workout

lock unit

setup time OR setup meters, optionally setup split

lock unit

start timer on stroke

<polling> until time expired/meters limit reached

Scenario for interval workout

lock units

setup time OR setup meters

setup split interval to same value

lock unit

start timer on stroke

<polling> until time expired/meters limit reached

display interval number (must be before setup rest period)

setup rest period

<repeat from start for next interval, parameters can be changed>

Scenario for starting a race

<same as workout start>

<polling> check that all unit report zero values for meters, timers stopped.

display br 5 downto 0, in one second intervals

lock unit

start timer immediately

<polling> until time expired/meters limit reached

Polling

read meters

-depending on status byte from read meters, read stroke and/or read split

-watch status byte for other conditions, like end of workout.

every X seconds, read heart rate

every X seconds, read timer (if necessary)

read meters (0xb0)

This command will read the number of meters as it appears on the unit display.

serial format (command)

0xb0 <rower number>

serial format (reply)

<status byte> <reverse order short float containing meters>

Conversions necessary

To get the same format as on the PM2+ unit, truncate the number of meters after the decimal point (no rounding takes place on the PM2+ display).

pm2d format (command)

GET /pm2-read-meters HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<status byte %02x>,<meters %f>

Notes

pm2d reports the meters number unconverted. If you want to display the number exactly as it is on the unit's display, convert as described above. Note however e:Row reports 5 decimal places in the log files.

The number of meters is always counting up, even if some setting for countdown from a distance is active on the unit.

After a complete workout (eg. the status byte has the END_OF_WORKOUT bit set) the 0xb0 command will return the time (as measured by the PM2+ unit) for the complete workout.

read stroke data (0xb1)

This command reports the data from the last stroke. It should be issued only if the status byte in the 0xb0 read meters reply indicates that stroke data is ready to be collected.

serial format (command)

0xb1 <rower number>

serial format (reply)

<byte SPM value> <reverse order short float containing speed of last stroke >

Conversions necessary

SPM can be used directly. To get the same speed format as on the PM2+ unit, multiply by 500, add 0.5 for rounding. The resulting number is the number of seconds needed to row 500 meters. Divide by 60 to get minutes, take remainder to get seconds.

pm2d format (command)

GET /pm2-read-stroke HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<spm %d>,<minutes %d:seconds %02d>

Notes

none

read heart rate (0xb2)

This command reports the current heart rate as sampled by the heart rate pickup (this is an optional device you can connect to the PM2+ unit). If no pickup is connected, the reported value will be 0.

serial format (command)

0xb2 <rower number>

serial format (reply)

<2 bytes heart rate data>

Conversions necessary

To get the same format as on the PM2+ unit, divide 576000 by the number from the unit.

pm2d format (command)

GET /pm2-read-heartrate HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<heart rate %d>

Notes

If no heart rate monitor is connected to the PM2+, the reported number will be 0. If you do the division with 0, your program will probably bomb out.

read timer (0xb3)

This command reports the current timer on the PM2+.

serial format (command)

0xb3 <rower number>

serial format (reply)

<status byte> <reverse order short float number of seconds>

Conversions necessary

Truncate the number. Divide by 60 to get minutes, take remainder to get seconds.

pm2d format (command)

GET /pm2-read-timer HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<status byte %02x>,<minutes %d:seconds %02d>

Notes

e:Row does not seem to use this command at all.

This is one of the few commands documented by C2 (thanks!). The meaning of the status byte, however, is not clear (but it is not the same as the status byte on the b0 command).

read checksum (0xb4)

serial format (command)

0xb4 <rower number>

serial format (reply)

<4 bytes checksum data>

Conversions necessary

None, however, e:Row only reports the first 2 bytes in reverse ordering.

pm2d format (command)

GET /pm2-read-checksum HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<byte1>,<byte2>,<byte3>,<byte4>

Notes

Mine reports 3b, ae, 8d, 00. If yours reports something else, please inform me at mailto:sytsezelf@users.sourceforge.net.

read split (0xb6)

serial format (command)

0xb6 <rower number>

serial format (reply)

<1 byte spm average?><unknown byte><reverse order short float meters at end of split>

Conversions necessary

truncate the float to get same display value as on unit.

pm2d format (command)

GET /pm2-read-split HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,<%d spm average>,<%f meters>

Notes

The SPM value seems to be an average of the SPM during the split period.

