WISPR
Passive Acoustic Monitoring
Serial Command Interface

Hardware Version: WISPR V3.0

Embedded Ocean Systems

Manual Version: 1.1
October 2024

Contents

1. Introduction

The WISPR serial command interface is used to configure and control all critical PAM functionality. The command interface uses ASCII messages similar to NMEA strings. All messages begin with a dollar sign symbol ($) followed by a three character command identifier. Data fields follow comma (,) delimiters and are variable in length. An asterisk (*) delimiter and CRC value follow the last field of the message. The CRC value includes all characters in the message, including the commas between fields, but not including the $ and asterisk (*) delimiters. The command string is terminated with a CR or NL.

The 8-bit CRC result is appended to the message after the asterisk (*) as two ASCII hexadecimal characters (0–9, A–F). The most significant character of the CRC value appears first. Appendix A provides the C code used to calculate CRC values.

When a CRC value of 0xff is sent, WISPR will ignore CRC checking on the incoming message - for example `$RUN*ff`.

Unless otherwise specified the command interface uses UART1 and is configured for 9600 baud, no parity, 1 stop bit. No hardware handshaking is used.

1.1 Software Handshaking

Serial command software handshaking can be enabled or disabled using the ACK and NAK commands. Software handshaking is enabled by sending Wispr an ACK command and disabled by sending a NAK command.

If enabled, Wispr will respond to received commands with an ACK, NAK or CRC message. An ACK response indicates the message was received and properly interpreted. The system responds with a NAK if an error occurred in receiving the message or invalid command arguments are detected. The system will respond with CRC if the message CRC doesn't match (and is not equal to 0xff).

If software handshaking is disabled, the PAM system does not send an ACK or NAK when it receives a message. It will still send the CRC reply to indicate a CRC error.

For commands that request a response, the response string is sent after the handshaking acknowledgement.

2. Command Interface

2.1 Command Summary

Message	Function	Direction
RUN	Start data acquisition	Input
EXI	End data acquisition	Input
PAU	Pause data acquisition	Input
RST	Soff Reset of PAM processor	Input
SLP	Sleep for specified seconds	Input
TME	Set RTC time	Input
ADC	Set ADC parameters	Input
PGN	Set Preamp Gain	Input
STA	Request PAM status	Input/Output
SDF	Request SD card usage	Input/Output
SDS	Request SD card status	Input/Output
SDI	Request SD card information	Input/Output
PSD	Request power spectrum	Input/Output
RPI	Enable/Disable RPi power	Input/Output
VER	Send firmware version	Input/Output
VMP	Set vehicle state	Input
GPS	Set GPS information	Input
DBG	Set console message debug level	Input
ACK	Enable Software Handshaking	Input
NAK	Disable Software Handshaking	Input

2.2 RUN - Start data acquisition

The RUN command starts data acquisition logging.

Command Format: `$RUN*CRC`

Input Arguments: none.

Example: Start data acquisition

Controller to PAM:
```
> $RUN*ee
```

PAM to Controller:
```
< $ACK*ad
```

2.3 EXI - End Data Acquisition

The EXI command terminates data acquisition and logging, closes the current data, and puts the system into deep sleep (backup mode sleep). The system is restarted by cycling power. Optionally, the system can be configured to wake from deep sleep mode on input to UART0 or UART1, an RTC alarm, or an external alarm.

Command Format: `$EXI*CRC`

Input Arguments: none.

Example:

```
Controller to PAM:
> $EXI*a6

PAM to Controller:
< $ACK*ad
```

2.4 PAU – Pause Data Acquisition

The PAU command stops data acquisition and logging and put the system in light mode sleep (WFI sleep mode). This allows the operator to change the system configuration. Used typically to change the gain, synching RTC time, or changing data acquisition parameters. Data acquisition resumes after time is up or a RUN command is received.

Command Format: `$PAU,seconds*CRC`

Input Arguments:
seconds: Pause time in second

Example:

```
Controller to PAM:
> $PAU,5*59      // Pause for 5 seconds

PAM to Controller:
> $ACK*ad
```

2.5 RST – Reset Command

Soft system reset.

Command Format: `$RST*CRC`

Input Arguments: none.

