Pleo Monitor

Table of Contents

Introduction

The Pleo Monitor is a software module built into the Pleo firmware. It allows access to almost all Pleo functionality - including the current status of most subsystems within Pleo – from a standard terminal program running on a PC. For example, HyperTerminal or PuTTY on Windows, the Terminal on OS X or gtkTerm on Linux.

This document describes how to access the Monitor and what commands are available within the Monitor.

Document Revision History

Revision	Date	Comment
0.1		*Initial version*
0.5	*May 28, 2008*	*Some formatting changes. Additional content*
0.6	*May 30, 2008*	*Formatting changes*
0.7	*June 6, 2008*	*Add fs and resource commands*
0.8	*June 9, 2008*	*Additional clarifications and additions*
0.9	*July 3, 2008*	*Additional content, clarification*
1.0	*August 13, 2008*	*Add UART connector details*
1.1	*August 18, 2008*	*Fix xmodem and help commands*

Connecting to Pleo

There are two ways to physically connect Pleo to a PC:

- Through Pleo's USB port.
- Through Pleo's (hidden) serial UART port.

USB Connection

To use the USB connection, you need a standard USB cable (Standard A to mini B connector).

When Pleo is connected to a PC, the steps necessary to connect vary depending on what operating system is used:

Windows

Windows requires an .INF file to reference the proper device driver (see Appendix A Windows Pleo USB Driver).

When you connect Pleo via the USB connection to Windows, a dialog asks for a driver disk or CD. Select the ugobe.inf file provided by Ugobe to direct Windows to use the Microsoft USB to serial driver (usbser.sys), which is available on the original Windows installation CD.

When the driver is successfully located and installed, an additional Virtual COM port becomes available (see the Windows Device Manager to determine which COM port).

Additional information can be found in the Pleo 1.1 Updater documentation.

Mac OSX

Mac OSX is simply plug and play. When Pleo is plugged in, he will appear as a serial device in the /dev directory. You can use any third-party terminal program, but you can also use the default Terminal App that is included with OS X. For example:

```
<login>$ ls /dev/tty.*
/dev/tty.Bluetooth-Modem
/dev/tty.Z600-Dial-upNetworking-2
/dev/tty.Bluetooth-PDA-Sync
/dev/tty.Z600-SerialPort1-1
/dev/tty.usbmodem1d11

<login>$ screen /dev/tty.usbmodem1d11 115200
```

You will now receive all Pleo output directly at the Terminal.

To exit this mode, type:

```
'control-a' followed by a 'control-\'.
```

Linux

Linux requires a version of the kernel compiled with USB-serial support. Install a Linux-based terminal program, such as **gtkterm**. Connect Pleo to the USB port, and you should find a new device in the /dev tree with a name like **/**dev/**ttyACM*.** Use your terminal program to connect to this device.

Serial Connection

NOTE: This connection mechanism is not officially supported by Ugobe due to potential issues with electro-static discharge. This connection should only be used if for some reason the USB option is not suitable.

There is a direct UART connection available to the Atmel processor, which is connected to a seven pin header between Pleo's front legs. The matching connector part can be found via Digi-Key (http://www.digikey.com/), part number WM1725-ND. You will also need the connector terminals to solder to, part number WM1775-ND. And, to crimp the connector, part number WM9931-ND.

To use this with a PC, you need a converter to go from TTL levels to USB or serial. We have found the following to be work very well:

http://www.acroname.com/robotics/parts/S22-USB-SERIAL-INT-CONN.html

http://www.acroname.com/robotics/parts/S13-SERIAL-INT-CONN.html

This connection may also be used with a number of wireless modules, including Bluetooth and Zigbee. Descriptions of these connection types will be forthcoming; meanwhile, some information can be found on the Web.

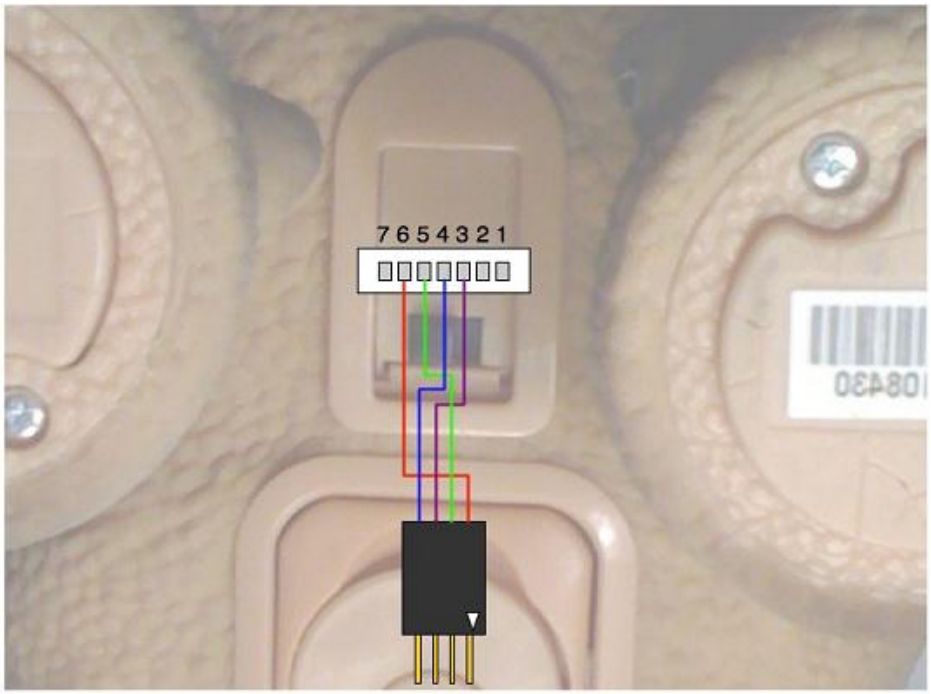

Figure 1: Wiring diagram of Pleo external UART based on Acroname Serial Interface Standard

Terminal Setup (Windows Example)

There are a number of different terminal programs available for Windows. This includes:

- Microsoft HyperTerminal: Note this is no longer included with Microsoft Vista. See http://www.hilgraeve.com/hyperterminal.html for an alternate version. We do not recommend the MS provided HyperTerminal since it does not handle the line lengths that the monitor outputs, and it's scroll back buffer is hard to use.
- PuTTY: http://www.chiark.greenend.org.uk/~sgtatham/putty/ PuTTY, though advertised as an SSH/Telmet client, also has support for serial connections.
- VanDyke CRT: http://www.vandyke.com/products/crt/index.html This is a commercial product which we use quite frequently internally at Ugobe.

When a connection to Pleo is established – via either USB or UART – it should be configured with the following settings:

Baud rate: 115200, Data bits: 8, Stop bits: 1, Parity: None, Flow control: None. Example screenshots follow.

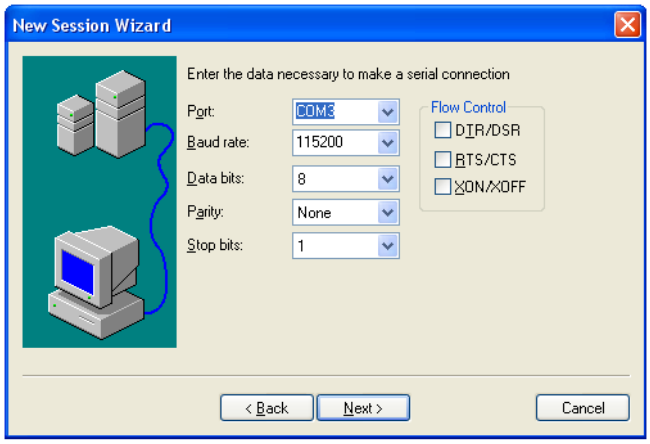

Figure 2: Serial Port setup in VanDyke CRT

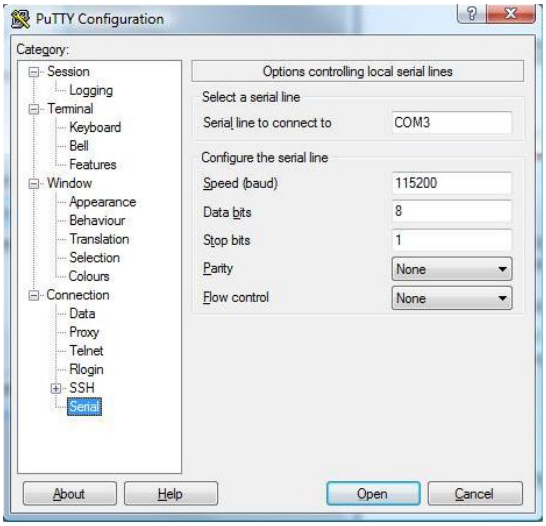

Figure 3: Serial Configuration from PuTTY

Using the Monitor

When the serial connection and serial port settings are configured, it's time to open up a terminal application and begin connecting to the Pleo Monitor. Open up your terminal application and turn Pleo on. In your terminal application you should see an output similar to what is displayed below.