The unknown byte is ignored.

lock unit (0xe0)

This command locks the PM2+ unit. None of the buttons will work after this command, apart from the display button (thus still allowing the rower to switch displays during a session). The main use for this command is to prevent accidental change to the unit while it has been set up for a race or workout.

serial format (command)

0xe0

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-unit-lock HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

unlock unit (0xc7)

This command unlocks the PM2+ unit. After this command, the state set by the lock unit command will be reverted. All races or workouts in progress are cleared, all setup items are canceled.

serial format (command)

0xc7

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-unit-lock HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

There are some states on the unit that will not be reset with this command. A symptom of such a state appears to be a – (double minus) sign in the strokes per minute field of the unit's display. So far, waiting a couple of minutes for the unit to shut itself down has cleared this for me.

start timer on stroke (0xc5)

If the unit has been set up with a setup (time or meters) command, the unit will not start responding if you start rowing. It will need a start timer command, either start timer on stroke (for a workout) or start timer immediately (for a race).

serial format (command)

0xc5

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-unit-starttimeronstroke HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

If you issue this command, the rower will reset the measured values for the current race or workout to zero.

start timer immediately (0xc6)

If the unit has been set up with a setup (time or meters) command, the unit will not start responding if you start rowing. It will need a start timer command, either start timer on stroke (for a workout) or start timer immediately (for a race).

serial format (command)

0xc6

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-unit-starttimerimmediate HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

setup meters for workout or race (0x94)

This command sets the meters counter for a workout or a race in the unit. Lock the unit before issuing this command. Start the timer with one of the timer commands (start timer on stroke, or start timer immediately).

serial format (command)

0x94 <short float containing number of seconds>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-setup-meters/<%d-formatted number of meters> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

If this command is issued, the display will show minor errors in the display. The meters field, top left, will contain the timer (but with the meters light on, and formatting for meters); the time field, bottom left, will display meters (but with the time light on, and timer formatting). This is corrected by sending a lock unit (0xe0) command.

Example

GET /pm2-setup-meters/2000 HTTP/1.1

will setup the unit for 2000 meters.

setup timer for workout or race (0x95)

This command sets the time for a workout or a race in the unit. Lock the unit before issuing this command. Start the timer with one of the timer commands (start timer on stroke, or start timer immediately).

serial format (command)

0x95 <short float containing number of seconds>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-setup-time/<%d formatted number of seconds> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

C2 software always seems to follow this command by a lock unit (0xe0). See the notes for setup meters.

Example

GET /pm2-setup-time/600 HTTP/1.1

will setup the unit for 10 minutes.

setup split time or meters (0x9a)

This command sets the split interval for a workout or a race in the unit. If setup meters has been issued, the split will be a number of meters. If setup time has been done, the split will be a time.

serial format (command)

0x9a <2 bytes reverse order number of meters or seconds>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-setup-split/<%d formatted number> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

Example

GET /pm2-setup-split/100 HTTP/1.1

will setup the unit for 100 meter splits, or 1:40 splits depending on the context.

setup rest time (0x97)

This command sets the rest time for a workout or a race in the unit.

serial format (command)

0x97 <reverse order short float number of seconds>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-setup-resttime/<%d formatted number of seconds> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

The command only works in context. First start a workout, row to finish it, then issue the setup rest time command. In this state, the display interval function also works.

Example

GET /pm2-setup-resttime/60 HTTP/1.1

will setup the unit for 60 seconds rest time.

display number in bottom right (0x99)

This command displays a number in the bottom right field (heart rate field) of the PM2+ LCD.

serial format (command)

0x99 <1 byte value to display>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-display-br/<%d formatted number> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

The command is used in the C2 software to display a countdown before the start of a race.

The number can be cleared from the display by the lock unit command, or the display place command.

Example

GET /pm2-display-br/5 HTTP/1.1

will put 5 in the display.

display place in bottom right (0x93)

This command displays a number in the bottom right field (heart rate field) of the PM2+ LCD. The difference with the 0x99 command is that the 'place' light is on in the field.

serial format (command)

0x93 <2 bytes> Exact meanings of the 2 bytes are not clear!

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

No command yet

pm2d format (reply)

n/a

Notes

0x93 0xff 0x80 will clear the place display.

0x93 <rank> <rower number> seems to work, but not every time?

If you have the heart rate pickup, and a transmitter is close to the pickup, the heart rate will overwrite the place number in the display. Disconnect the pickup, or make sure no transmitter is near to see the place display.

Example

GET /pm2-display-br/5 HTTP/1.1

will put 5 in the display.

display interval number (0x98)

This command displays the interval number in the top right field (SPM field) of the PM2+ LCD.

serial format (command)

0x98 <1 byte value to display>

serial format (reply)

none

Conversions necessary

n/a

pm2d format (command)

GET /pm2-display-int/<%d formatted number> HTTP/1.1

pm2d format (reply)

<poller id %d>,<rower id %d>,OK

Notes

Since this command applies to all rowers on a daisy-chained serial cable, the rower number does not apply. The reported rower number in the pm2d output will always be -1.

This command only works in context (see the notes with the setup rest time command. It will only display the number at the end of a (partial) workout. Also, at the end of a rest period, after a new work period has been started, the SPM value will automatically be displayed again.

Example

GET /pm2-display-int/5 HTTP/1.1

will put 5 in the display.