Example:

```
Controller to PAM:
> $RST*33

PAM to Controller:
< $ACK*ad
```

2.6 SLP – Sleep Command

Sleep for specified number of seconds. Used for data acquisition duty cycling.

Command Format: `$SLP,seconds*CRC`

Input Argument:
seconds: Number of seconds to sleep

Example:

```
Controller to PAM:
> $SLP,60*cb   // Sleep for 60 seconds

PAM to Controller:
< $ACK*ad
```

2.7 STA – Request System Status

Request PAM system status message.

Command Format: `$STA*CRC`

Input Argument: none

Example:

Controller to PAM:
> $STA*00

PAM to Controller:
< $ACK*ad
< $PAM,1660743752,01,03,50000,3,0,14.418,1999224832,1936890880*31

Status response message format is:

```
"$PAM,%lu,%02x,%02x,%lu,%d,%d,%.3f,%lu,%lu*%x02",
 epoch, state, mode, sampling_rate, sample_size, gain, file_size, total, free, crc
```

epoch: Current time in epoch seconds
sampling_rate: Sampling rate of ADC in Hz
sample_size: Number of bytes per sample (2 or 3)
gain: Pre-amp gain (0,1,2,or 3)
file_size: File size in seconds
total: Total space of the active SD card (in 512 byte blocks)
free: Free space of the active SD card (in 512 byte blocks)

Mode:	Numeric Value	Description
DAQ	h01	Data acquisition ON, reading ADC
LOG	h02	Log data to SD cards
PSD	h04	Power spectral density estimation ON

State:	Numeric Value	Description
IDLE	h00	Idle
RUN	h01	Running
PAUSE	h02	Paused

2.8 TME – Set System Time

Set PAM system time.

Command Format: `$TME,seconds*CRC`

Input Arguments:
seconds: Seconds in Linux epoch time

Example: Set PAM time to 22/08/17 08:58:27

Controller to PAM:
*> $TME,1660726707*f1*

PAM to Controller:
< *$ACK*ad*

2.9 PSD – Request Spectrum

Request estimate of the average power spectral density. Returns fftsize/2+1 spectrum density bins in units of dB re: 1 mPa^2/Hz or dB re: 1 V^2/Hz if no pre-amp calibration data is available.

Command Format: `$PSD,seconds,fft_size*CRC`

Input Argument:
seconds = number of seconds to average
fft_size - Size of FFT used to estimate spectrum (32, 64, 128, 256, 512, or 1024)

Example: Compute a 30 second averaged PSD using an FFT of size 128.

Controller to PAM:
> $PSD,30,128*08

PAM to Controller:
< $ACK*ad

PAM response to Controller after 30 seconds:

```
$PSD,1660725557,128,50000,64,0,-125.6,-122.4,-126.3,-127.4,-
128.3,-129.1,-129.8,-130.2,-130.5,-130.6,-130.9,-131.1,-131.3,-
131.4,-131.7,-131.8,-131.8,-131.8,-131.8,-131.9,-132.0,-132.0,-
132.1,-132.2,-132.2,-132.2,-132.2,-132.3,-132.2,-132.3,-132.4,-
132.5,-135.4,-132.4,-132.5,-132.5,-132.6,-132.6,-132.9,-133.1,-
133.2,-133.7,-134.1,-134.8,-135.7,-136.6,-137.7,-138.9,-140.3,-
142.1,-144.2,-146.4,-149.1,-152.0,-155.3,-158.7,-162.1,-164.7,-
165.9,-166.1,-166.2,-166.2,-166.3,-166.4*78
```

PSD response message is formatted as:

```
$PSD,start_second,fft_size,sampling_rate,nbins,gain,
   psd[0],psd[1], … psd[nbins-1]*CRC

Where:
start_second:    Start time of PSD
fft_size:        Size of FFT (must be a pow 2 >= 8 and <= 1024)
sampling_rate:   ADC sampling rate in Hz
```

```
nbins:          Number of frequency bins in the PSD
gain:           ADC gain
psd:            Power spectral density value
```

2.10 ADC – Set ADC parameters

Set PAM ADC parameter. New ADC parameters go into effect immediately. If the system is running when this command is sent, it will pause, close the current data file, and restart data acquisition with the new parameters.