NOTE: The output below is from a debug build of the firmware, which will likely differ radically from a release build.

The first few lines show the output of the Pleo boot loader. This is a special startup block that validates that the main Atmel firmware is valid. If so, then it will start it up. If not, it will wait for either an SD Card with a "pleo.img" firmware image, or a USB connection to initiate a DFU (device firmware upgrade) firmware transfer.

You will see our standard version string output for the boot loader, the main Atmel firmware, and the head NXP processor. A version string is the method by which we uniquely identify any specific build of a firmware or personality.

The format of the version strings is: <Subversion revision> <Build Date> <Build Time>

For example, this output shows the Pleo boot loader version as:

```
      Subversion revision number: 7361
      Build Date:  Nov 15 2007
      Build time: 21:34:31

Pleo boot loader
7361 - Nov 15 2007 21:34:31
Starting image 8007M - Mar  5 2008 17:27:23

8007 - Mar  5 2008 17:27:23
```

```
Reset Source = 0x00010001: User Reset 1; Brownout 0; Type 0 = Normal
INFO: low: motion: Initializing motion control subsystem
INFO: low: motion: MotorController 0 firmware version 28
INFO: low: motion: MotorController 0 comms test passed
INFO: low: motion: MotorController 0 A/D filter K=1 N=2
INFO: low: motion: MotorController 0 outputs set to P00=1, P01=0
INFO: low: motion: MotorController 1 firmware version 28
INFO: low: motion: MotorController 1 comms test passed
INFO: low: motion: MotorController 1 A/D filter K=1 N=2
INFO: low: motion: MotorController 1 outputs set to P00=1, P01=0
INFO: low: motion: MotorController 2 firmware version 28
INFO: low: motion: MotorController 2 comms test passed
INFO: low: motion: MotorController 2 A/D filter K=1 N=2
INFO: low: motion: MotorController 2 outputs set to P00=1, P01=0
INFO: low: motion: MotorController 3 firmware version 28
INFO: low: motion: MotorController 3 comms test passed
INFO: low: motion: MotorController 3 A/D filter K=1 N=2
INFO: low: motion: MotorController 3 outputs set to P00=1, P01=0
INFO: low: motion: Successfully loaded calibration data from DataFlash.
INFO: low: motion: ident | calibration  |  PID constants    |
INFO: low: motion: joint  low zero upper kp ki kd kf kni kdb
INFO: low: motion:     0  60  120   225 70 35  0  0  16   4
INFO: low: motion:     1  66   67   169 50 20  0  0  16   4
INFO: low: motion:     2  65  123   224 70 35  0  0  16   4
INFO: low: motion:     3  62   63   167 50 20  0  0  16   4
INFO: low: motion:     4  43  118   199 50 100  0  0  12   4
INFO: low: motion:     5  71  180   180 70 20  0  0  10   4
INFO: low: motion:     6  47  125   201 50 100  0  0  12   4
INFO: low: motion:     7  74  186   186 70 20  0  0  10   4
INFO: low: motion:     8 101  127   146 80 40  0  0  10   4
INFO: low: motion:     9  20  117   225 40 90  0  0   8   4
INFO: low: motion:    10  14  126   220 75 50  0  0  24   4
INFO: low: motion:    11  22  125   226 80 50  0  0  32   4
INFO: low: motion:    12  17  146   229 60 30  0  0  32   4
INFO: low: motion:    13  64  123   225 40 35  0  0   8   4
INFO: low: motion: Acroname MotorControllers present and functioning.
Enabling motion control system.
INFO: low: Rev 17.8; setting current sense gain to 9, vbat_kb = 24,
vbat_ki = 46
INFO: low: io: ts 937: eopen(/SYS_STATS.SSF, r)
INFO: low: io: ts 939: eclose()
INFO: low: res: stats: Statistics file read ok.
INFO: low: res: stats: CRC is ok!  Statistics file is good.
INFO: nxp: Resetting head; flash = 0
INFO: nxp: 7361 - Nov 15 2007 21:34:25
```

After the initial Pleo “startup” information, Pleo will continue to run. If there is an application on the SD Card or in the internal memory (DataFlash), it will execute and the output from the firmware and the application will output to your terminal.

You can press any key while in the terminal, and it will pause the execution of Pleo and wait for the entry of any command.

NOTE: While paused in the monitor, the low-level timer is still active. If there are any scripts that were waiting for time to pass, it may skip ahead more than expected when resumed.

To resume Pleo, press **Return** or **Enter**.

Now you can begin working with the Pleo Monitor commands. The first monitor command to learn is **help**. Typing **help** with no parameters will print all available commands or command groups. A command group – or module – is a related collection of commands under one heading. For example, 'motion' is not a command, but a group of commands consisting of 'motion play', 'motion stop', etc.

```
help
Monitor help screen:
  access <password>                          Change monitor access
level
  app                                        App commands
  audio                                      Audio commands
  camera <command>                           Camera commands
  cat <file>                                 Dump file contents
  cd <directory>                             Change current directory
  chdrv <drive letter>                       Set default drive (A or B)
  clear                                      Clear screen
  command <command>                          Motion and Sound Commands
  config                                     Display configuration
  copy <src> <dst>                           Copy files
  curdrv                                     print current directory
  fs <fs>                                    FS commands
  help [command]                             Print this help screen
  ir <cmd>                                   IR commands
  joint <command>                            Joint commands
  log <command>                              Logging commands
  ls                                         List SD card contents
  micfilter [on | off]                       ignore loudness &
direction
                                             if head, neck, or torso
moving
  monitor <command>                          Monitor commands
  motion <command>                           Motion commands
  passthru <Port0Baud> <Port1Baud>           Serial passthrough
  pkt                                        Packet commands
  property <command>                         Property commands
  pwd                                        Print working directory
  reset                                      Reset device
  resource                                   Resource commands
  sensor <command>                           Sensor commands
  sound <command>                            Sound commands
  stats [module]                             Display statistics
  time                                       Print device time.
  upgrade <file>                             Load a new image from SD
  vm <vm>                                    VM commands
>
```

To get even more detail on a command or command group, you can type **help <command>.** For example type **help motion**:

```
> help motion
Syntax: motion <command>
Motion help screen:
  csv                                    start/stop dumping csv data.
  dump                                   dump info about all motion
files.
  load <motion_file_num>                 load motion file; returns
motion handle.
  pause <motion_handle>                  pause a loaded motion file.
  run <motion_handle> <wait for joints>  run a loaded motion file;
optionally make it wait for joints to reach position.
  speed <percent playback speed>         set playback speed; 100 =
normal, 50 = 1/2 as fast.
  step <motion_handle>                   step a loaded motion file.
  stop <motion_handle>                   stop a loaded motion file.
```

Visibility

Depending on the user, we want to hide or show certain commands. For example, an end user should not be permitted to format the internal memory (DataFlash). Likewise, a third-party developer usually does not need access to some of the high-level commands.

Here we list each module, and the visibility of each. We use the following tags:

- *public*: these commands are left in for end users
- *protected*: these commands are left in the firmware, but hidden by default. They would not cause serious injury, but they are deemed necessary for the average user. A pass-phrase is needed to make them visible.
- *private*: these commands should only be available to Ugobe employees or trusted third-parties, since they can cause damage to Pleo or render a Pleo inoperable. A pass-phrase is required to unlock these commands.

Access to the various groupings is accomplished through the access command. Typing *access* followed by a password specific to that grouping will unlock that visibility level, and any level below.

If the current access level does not allow a certain command, then attempting to use that command will result in an error stating that that command is not found.

If you need access to a different level, it is best to set that level, use the command of interest, and then reset the level back to the lowest level.

Command Reference

Most of the monitor commands are organized into groups or modules. For example, the Sound Module contains a set of monitor commands that deal with sound files and playback.

We have listed all top-level commands having to do with the file system under a File System Commands section. This includes `cd`, `ls`, etc.

We have listed all top-level commands that are not part of any specific module under the Utility Commands section. This includes commands like `upgrade`, `passthru`, etc.

Application Module

The Application Module is a set of commands that deal with Pleo applications. Pleo applications are usually in the form of a Ugobe Resource File, identified by a file with an extension of .URF.

app autorun

The app autorun command determines whether an application will automatically execute when Pleo is powered up, or an SD Card is inserted into a powered on Pleo.

An application may be installed on the internal DataFlash, or located on an SD Card. In either case, if the autorun flag is set to false (0), then no application will be loaded. If the autorun flag is set true (1), then the normal startup procedure will take place. This includes first checking the SD Card for an application. If there is no application on the SD Card, then any application installed on the DataFlash will be executed.

This command is normally placed within an automon.txt file on an SD Card. This will prevent any application from loading, and does not require the use of the Monitor directly via a terminal.

Format

```
app autorun <0|1>
```

Arguments

0 = do not automatically execute an application.

1 = automatically execute an application.

Return values

None

Example

```
> app autorun 0
Now, if an SD Card is inserted with a valid .urf file, it will
NOT be executed, since the option to load has been turned off.
```

app load

The app load command will look for and execute a URF file if the current drive is the SD Card (A:), or it will load the built-in application from DataFlash (B:). Note the current drive can be changed via the chdrv command.

If no URF file is given to the command, then in Pleo 1.0 we will search for a “pleo.urf”. Whereas in Pleo 1.1 we will search for any URF other than “pleo.urf”.

Format

```
app load [urf_file]
```

Arguments

urf_file: this is an optional filename to use as the application.

Return values

None

Example

```
> app load myapp.urf
```

app unload

The app load command will unload the currently executing application, if any.

Format

```
app unload
```

Arguments

None

Return values

None

Example

```
>app unload
```

Audio Module

The audio module deals with capturing audio data from the microphones in the head and saving the data to memory. Sound data can be stored to SD (preferred, since it is much larger) or DataFlash (limited to 256k).

Since Pleo has binaural hearing, a stereo stream can be saved. But due to limits in transfer speed between the NXP ARM7 in the head and the Atmel ARM7 in the body (a 115k UART connection), some data loss may occur. It may be possible to increase the baud rate – see the pkt baud command – but this may also generate more noise in the system.

The speed of the destination media can also cause data loss. Saving to DataFlash is typically much slower than SD Card, so data loss normally occurs when saving to DataFlash. It is recommended that a high Class SD Card is used.

All sound data saved is in PCM format in a .WAV wrapper.

AUDIO AVERAGE

The audio average command will save a mono data stream that is an average of the left and right audio channels.

Format

```
audio average <filename> <seconds>
```

Arguments

filename: filename of wav format file to save. Note you should include the .wav extension since it will not be appended for you.

seconds: number of seconds of data to record.

Return values

None

Example

```
> audio average av.wav 5
attempting to record 5 seconds
Audio file av.wav opened
write_audio_data: 4 != 6
write_audio_data: 28 != 31
write_audio_data: 45 != 47
write_audio_data: 104 != 106
Audio file closed - wrote: 43296
>
```

Note in the example above the 4 != 6, 28 != 32, etc. represent errors in the data transfer between the head NXP processor – where the audio data is captured from the microphones – and the main Atmel processor. This is due to transferring more information than can be handled with the default 115200 baud connection between the processors.

audio both

The audio both command will save a stereo data stream that contains both left and right channel data.

Format

```
audio both <filename> <seconds>
```

Arguments

filename: filename of wav format file to save. Note you should include the .wav extension since it will not be appended for you.

seconds: number of seconds of data to record.

Return values

Example

```
>audio both both.wav 5
attempting to record 5 seconds
Audio file both.wav opened
write_audio_data: 52 != 54
write_audio_data: 78 != 81
write_audio_data: 117 != 119
Audio file closed - wrote: 44000
>
```

audio left

The audio left command will save a mono data stream that contains data only from the left microphone.

Format

```
audio left <filename> <seconds>
```

Arguments

filename: filename of wav format file to save. Note you should include the .wav extension since it will not be appended for you.

seconds: number of seconds of data to record.

Return values

None

Example

```
> audio left left.wav 5
attempting to record 5 seconds
Audio file left.wav opened
Audio file closed - wrote: 46464
>
```

audio right

The audio right command will save a mono data stream that contains data only from the right microphone.

Format

```
audio right <filename> <seconds>
```

Arguments

***filename**: filename of wav format file to save. Note you should include the .wav extension since it will not be appended for you.*

***seconds**: number of seconds of data to record.*

Return values

None

Example

```
> audio right right.wav 5
attempting to record 5 seconds
Audio file right.wav opened
Audio file closed - wrote: 46464
>
```

AUDIO TERMINAL

The audio terminal command will cause the audio data captured with any of the other audio commands to be sent out the monitor as binary data.

Note the data is NOT saved to the filename passed in the other commands.

Format

```
audio terminal <"on"|"off">
```

Arguments

on: turn on the redirection to monitor

off: turn off redirection to the monitor

Return values

None

Example

```
> audio terminal on
INFO: monitor: Audio data now sent to terminal.
> audio terminal off
INFO: monitor: Audio data no longer sent to terminal.
>
```

Camera Module

The camera module provides commands to interact with the camera located in Pleo's nose. This includes capturing images, sending configuration settings to the camera, and getting information related to the color tracking.

Note that the NXP ARM7 processor in Pleo's head is responsible for image capture and analysis, and the high-level information is then passed to the main Atmel ARM7 processor in Pleo's body. They are connected via a relatively low bandwidth UART connection. For this reason we cannot stream live video or audio from Pleo.

The camera in Pleo is set up by default to capture images in QCIF (Quarter CIF, see: http://en.wikipedia.org/wiki/Common_Intermediate_Format) format, which is 176x144 pixels.

The coordinate system of the camera is such that [0,0] is the upper-left corner and [175,143] is the lower-right.

camera average

The camera average command will return the full frame YUV average as an INFO NXP message. Appropriate log levels should be enabled to see these messages.

Format

```
camera average
```

Arguments

None

Return values

None

Example

```
> camera average
INFO: monitor: full frame average requested
INFO: nxp: Camera Average Data: Y=127, U=135, V=124
```

camera capture

The camera capture command will save a frame of camera data to a file. Note that due to bandwidth issues, the image may actually be made up of data that spans multiple frames. When this command is issued, the SRAM in the Pleo head is used as a buffer for that image data. This data can be retrieved again if the power is not removed from Pleo.

Format

```
camera capture <filename> [bmp|raw] [new|last]
```

Arguments

filename: filename to store image data. Note that the proper file extension should be part of this name – it is not added automatically.

bmp: store image data into a .BMP formatted file. This is the default.

raw: store image data directly to the file, with no header.

Note: Raw is not the normal photographic RAW format, but just a raw data dump. Image processing applications will not know what to do with this file; it is raw YUVY data at 4 bytes per 2 pixels.

new: obtain new data from the camera

last: re-get the image data that is cached in the SRAM

Return values

None

Example

```
> camera capture test.bmp
Camera image requested from SRAM
Starting camera system...Done.
Capturing image...
...End... ?
```

CAMERA COLOR

The camera color command will change the cameras output to the given color space.

Note: This command is obsolete and unsupported in the 1.1 version of the software.

To change the camera color mode, use the "pkt send" command with the ce, ce1, ce2, ce3, ce4 and ce5 arguments.

FORMAT

```
camera color <mono | yuv | rgb>
```

ARGUMENTS

mono: use only the luminance data

yuv: instruct the camera to output data in YUV format

rgb: instruct the camera to output data in RGB format

RETURN VALUES

None

EXAMPLE

```
> camera color rgb
cam_color_cmd
```

CAMERA CONFIG

The camera config command will configure the current mode of the camera interface code in the NXP processor. This mode determines the data format and the kind of statistics that are gathered during the capturing process.

Note: This command is obsolete and unsupported in the 1.1 version of the software.

FORMAT

```
camera config <mode> <format> <gm> <w> <h> <x> <y> <pw> <ph>
```

ARGUMENTS

mode: 0=idle, 1=full frame, 2=frame average, 3=Y histogram, 4=red histogram, 5=green histogram, 6=blue histogram

format: 1=Y, 2=YUV, 3=RGB

grid: grid mode

w, h, x, y: window dimensions

pw, ph: partition dimensions.

RETURN VALUES

None

EXAMPLE

```
> camera config 0
```

camera getobjects

The camera getobjects command retrieves the current set of color objects that are being tracked by the color tracking algorithm executing on the NXP ARM7.

Note: This command is obsolete and unsupported in the 1.1 version of the software.

Format

```
camera getobjects
```

Arguments

None

Return values

Returns a list of objects being tracked, and their dimensions in the current frame

Example

```
> camera getobjects
     ##  x1  y1  x2  y2
     01  32  43  132 109
>
```

CAMERA TERMINAL

The camera terminal command redirects the output of the camera capture command to the monitor. This may be used by host applications to capture image data directly through the monitor.

Note: This command is obsolete and unsupported in the 1.1 version of the software.

FORMAT

```
camera terminal <"on"|"off">
```

ARGUMENTS

on: turn on monitor redirection

off: turn off monitor redirection

RETURN VALUES

None

EXAMPLE

```
> camera terminal on
INFO: monitor: Video data now sent to terminal.
> camera terminal off
INFO: monitor: Video data no longer sent to terminal.
>
```

CAMERA WINDOW

The camera window command sets the currently active window.

Note: This command is obsolete and unsupported in the 1.1 version of the software.