Command Format:
`$ADC,sample_size,sampling_rate,decimation,gain,file_size,timestamp*CRC`

Input Arguments:

sample_size:	Sample size in bytes (not currently used)
sampling_rate:	Sampling rate in Hz (1000 - 200000)
decimation:	Decimation factor used by oversampling ADC (4, 8 or 16)
gain:	Preamp Gain (0, 1, 2, or 3 corresponding to 0, 6, 12, or 18 dB)
file_size:	File size in seconds
Timestamp:	Buffer Timestamp size in bytes (0 or 6 bytes)

Note: Sample size is for future use only. Currently the sample size is fixed for each system.

Example: Set 24 bit samples, Sampling rate 50 KSPS, Decimation factor 8, Gain setting 3, File size 60 second, and timestamp size 6 bytes.

In the current firmware, the sample size is fixed at 2 or 3 bytes. Setting a new sample size has no effect and changing the sample size returns an error.

Controller to PAM:
```
> $ADC,3,50000,8,3,60,6*ff
```

PAM to Controller:
```
< $ACK*ad
```

2.11 SDF – Request Available Data Storage

The SDF command requests the amount of disk space on the system SD cards. Disk space is reported in units of blocks of size 512 bytes for each SD card installed.

Command Format: `$SDF*CRC`

Input Arguments: none.

Output Format:

```
$SDF,number_of_cards,
   ,card1_state,card1_total_storage,card1_free_storage
   ,card2_state,card2_total_storage,card2_free_storage
    ...
*CRC
```

Example: Response from system with 2 SD cards, SD card 1 total storage is 128 GB, and SD card 2 total storage is 1 TB.

```
Controller to PAM:
> $SDF*94

PAM to Controller:
< $ACK*ad
< $SDF,2,07,249671680,249554432,02,1999486976,1959678976*2b
```

SD card state code are defined as:

```
#define SD_STATE_UNKNOWN     0x00
#define SD_STATE_ENABLED     0x01
#define SD_STATE_CHECKED     0x02
#define SD_STATE_INIT        (SD_STATE_ENABLED |SD_STATE_CHECKED)
#define SD_STATE_MOUNTED     0x04
#define SD_STATE_OK          (SD_STATE_INIT | SD_STATE_MOUNTED)
#define SD_STATE_FULL        0x10
#define SD_STATE_ERROR       0x20
#define SD_STATE_UNFORMATED  0x40
```

2.12 SDS – Request SD Card Status

The SDS command requests the state of the SD cards installed in the system. State codes are defined above.

Command Format: `$SDS*CRC`

Input Arguments: none.

Output Format:

```
$SDS,number_of_cards,card1_state,card2_state, ... *CRC
```

Example:

```
Controller to PAM:
> $SDS*00

PAM to Controller:
< $ACK*ad
< $SDS,2,07,02*0d
```

2.13 SDI – Request SD Card Info

The SDI command requests the manufacturer info of the system SD cards.

Command Format: `$SDI*CRC`

Input Arguments: none.

Example: With 2 SD cards installed

```
Controller to PAM:
> $SDI*ff

PAM to Controller:
< $ACK*ad
< $SDI,0,07,WISPR_SD1,124868608,30,3,SD,SR128,1,86*6d
< $SDI,1,02,WISPR_SD2,999874560,30,3,SD,SR01T,1,86*b8
```

Output Format:

The SDI response string is formatted as:

```
sprintf(msg, "SDI,%d,%02x,%s,%lu,%02x,%d,%s,%s,%d,%x",
   n,state,label,capacity,hw_ver,mid,oid,pnm,highspeed,prv;
```

The fields are defined as:

```
n = card number - 1
state = card state (defined in SDF command)
```

```
label = card user label
capacity = user data area capacity in 1K Bytes
MID = 8 bit number which identifies manufacturer.
OID = 2-character ASCII string that identifies the card OEM.
PNM = The product name is a string, 5-character ASCII string.
HS  = High speed flag (1 or 0).
PRV = Product revision.
```

HW info is read from the card CID and CSD registers. See SD card specification for more information. For example:

Company	MID	OID
Panasonic	1	PA
Toshiba	2	TM
SanDisk	3	SD (PT)
Samsung	27	SM
Lexar	40	BE
Kingston	65	42
Transcend	116	JE (J)

2.14 PGN – Set Preamp Gain

Set Preamp Gain. New gain value goes into effect immediately.