FORMAT

```
camera window <x> <y>
```

ARGUMENTS

x: horizontal distance from the center of the image that should be active

y: vertical distance from the center of the image that should be active

RETURN VALUES

None

EXAMPLE

```
> camera window 20 10
cam_window_cmd
>
```

DataFlash Module

The DataFlash module provides a group of commands that deal with the DataFlash in Pleo. This is an Atmel AT45DB321D, a 32Mbit (4MB) part. See here: http://www.atmel.com/dyn/products/product_card.asp?part_id=3818

The DataFlash is partitioned into three sections for our use:

Sectors 0-1: reserved for calibration, serial number and boot loader backup storage

Sectors 2-55: reserved for default Pleo personality URF

Sectors 56-65: reserved for an available file system

```
> help df
Syntax: df <cmd> <args>...
DF area help screen:
  boot                  Load booter image and jump to it.
Doesn't WORK!
  format                Format current device. Destructive!
  save <booter image>   Put booter image in reserved area of DF
  test r|rw|w           Perform DF transfer tests.
```

df boot

The df boot command will load a previously saved boot loader from the reserved section of DataFlash and execute it.

Format

```
df boot
```

Arguments

None

Return values

None

Example

```
> df boot
```

df format

The df format command will do a file system level format of the file system portion of the DataFlash.

Format

```
df format
```

Arguments

None

Return values

None

Example

```
> df format
format returned: F_NOERR
> chdrv b
> ls
No files in directory: *
>
```

df save

The df save command is designed to store the image of the boot loader in a reserved section of the DataFlash. Currently, this is pages 2-17, which is within the first sector, after the calibration data.

Format

```
df save <boot_loader_file>
```

Arguments

boot_loader_file: this is the Pleo boot loader binary, usually "pleoboot.bin".

Return values

None

Example

```
> df save pleoboot.bin
```

DF TEST

The df test command will perform reads from DataFlash, writes to DataFlash or reads and writes to DataFlash in order to measure the performance.

Note: When this command is executed, the test will run until interrupted by another key press.

Format

```
df test <r|rw|w>
```

Arguments

r: perform read test

rw: perform read and write test

w: perform write test

Return values

None

Example

```
> df test r
bytes transferred: read=1507968 write=528
elapsed time: 2447
> df test rw
bytes transferred: read=2442000 write=2442000
elapsed time: 5615
> df test w
bytes transferred: read=0 write=29091216
elapsed time: 19870
>
```

File System Commands

The file system module is a logical grouping of all commands related to the file systems within Pleo.

There are two ‘drives’ – or ‘devices’ – in Pleo with respect to the file system. The SD Card is known as drive ‘a’ and the internal DataFlash memory is drive ‘b’. This model is taken directly from the old DOS days. So, to refer to a file on the SD Card, you may use something like: “a:/test.txt”. To refer to a file on the internal DataFlash, you would use something like “b:/test.txt”.

Path separators should always be ‘/’. For example, a:/dir/test.txt. To refer to files at the root of a drive, you should use drive:/name.ext. For example, a:/automon.txt.

Unlike the other modules in the monitor, most of these commands do not use any command group naming, but are available directly. For example, ‘motion play’ versus ‘ls’.

CAT

The cat command dumps file contents to the monitor. This should be used only on text files - binary files will likely result in odd behavior in the terminal program, since it will be attempting to translate this data as terminal commands.

However, if you are using a host program to communicate directly with Pleo through the monitor, this may be a way to transfer binary files from Pleo to a host. This is not officially supported or tested though.

Format

```
cat <file_name>
```

Arguments

file_name: file to dump to monitor.

Return values

None

Example

```
> cat automon.txt
log disable all
>
```

cd

The cd command allows the user to change the current working directory.

Format

```
cd <directory_name>
```

Arguments

directory_name: name of directory to change to. This may be the special value ".." which indicates parent directory, or "." which indicates current directory.

Return values

None

Example

```
> mkdir one
mkdir returned: F_NOERR
> cd one
chdir returned: F_NOERR
> pwd
pwd returns: /one
> cd ..
chdir returned: F_NOERR
> pwd
pwd returns: /
>
```

chdrv

The chdrv command allows the user to change the current working drive or device.

Format

```
chdrv <drive>
```

Arguments

drive: drive to change to. SD Card is “a” and DataFlash is “b”.

Return values

None

Example

```
> curdrv
Current drive is SD Card (A:)
> chdrv b
> curdrv
Current drive is DataFlash (B:)
> chdrv a
> curdrv
Current drive is SD Card (A:)
>
```

COPY

The copy command copies files from the source directory to the destination directory. See the introductory material in this section on how to format path names.

Format

```
copy <src_path> <dst_path>
```

Arguments

src_path: the full path to the file to copy

dst_path: the full path of the destination

Return values

None

Example

```
> copy a:left.wav b:left2.wav
INFO: low: io: ts 1184260: eopen(left.wav, r)
INFO: low: io: ts 1184271: eopen(left2.wav, w)
INFO: low: io: ts 1185694: eclose()
INFO: low: io: ts 1185746: eclose()
>
```

curdrv

The curdrv command displays the current drive, or device. This command is commonly used in conjunction with the chdrv command. For example, you can display the current drive, change to the b drive and verify the current drive.

Format

```
curdrv
```

Arguments

None

Return values

None

Example

```
> curdrv
Current drive is SD Card (A:)
> chdrv b
> curdrv
Current drive is DataFlash (B:)
> chdrv a
> curdrv
Current drive is SD Card (A:)
>
```

fs close

The fs close command will close the file previously opened with the fs file command.

Format

```
fs close
```

Arguments

None

Return values

None

Example

```
> fs close
INFO: low: io: ts 1986866: eclose()
```

fs file

The fs file command opens an existing file or creates a new file. Note only one file via this command can be open.

Format

```
fs file <name> [access]
```

Arguments

Name: name of file to open or create

Access: access mode. One or more of: 'write' or 'binary'. Default mode is 'read' and 'text'

Return values

None

Example

```
> fs file a.txt
INFO: low: io: ts 780807: eopen(a.txt, r)
```

fs flush

The fs flush command will write to DataFlash or SD Card any data that has not yet been written to any open files in the file systems.

Format

```
fs flush
```

Arguments

None

Return values

None

Example

```
> fs flush
```

ls

The ls command displays a file listing for the current directory on the current drive.

Format

```
ls [path]
```

Arguments

path: path is an optional source for the ls command. If not given, the current drive and directory will be used.

Return values

None

Example

```
> ls b:
---rw-x     1158 SYS_STATS.SSF
---rw-x    19240 pleo.urf
---rw-x    46508 left2.wav
3 file(s) with total size 66906
> ls a:
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 one
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax     2135 init.amx
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
10 file(s) with total size 311940
> ls a:/one
--drw-x        0 one
1 file(s) with total size 0
> ls a:/one/*
--drw-x        0 .
--drw-x        0 ..
2 file(s) with total size 0
>
```

pwd

The pwd command prints the current directory. Commonly used in conjunction with the cd command.

Format

```
pwd
```

Arguments

None

Return values

None

Example

```
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 one
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax     2135 init.amx
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
10 file(s) with total size 311940
> pwd
pwd returns: /
> cd one
chdir returned: F_NOERR
> pwd
pwd returns: /one
>
```

mkdir

The mkdir command creates a new directory with the given name in the current directory.

Format

```
mkdir <directory_name>
```

Arguments

directory_name: name of new directory to create.

Return values

None

Example

```
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax     2135 init.amx
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
9 file(s) with total size 311940
> mkdir two
mkdir returned: F_NOERR
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 two
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax     2135 init.amx
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
10 file(s) with total size 311940
>
```

rm

The rm command removes, or deletes, files from the current directory.

Note that the access command must first be used to set the highest access level since deleting files can be a dangerous operation.

Format

```
rm <file_name>
```

Arguments

file_name: name of file to remove.

Return values

None

Example

```
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 two
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax     2135 init.amx
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
10 file(s) with total size 311940
> rm init.amx
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 two
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
9 file(s) with total size 309805
>
```

RMDIR

The `rmdir` command removes, or deletes, directories.

Format

```
rmdir <directory_name>
```

Arguments

directory_name: name of directory to remove.

Return values

None

Example

```
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
--drw-x        0 two
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
9 file(s) with total size 309805
> access 3
> rmdir two
rmdir returned: F_NOERR
> ls
---rwax        0 log.txt
---rwax      337 propnull
---rwax     2443 proptest
---rwax      581 int_test
---rwax    46508 left.wav
---rwax        0 test.bmp
---rwax   240696 pleo.img
---rwax    19240 pleo.urf
8 file(s) with total size 309805
>
```

xcopy

The xcopy command is used to recursively copy files and directories from a source location to a destination location. Currently, we can only copy to a directory depth of two.

Format

```
xcopy <src> <dst>
```

Arguments

src: source path to use

dst: destination of files

Return values

None

Example

```
> xcopy ...
```

IR Module

The IR module includes commands to send, receive and set parameters related to the IR transmitter and receiver located in Pleo's nose.

ir beacon

ir object

ir send

ir watch

IR BEACON

The ir beacon command sets the configuration parameters for the IR beacon. The beacon will be transmitted continuously (about once a second) if the tx_enable argument is 1. The on_sensor() function in the sensor VM will receive a SENSOR_BEACON message whenever it receives another Pleo's beacon. Reception of another Pleo's beacon will also be logged and displayed in the monitor.

Note that the data bytes 1-3 are the user data payload, and can be used for any purpose, though Pleo's built-in personality (PM) will have its own uses for these bytes. Setting of the beacon payload via the monitor is not very useful unless the PM is unloaded, preventing it from overwriting your settings.

FORMAT

```
ir beacon <rx_enable> <tx_enable> <id> <data1> <data2> <data3>
```

ARGUMENTS

rx_enable: enable reception of IR beacon message

tx_enable: enable transmission of IR beacon message

id: set the ID to use for this Pleo. Defaults to one byte of serial number

data1: user-definable data byte 1

data2: user-definable data byte 2

data3: user-definable data byte 3

RETURN VALUES

None

EXAMPLE

```
> ir beacon 1 1 32 9 8 7
```

IR OBJECT

The ir object command enables IR object detect using a given transmit code and sets the required accuracy. Passing no arguments to this command will report the current settings.

Format

```
ir object [tx_code] [percent]
```

Arguments

tx_code: the new IR code to use for transmission

percent: the required percentage of return data to enable a trigger on the SENSOR_IR sensor.

Return values

None

Example

```
> ir object
IR Object Detector txen = 1, percent = 100, sensor enable = 1
> ir object 88 90
INFO: low: object tx=1, speed=90
Set IR Object Detector txen = 88, percent = 90
>
```

IR SEND

The ir send command will send an arbitrary string out the IR port.

Format

```
ir send <string>
```

Arguments

string: the string to send out the IR port

Return values

None

Example

```
> ir send one
Sending string over IR: one
INFO: low: Pleo IR char 'o' (0x6F) received.
INFO: low: Pleo IR char 'e' (0x65) received.
INFO: low: Pleo IR char ' ' (0x00) received.
```

IR WATCH

The ir watch command will watch for any IR data and print it to the monitor.

Format

```
ir watch
```

Arguments

None

Return values

None

Example

```
> ir watch
Received IR line (from previous irsend): oe :
  Hex: 0x6F 0x65
Received IR line (from previous irsend): on :
  Hex: 0x00 0x00
Received IR line (from previous irsend): one :
  Hex: 0x03 0x00 0x00
```

Joint Module

The joint module allows the moving of joints, getting joint status, viewing calibration data, saving calibration data, and setting low-level PID values for the Toshiba firmware.

The possible joint values are:

```
Joints:
   0 J_RIGHT_SHOULDER        1 J_RIGHT_ELBOW           2
J_LEFT_SHOULDER
   3 J_LEFT_ELBOW            4 J_LEFT_HIP              5
J_LEFT_KNEE
   6 J_RIGHT_HIP             7 J_RIGHT_KNEE            8 J_TORSO
   9 J_TAIL_HORIZONTAL      10 J_TAIL_VERTICAL        11
J_NECK_HORIZONTAL
  12 J_NECK_VERTICAL        13 J_HEAD                 14
J_TRANSITION
  15 J_SOUND
```

Note: Joints 14 and 15 are 'virtual' joints; that is, they do not represent real physical joints. So the majority of the joint commands do not apply to these two.

Note: The co-ordinate system of Pleo is such that 0 degrees is defined as neutral, neutral being straight down, straight ahead, and straight back. Positive angle values move a joint forward, to the right, or up. And negative angle values move a joint backwards, to the left, or down. Left and right are from Pleo's perspective.

JOINT ANGLE

The joint angle command moves a specific joint to a specific location, measured in degrees, in a specific amount of time.

Format

```
joint angle <joint_id> <angle> [time_out] [alloted_time]
```

Arguments

joint_id: ID of the joint to move

angle: angle in degrees to move the joint

time_out: maximum amount of time to wait for the move to complete; set to 0 to wait forever or until a keypress on the monitor

allotted_time: the amount of time from now, in milliseconds, at which point the motion should complete; a time value of 0 will cause it to move as fast as it can; if omitted, the allotted_time value is automatically set to be the value which would move the joint from its current angle to the requested angle at a speed of 180 degrees per second

Return values

Prints continuous status to the monitor output

Example

```
> joint angle 12 90
WARN: monitor: Clipped angle to 75
j12 a-22 39271 ms J_ST_RUNNING (1)
j12 a-22 39336 ms J_ST_SLOW (5)
j12 a-22 39400 ms J_ST_RUNNING (1)
j12 a-10 39784 ms J_ST_SLOW (5)
j12 a70 42504 ms J_ST_STOPPED (2)
Joint moved to 70 in 3233 ms
INFO: monitor: Max vel = 42 degrees/sec, max accel = 193
degrees/sec^2
>
```

joint disable

The joint disable command will disable a specific joint, preventing it from responding to any movement commands – from the monitor or motions – until enabled again.

Format

```
joint disable <joint_id>
```

Arguments

joint_id: joint ID of the joint to disable. See the table at the beginning of this section.

Return values

None

Example

```
> joint disable 7
INFO: low: motion: global joint 7: controlled by script
>
```

JOINT ENABLE

The joint enable command will enable a specific joint, allowing it to respond to any movement commands – from the monitor or motions – until disabled.

Format

```
joint enable <joint_id>
```

Arguments

joint_id: joint ID of the joint to enable

Return values

None

Example

```
> joint enable 7
INFO: low: motion: global joint 7: controlled by motion file
>
```

JOINT INFO

The joint info command prints the configuration information for a specific joint, or all joints.

Format

```
joint info [joint_id]
```

Arguments

joint_id: ID of joint to print. If omitted, print all joints

Return values

All current configuration information for the requested joint.

Example

```
> joint info 7
                 |   degrees      | degrees   | motor units   |   raw VR     |
Joint Servo TSB Ch AMin ANeut AMax  FEP NEP SEP PMin PNeut PMax LoLm Neut HiLm Rev Name
-------------------------------------------------------------------------------------
    7     9   2  2  -90     0    0   0   0 -90   10   245  245   74  186  186   0
J_RIGHT_KNEE
>
```

JOINT MOVE

The joint move command will move a specified joint to a specified angle. The monitor will immediately return to the prompt; that is, the monitor will NOT wait for the joint to finish the move. This differs from the joint angle command in that the joint angle command will continuously print status information to the monitor.

Format

```
joint move <joint_id> <angle>
```

Arguments

joint_id: ID of the joint to move

angle: position in degrees to move the joint to

Return values

None

Example

```
> joint move 7 90
>
```

JOINT NEUTRAL

The joint neutral command will enumerate all joints and move them to their neutral – or 0 degree – positions.

Format

```
joint neutral [timeout] [angle] [quiet]
```

Arguments

timeout: how long to wait for each joint to reach neutral. If not given, will use the default.

angle: override angle to use instead of 0

quiet: do not print status messages to the monitor

Return values

Prints status of each joint move

Example

```
> joint neutral
j0 a-1 735853 ms J_ST_STOPPED (2)
Joint moved to -1 in 0 ms
j1 a0 735853 ms J_ST_STOPPED (2)
Joint moved to 0 in 1 ms
j2 a-1 735854 ms J_ST_STOPPED (2)
Joint moved to -1 in 0 ms
j3 a0 735855 ms J_ST_STOPPED (2)
Joint moved to 0 in 0 ms
j4 a-1 735855 ms J_ST_STOPPED (2)
Joint moved to -1 in 0 ms
j5 a-2 735856 ms J_ST_STOPPED (2)
Joint moved to -2 in 0 ms
j6 a0 735856 ms J_ST_STOPPED (2)
Joint moved to 0 in 1 ms
j7 a-2 735858 ms J_ST_RUNNING (1)
Joint moved to -2 in 0 ms
j8 a-1 735870 ms J_ST_STOPPED (2)
Joint moved to -1 in 0 ms
j9 a-2 735870 ms J_ST_STOPPED (2)
Joint moved to -2 in 1 ms
j10 a0 735871 ms J_ST_STOPPED (2)
Joint moved to 0 in 31 ms
j11 a-2 735903 ms J_ST_STOPPED (2)
Joint moved to -2 in 0 ms
j12 a-2 735904 ms J_ST_RUNNING (1)
Joint moved to -2 in 0 ms
j13 a0 735905 ms J_ST_STOPPED (2)
Joint moved to 0 in 0 ms
```

```
All joints set to angle 0.
>
```

JOINT PID

The joint pid command sets a joint's PID constants. PID is an abbreviation of Proportional Integral Differential, the type of motion control loop used for Pleo's joints. It is not recommended that users modify the PID constants.

Note that if only the joint_id is specified, and all other arguments are omitted, the joint pid command will display the current PID values for the specified joint. If the joint_id is also omitted, then the command will display the current PID values for all joints.