Command Format: `$PGN,gain*CRC`

Input Arguments:
gain: Preamp gain of (0, 1, 2 or 3) corresponding to (0, 6, 12, or 18 dB)

Example: Set preamp gain to 1

Controller to PAM:
```
> $PGN,1*d6
```

PAM to Controller:
```
< $ACK*ad
```

2.15 VER – Request Firmware Version

Request the PAM system firmware version.

Command Format: `$VER*CRC`

Input Arguments: none

Example:

Controller to PAM:
`> $VER*ff`

PAM to Controller:
```
< $ACK*ad
< $VER,v1.5.0,Oct 14 2024,17:03:39*f7
```

2.16 VMP – Set Vehicle Main Processor State

Inform the PAM system of the vehicle state. This optional information is logged in the data file header to assist in post-processing of the PAM data. Units are defined by the user.

Command Format: `$VMP,sec,state,depth,heading,roll,pitch*CRC`

Input Arguments:

sec:	Time in seconds
state:	User defined vehicle state represented as a 16 bit unsigned integer,
depth:	Vehicle depth as a 16 bit unsigned integer
heading:	Vehicle heading as a 16 bit unsigned integer
roll:	Vehicle roll as a 16 bit unsigned integer
pitch:	Vehicle pitch as a 16 bit unsigned integer

Example:

Controller to PAM:
`> $VMP,`*`1660726707`*`,1,10,20,30,40*ff`

PAM to Controller:
```
< $ACK*ad
```

The user defined state variable is intended to provide information about the vehicle that may affect the PAM data. For example, a vehicle controller on a glider could tell the PAM system the dive modes, motor state, and other sensor states:

```
#define GMP_STATE_UNKNOWN 0
```

```
#define GMP_STATE_SURFACE 0x01
#define GMP_STATE_DESCENT 0x02
#define GMP_STATE_APOGEE  0x04
#define GMP_STATE_ASCENT  0x08
```

2.17 GPS – Set GPS position

Provide GPS information to the PAM system.

Command Format: `$GPS,sec,lat,lon,cog,sog*CRC`

Input Arguments:
sec: GPS time in seconds
lat: GPS latitude as decimal floating point number
lon: GPS longitude as decimal floating point number
cog Course over ground as decimal floating point number
sog: Speed over ground as decimal floating point number

Example:

Controller to PAM:
> $GPS,*1660726707*,10.9900,20.9999,1.0,2.0*ff

PAM to Controller:
< $ACK*ad

2.18 RPI – Turn RPi ON/OFF

Turn the power to the Rpi coprocessor ON or OFF.

Command Format: `$RPI,on*CRC`

Input Arguments:
On: On/Off flag - 1 = ON, 0 = OFF

Example:

Controller to PAM:
> $RPI,1*ff

PAM to Controller:
< $ACK*ad

2.19 DBG – Set debug message level

Set the level of debug messages sent to the system console. Debug messages are only printed to the console UART.

Command Format: `$DBG,level*CRC`

Input Arguments:
level: Debug message level from 1 to 4 with 1 being the lowest level messaging or the fewest messages.

Example:

Controller to PAM:
```
> $DBG,1*ff
```

PAM to Controller:
```
< $ACK*ad
```

APPENDIX A - CRC code:

```
unsigned short Calc_CRC(unsigned char *buf, unsigned short cnt)
      /* calculate CRC of buffer */
{
      unsigned int accum = 0;

      while( cnt-- )
      {
            accum = accum_crc( accum, *buf++ );
      }
      /* The next 2 lines forces compatibility with XMODEM CRC */
      accum = accum_crc( accum, 0 );
      accum = accum_crc( accum, 0 );
      return (unsigned short) ( accum >> 8 );
}


unsigned int accum_crc(unsigned int acmul, char ch)
      /* CRC accumulator */
{
      short i;

      acmul |= ch & 0xFF;
      for( i=0; i<8; i++ )
      {
            acmul <<= 1;
            if( acmul & 0x1000000L )  acmul ^= 0x102100L;
      }
      return acmul;
}
```