FORMAT

```
joint pid <joint_id> [<proportional> <integral> <differential>
<feedforward> <integral-len>]
```

ARGUMENTS

joint_id: the joint to query or modify

proportional: a numeric value between 0 and 255; it is multiplied by the current error in position, the result of which is added to the PWM output command to the motor

integral: a numeric value between 0 and 255; it is multiplied by the sum of the last <integral_len> number of position errors, and then added to the PWM output command

differential: a numeric value between 0 and 255; it is multiplied by the rate if change (derivative) of the position error, then added to the PWM output command

feedforward: a numeric value between 0 and 255; it is multiplied by the commanded velocity then added to the PWM output command

integral-len: a numeric value between 0 and 48; it sets the number of previous position errors being accumulated for the integral term

RETURN VALUES

This will return the current PID values if requested by omitting all arguments after the joint_id.

Example

```
> j pid
PID constants:
joint P  I  D  F  NI name
   0  70  35   0   0  16 (J_RIGHT_SHOULDER)
   1  50  20   0   0  16 (J_RIGHT_ELBOW)
   2  70  35   0   0  16 (J_LEFT_SHOULDER)
   3  50  20   0   0  16 (J_LEFT_ELBOW)
   4  50 100   0   0  12 (J_LEFT_HIP)
   5  70  20   0   0  10 (J_LEFT_KNEE)
   6  50 100   0   0  12 (J_RIGHT_HIP)
   7  70  20   0   0  10 (J_RIGHT_KNEE)
   8  80  40   0   0  10 (J_TORSO)
   9  40  90   0   0   8 (J_TAIL_HORIZONTAL)
  10  75  50   0   0  24 (J_TAIL_VERTICAL)
  11  80  50   0   0  32 (J_NECK_HORIZONTAL)
  12  60  30   0   0  32 (J_NECK_VERTICAL)
  13  40  35   0   0   8 (J_HEAD)
>
```

JOINT RANGE

The joint range command will move a joint back and forth between two different positions a given number of times.

Format

```
joint range <joint_id> [loop_count] [min_angle] [max_angle]
```

Arguments

joint_id: ID of joint to print. If omitted, print all joints

loop_count: how many loops to perform. Defaults to 5

min_angle: starting angle for loop. Defaults to min possible angle for the joint

max_angle: ending angle for loop. Defaults to max possible angle for the joint

Return values

Prints status information as it progresses

Example

```
> joint range 7
Range checking joint 7 using 5 loops between -90 and 0.
Joint 7 min angle -70, max angle -2, speed 40 degrees/sec, max
speed 91, max accel 634, time for full swing 1670 msec
```

joint show

The joint show command prints current status and configuration data for any joint or all joints.

Format

```
joint show [joint_id]
```

Arguments

joint_id: ID of joint to print. If omitted, print all joints

Return values

All current status information for the requested joint.

Example

```
> joint show 7
            Low  Neutral  High
Joint Servo Limit Position Limit Dir Angle En Time  Err Pos  SetP  Vel  Accel PWM  Load
DSTime Move Offs Status      Name
-------------------------------------------------------------------------------------------
    7     9   -90     0      0     0  -2   1 0000   -2  243  245     0     0    0  0000
25    0    0 J_ST_STOPPED J_RIGHT_KNEE
>
```

JOINT TARGET

The joint target command will move a specified joint to the position given in motor units.

Format

```
joint target <joint_id> <target>
```

Arguments

joint_id: ID of joint to move

target: motor unit value (0-255) to move this joint

Return values

None

Example

```
> joint target 7 200
>
```

JOINT WATCH

The joint watch command will continuously monitor and print the current status of a joint.

Format

```
joint watch <joint_id>
```

Arguments

joint_id: ID of joint to watch

Return values

Continuously prints status lines, terminated by a LF (‘\r’) character.

Example

```
> joint watch 7
            Low  Neutral  High
Joint Servo Limit Position Limit Dir Angle En Time  Err Pos  SetP  Vel  Accel PWM  Load
DSTime Move Offs Status      Name
-------------------------------------------------------------------------------
   7     9   -90     0      0     0 -17   1 0000    2  202  200     0     0    0  0000
39    0    0 J_ST_STOPPED J_RIGHT_KNEE
>
```

Log Module

The log module deals with the logging system in the firmware. This is the system that deals with output messages from the firmware to the monitor. For example, messages from each sub-system.

Certain logging can be enabled, disabled, color coding turned on and off, and the monitor IO redirected to a different port or device.

Almost all log messages output from the firmware are tagged, so that its source and severity can be identified. The logging can also be filtered based on these types – see log enable and log disable.

To see a list of all the types, simply execute the log command with no parameters.

Note: There is also a special 'all' type which can be used to quickly enable or disable all logging.

For example, the module in the firmware that deals with sound playback may print a message with code like the following:

```
Info(MSG_SOUND, "Sound %d started\n", sound_id);
```

This will be sent via the monitor to the current device, and will appear something like:

```
INFO: sound: Sound 4097 started
```

The log enable and log disable commands can filter based on the INFO, the sound or any of the other types.

log color

The log color command enables or disables ANSI color codes in the monitor output.

Format

```
log color <on | off>
```

Arguments

on: turn on ANSI color codes

off: turn off ANSI color codes

Return values

None

Example

```
> log color on
```

LOG DEVICE

The log device command changes the active port used for logging and monitor interaction. Currently, the choices are the UART and the USB connection.

Format

```
log device null | usb | serial
```

Arguments

null: redirect logging to the null device, which will basically drop all output.

usb: redirect logging to the USB port

serial: redirect logging to the serial/UART port

Return values

None

Example

```
> log device serial
```

log disable

The log disable command disables log output based on log types.

Format

```
log disable <log_type>...
```

Arguments

log_type: names of tags to disable

Return values

None

Example

```
> log disable DEBUG motion
> log status
FATAL_ERROR ERROR WARN INFO low high vm monitor mi mu sensor
joint sound timing io anim drive emotion cam attn prop seq nxp
res plog script stats application manager all
>
```

LOG ENABLE

The log enable command enables log output based on log types.

Format

```
log enable <log_type>...
```

Arguments

log_type: names of tags to enable

Return values

None

Example

```
> log enable DEBUG motion
> log status
FATAL_ERROR ERROR WARN INFO DEBUG low high vm monitor mi mu
sensor joint motion sound timing io anim drive emotion cam attn
prop seq nxp res plog script stats application manager all
>
```

LOG EOL

The log eol command sets the line termination character used in monitor output. This allows the output to match the default requirement for the platform being used. For example, if capturing log output to a file, adjusting this setting will help some native applications in reading the file properly.

CR signifies Carriage Return, which is usually put in the output stream with a '\r' character.

LF signifies LineFeed, which is usually put in the output stream with a '\n' character.

See http://en.wikipedia.org/wiki/Newline for more information

FORMAT

```
log eol <dos | unix>
```

ARGUMENTS

dos: use CR-LF pairs (\r\n). default.

unix: use only LF (\n)

RETURN VALUES

None

EXAMPLE

```
> log eol unix
>
```

LOG FILE

The log file command controls redirection of monitor output to a file. The resultant file is a text file containing all the monitor output that is also sent to the serial connection. The same filters that are set for the serial connection – via log enable and log disable – also apply to the information saved in this file.

Format

```
log file <filename> | off | flush
```

Arguments

filename: name of log file to create and use for monitor output

off: stop redirection to the previously given log file and close the file

flush: commit any unwritten data to the current log file, if any

Return values

None

Example

```
> log file test.log
INFO: low: io: ts 887726: eopen(test.log, a)
> log file off
INFO: low: io: ts 920422: eclose()
>
```

log status

The log status command will print all the log types that are currently enabled.

Format

```
log status
```

Arguments

None

Return values

None

Example

```
> log status
FATAL_ERROR ERROR WARN INFO DEBUG low high vm monitor mi mu
sensor joint motion sound timing io anim drive emotion cam attn
prop seq nxp res plog script stats application manager all
```

Monitor Module

The monitor module provides a way to watch all sensor, joint, and motion status at once.

The monitor module provides a group of commands for configuring the Pleo Monitor.

```
help monitor
Syntax: monitor <command>
Monitor help screen:
  disable <monitor target>. . .  Turn off monitor updates
  enable <monitor target>. . .   Turn on monitor updates
  mode [on | off]
```

MONITOR DISABLE

The monitor disable command allows turning off certain logging that was turned on by the monitor enable command.

Format

```
monitor disable all | sensor | joint | motion | sound
```

Arguments

all: disable all monitor watches

sensor: disable the sensor output

joint: disable the joint output

motion: disable the motion output

sound: disable the sound output

Return values

None

Example

```
> monitor disable all
```

MONITOR ENABLE

The monitor enable command allows turning on certain logging of sensor, joint, motion and sound states in a way that makes it easier to parse for a controlling application.

Format

```
monitor enable all | sensor | joint | motion | sound
```

Arguments

all: enable all monitor watches

sensor: enable the sensor output

joint: enable the joint output

motion: enable the motion output. Note: currently not implemented

sound: enable the sound output. Note: currently not implemented

Return values

None

Example

```
> monitor enable sensor
mi: sensor: SENSOR_RESERVED SENSOR_RESERVED SENSOR_BATTERY
SENSOR_IR SENSOR_IR_ACTIVITY SENSOR_RESERVED SENSOR_HEAD
SENSOR_CHIN SENSOR_BACK SENSOR_LEFT_LEG SENSOR_RIGHT_LEG
SENSOR_LEFT_ARM SENSOR_RIGHT_ARM SENSOR_ARSE SENSOR_FRONT_LEFT
SENSOR_FRONT_RIGHT SENSOR_BACK_LEFT SENSOR_BACK_RIGHT
SENSOR_CARD_DETECT SENSOR_WRITE_PROTECT SENSOR_LEFT_LOUD
SENSOR_LIGHT SENSOR_RIGHT_LOUD SENSOR_OBJECT SENSOR_MOUTH
SENSOR_RESERVED SENSOR_SOUND_DIR SENSOR_LIGHT_CHANGE
SENSOR_SOUND_LOUD SENSOR_TILT SENSOR_TERMINAL
SENSOR_POWER_DETECT SENSOR_USB_DETECT SENSOR_WAKEUP
SENSOR_BATTERY_TEMP SENSOR_CHARGER_STATE SENSOR_SHAKE
SENSOR_SOUND_LOUD_CHANGE SENSOR_BEACON SENSOR_BATTERY_CURRENT
SENSOR_PACKET
> mu: sensor: 0 0 58 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 56 31 52
73 0 0 0 1 56 2 21 0 0 0 20 0 0 16 0 307 0
mu: sensor: 0 0 58 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 56 31 52
73 0 0 0 1 56 2 21 0 0 0 20 0 0 16 0 300 0
mu: sensor: 0 0 58 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 56 31 52
73 0 0 0 1 56 2 21 0 0 0 20 0 0 16 0 300 0
mu: sensor: 0 0 58 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 56 31 52
73 0 0 0 1 56 2 21 0 0 0 20 0 0 16 0 300 0
```

MONITOR MODE

The monitor mode command ???

Format

```
monitor mode [on | off]
```

Arguments

off: turn monitor mode off

on: turn monitor mode on. default if no parameter given

Return values

None

Example

```
> monitor mode on
```

Motion Module

The Motion module provides a group of commands for interacting with motion files.

The motion files can be individual Ugobe Motion Files (files with a .UMF extension) on an SD Card. Or, they can be motions that are contained within a running Pleo application, referenced either by Motion ID (number) or name.

MOTION COMMAND

The motion command will load and execute a Ugobe Command File (.UCF) file. Note that there are two types of Command Files: Sound and Motion. You must ensure that the command_id passed to this command represents a Motion Command, not a Sound Command.

Format

```
motion command <command_id|command_name>
```

Arguments

command_id: the ID of the command to load and execute

command_name: the name of the command to load and execute

Return values

None

Example

```
> motion command cmd_sit
```

motion csv

The motion csv command will toggle a flag in the motion system which will print to the monitor the equivalent CSV data of a motion being played. CSV is a comma separated value representation of the current frame.

Format

```
motion csv
```

Arguments

None

Return values

This command will display the new status of the csv flag after completed.

Example

```
> motion csv
Turned CSV logging on.
>
> motion csv
Turned CSV logging off.
>
```

MOTION DUMP

The motion dump command will print information about the motion system and the motions currently being played.

Format

```
motion dump
```

Arguments

None

Return values

Prints number of motion files and statistical information of each joint.

Example

```
> motion dump
*** Motion File Status ***
Num files playing: 0
Joint     Late  OnTime  PercentLate  AveDistShort  Name
 0.      0        0        0%       0    J_RIGHT_SHOULDER
 1.      0        0        0%       0    J_RIGHT_ELBOW
 2.      0        0        0%       0    J_LEFT_SHOULDER
 3.      0        0        0%       0    J_LEFT_ELBOW
 4.      0        0        0%       0    J_LEFT_HIP
 5.      0        0        0%       0    J_LEFT_KNEE
 6.      0        0        0%       0    J_RIGHT_HIP
 7.      0        0        0%       0    J_RIGHT_KNEE
 8.      0        0        0%       0    J_TORSO
 9.      0        0        0%       0    J_TAIL_HORIZONTAL
10.      0        0        0%       0    J_TAIL_VERTICAL
11.      0        0        0%       0    J_NECK_HORIZONTAL
12.      0        0        0%       0    J_NECK_VERTICAL
13.      0        0        0%       0    J_HEAD
14.      0        0        0%       0    J_NONE
15.      0        0        0%       0    J_NONE
*** Motion File Status Done ***
>
```

motion load

The motion load command will load a motion file from an SD Card, preparing it to play with the motion run, pause, step or stop commands. This command only works with stand-alone .umf files, not the motions built into a .urf file. Doing an 'ls' on the SD Card that contains the motions is the easiest way to see the available motions.

You should ensure that the current drive is set to the SD Card (A:) via a chdrv command.

Format

```
motion load <motion_id>
```

Arguments

motion_id: ID of motion file to load.

Return values

Returns a motion handle that can be used with the motion run, pause step or stop commands.

Example

```
> motion load 16323
```

motion pause

The motion pause command will pause a currently playing motion.

Format

```
motion pause <motion_handle>
```

Arguments

motion_handle: handle returned from the motion load command

Return values

None

Example

```
> motion pause 1
```

MOTION PLAY

The motion play command will immediately load and play a motion. The motion can be an individual motion file (.umf) on a SD Card or DataFlash, or a motion in the application that is currently executing.

To get a list of motions available to play within the running application, use the motion show command.

Format

```
motion play <motion_id|motion_name>
```

Arguments

motion_id: ID of motion to play

motion_name: name of motion to play

Return values

Return motion handle that can be used in the other motion commands

Example

```
> motion play mot_sit
```

motion run

The motion run command will start playback of a motion loaded with the motion load command.

Format

```
motion run <motion_handle> [loops] [wait]
```

Arguments

motion_handle: handle returned from the motion load command

loops: number of times to play the motion

wait: whether to wait for each joint to reach its target before moving on

Return values

None

Example

```
motion run 1
```

motion show

The motion show command will list all available motions - their IDs and names – for the currently loaded application.

Format

```
motion show
```

Arguments

None

Return values

Returns list of all available motions

Example

```
> motion show
motion: ID=8192, name=2_step_f_sniff
motion: ID=8193, name=3_step_fullleft_f_left_t2
motion: ID=8194, name=3_step_fullright_f_left_t2
motion: ID=8195, name=angry
motion: ID=8196, name=angry_hiss_sec
motion: ID=8197, name=arse_dip
motion: ID=8198, name=arse_dip_down
motion: ID=8199, name=arse_dip_up
motion: ID=8200, name=arse_wiggle
motion: ID=8201, name=baby_bird_beg
motion: ID=8202, name=baby_bird_feeding
motion: ID=8203, name=baby_bird_nofood
motion: ID=8204, name=back_3step_left
motion: ID=8205, name=back_3step_right
motion: ID=8206, name=back_dip
motion: ID=8207, name=back_up_fast
motion: ID=8208, name=back_up_slow
motion: ID=8209, name=bite
motion: ID=8210, name=bite_s
motion: ID=8211, name=blink_head_move
motion: ID=8212, name=blink_med
motion: ID=8213, name=blink_slow
motion: ID=8214, name=breath_taking_large
motion: ID=8215, name=breath_taking_med
motion: ID=8216, name=burp_large
motion: ID=8217, name=burp_large_1
motion: ID=8218, name=carry_talking
```

MOTION SPEED

The motion speed command will adjust the speed of motion playback globally.

Format

```
motion speed <percentage>
```

Arguments

percentage: the percentage to change speed. Maximum is 100, minimum is 5.

Return values

Status

Example

```
> motion speed 50
INFO: low: motion: Set playback speed to 50% of normal.
Set motion speed to 50% of normal.
>
```

MOTION STEP

The motion step command will move a motion loaded via the motion load command forward by one frame or vector.

Format

```
motion step <motion_handle>
```

Arguments

motion_handle: handle returned from the motion load command

Return values

None

Example

```
motion step 1
```

motion stop

The motion stop command will stop playback of a motion that was loaded via the motion load command.

Format

```
motion pause <motion_handle>
```

Arguments

motion_handle: handle returned from the motion load command

Return values

None

Example

```
motion stop 1
```

Packet Module

The packet module is a monitor interface to the NXP ARM7 processor in Pleo's head. The firmware in the NXP may be updated using the pkt burn command. Features implemented within the NXP – like IR, Camera, Sound - can be enabled or disabled using the pkt commands.

A better name for this group may have been 'nxp', since not all commands are directly related to the packet system itself. For example, pkt burn and pkt reset update the NXP firmware and reset the NXP processor, respectively.

PKT BAUD

The pkt baud command changes the speed at which the main Atmel ARM7 and the NXP ARM7 communicate over the UART connection. The default speed is 115200 baud. Changing this value may cause more packet errors, thus affecting the performance of Pleo.

Format

```
pkt baud <rate>
```

Arguments

rate: new baud rate to use for NXP-Atmel UART connection.

Return values

None

Example

```
> pkt baud 230400
>
```

pkt burn

The pkt burn command is used to upload a new firmware into the NXP processor. This firmware image can either be encrypted (.img extension) or unencrypted (.bin extension).

The system validates that this is a valid Pleo NXP image by looking for a proper signature within the file.

Format

```
pkt burn [file_name]
```

Arguments

file_name: name of file to use for burning. Defaults to 'head.img', assuming it is in the current directory. file_name can also be a full path to the file; for example, a:/head.img (check this!)

Return values

While it is uploading and flashing, a percentage complete will be printed to the monitor.

Example

```
> pkt burn
```

PKT FILTER

The pkt filter command sets what packet types will be redirected to script. On reception of a packet that has been set to be received by script, a SENSOR_PACKET trigger will be sent to the on_sensor function in the current script executing in the Sensor VM – typically sensors.amx.

Allowing scripts to intercept packets allows new features to be added to the NXP firmware. This would be done by creating a new packet type, or overriding an existing one, and then handling it in script. In this way access to the main Atmel ARM7 source code is not necessary.

Format

```
pkt filer a|c|e|i|l|m|p|s|t|v
```

Arguments

a: audio

c: camera

e: echo

i: IR

l: log

m: mouth

p: statistic

s: serial ram

t: touch

v: version

Return values

None

Example

```
> pkt filter a
```

>

pkt reset

The pkt reset command will cycle power to the head NXP processor. This will also cause the camera to be reset.

Format

```
pkt reset
```

Arguments

None

Return values

Status as to reset conditions

Example

```
> pkt reset
INFO: nxp: Resetting head; flash = 0
> INFO: nxp: 8128 - Mar 19 2008 06:25:46
INFO: nxp: Stack end:
INFO: nxp: 40001C1C
INFO: nxp: SRAM end:
INFO: nxp: 40001490
INFO: nxp: SRAM remaining:
INFO: nxp: 0000078C
```

PKT SEND

The pkt send command allows sending packet data directly to the NXP processor. These packets are used to configure all functionality in the NXP processor.

Format

```
pkt send <data>
```

Arguments

data: packet data to send

Return values

Depends on packet sent to NXP

Example

```
> pkt send ?
> INFO: nxp: 8128 - Mar 19 2008 06:25:46

> pkt send ce
> INFO: nxp: YCbCr422 C0 Registers
```

Note: The example above sends two different packets. Example 1 sends a '?' packet, which requests the NXP firmware to send back its version string via a log packet. Example 2 is sending a 'ce' packet, which is the 'c'amera 'e'nable function.

pkt stats

The pkt stats command will print running statistical data about the packet system.

Format

```
pkt stats
```

Arguments

None

Return values

Returns number of packets sent, received, number of packet errors detected and the number of errors detected over the UART.

Example

```
> pkt stats
tx_pkt: 8
rx_pkt: 1217
rx_err_pkt: 0 (0ms)
rx_err_com: 0 (0ms)
> head stats:
t00000482r00000005a00000000o00000001100000000x00000001
```

pkt test

The pkt test command will send an echo packet with all character values to ensure that no data is lost. That is, that all character values properly go to and from the NXP processor.

Format

```
pkt test
```

Arguments

None

Return values

Success string or failure code

Example

```
> pkt test
Echo packet packet received
Echo packet test received and good.
```

Property Module

The Property module provides an interface into the Property system of the firmware. Properties are either system properties, defined within the firmware, or user properties, defined within the application.

PROPERTY SET

The property set command sets the value of a specific property. All available properties can be displayed via the property show command.

Format

```
property set <property_id> <value>
```

Arguments

property_id: ID of property or property name

value: new value of property

Return values

Name and value of the requested property

Example

```
> property set 4 1
DEBUG: prop: property_set_value: property 4 not found, setting
at index 0
```

PROPERTY SHOW

The property show command will show a specific property, or all properties if no property ID is passed.

Format

```
property show [property_id]
```

Arguments

property_id: ID of property or property name

Return values

Name and value of the requested property

Example

```
> property show
property: ID=4, name=age, value=1
>
> property show
property: ID=20, name=platform, value=4
property: ID=32, name=attn_track_weight, value=1
property: ID=35, name=attn_track_mindist, value=3
property: ID=33, name=attn_hold_flags, value=0
property: ID=37, name=attn_p2p_timeout, value=300000
property: ID=34, name=cam_img_progress, value=0
>
```

PROPERTY REPORT

The property report command will enable a property value change to trigger an on_property call in the script that is currently executing in the Sensor VM. If the script in the Sensor VM does not have an op_property function exposed, then this command will have no effect.

Format

```
property report on| off | (<property_id> <min> <max>)
```

Arguments

on | off: whether to enable or disable the property change trigger

property_id: which ID to watch and trigger

min: minimum value that the property has to change by to cause a trigger

max: maximum value that the property can change by to cause a trigger

Return values

Success message whether report was turned on or off.

Example

```
> property report 4 1 1
INFO: prop: Added reporter for prop 4: trig 1 change 1
INFO: monitor: property reporter added
>
> property report off
INFO: monitor: property reporting OFF
>
```

Resource Module

The Resource module provides an interface into the Resource System of the firmware. Resources are bundled as URF files, and can be executed either from an SD Card or be installed and executed from internal DataFlash.

Resource types include sounds, motions, commands and scripts. Application-defined properties are also considered resources since they are included in the URF file.

Resources have both a numeric ID, generated when the URF is built, and a name, which is the original resource file name minus any suffix.

RESOURCE BACKUP

The resource backup command will copy the URF data from DataFlash to an SD Card.

This is a ‘protected’ command. The access command is required to use this command.

Format

```
resource backup <filename>
```

Arguments

filename: file name to store the DataFlash URF image to.

Return values

None

Example

```
> resource backup one.urf
INFO: low: io: ts 180906: eopen(one.urf, w)
INFO: low: io: ts 181162: eclose()
INFO: res: stats: Resource file backup successfull.
>
```

RESOURCE CLEAR

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

The resource clear command will initialize the DataFlash in such a way as to remove the URF from the reserved DataFlash memory.

This is a 'private' command. The access command is required to use this command.

FORMAT

```
resource clear
```

ARGUMENTS

None

RETURN VALUES

None

EXAMPLE

```
> resource clear
INFO: res: stats: Clearing DF resource area...
INFO: res: stats: DF Resource Clear succeeded.
>
```

RESOURCE INFO

The resource info command displays information about the currently executing URF. This information currently includes the revision number and the date and time the URF was built. More information may be added in future versions.

Format

```
resource info
```

Arguments

None

Return values

Version and build date of URF

Example

```
> resource info
Resource file (current) information:
    revision: 0
    built: May 09 2008 04:18:12
>
```

resource show

The resource show command will enumerate all resources within the currently executing URF and display the type, ID and name of each.

Format

```
resource show
```

Arguments

None

Return values

Type, ID and Name of all resources

Example

```
> reso sh
sound: ID=4096, name=do_re_mi
script: ID=16384, name=main
script: ID=16385, name=neckTail
script: ID=16386, name=jointN
property: ID=20, name=platform, value=4
property: ID=32, name=attn_track_weight, value=1
property: ID=35, name=attn_track_mindist, value=3
property: ID=33, name=attn_hold_flags, value=0
property: ID=37, name=attn_p2p_timeout, value=300000
property: ID=34, name=cam_img_progress, value=0
>
```

RESOURCE UPGRADE

The resource upgrade command will install a URF file from an SD Card into the internal DataFlash. If the URF installed in DataFlash matches that being requested, the resource upgrade command will not perform any work and will return immediately.

While the URF is being installed, it will maintain a checksum value of the written data. When the URF has been completely transferred, that checksum will be compared to the one written in the source URF. If they do not match, the DataFlash will be cleared so that we will not attempt to execute a bad URF from DataFlash.

Format

```
resource upgrade [filename] [T]
```

Arguments

filename: name of URF to install into DataFlash. If not given, will default to look for a "pleo.urf".

T: if this parameter is passed, then the command will skip the normal version checks and install the resource file anyway. Available only in Pleo firmware version 1.1 and above.

Return values

None

Example

```
> resource upgrade testDF.urf
INFO: low: io: ts 3619820: eopen(testDF.urf, r)
INFO: low: io: ts 3619833: eopen(pm_ilock, w)
INFO: low: io: ts 3620032: eclose()
Adler32 result: 0xF2156900
INFO: res: stats: Resource file install passed
>
```

RESOURCE VERIFY

The resource verify command executes a checksum calculation on the currently loaded Pleo application, and compares it to the checksum that is stored inside the resource file (.urf). This allows us to confirm whether the data of the application may have been damaged in some way.

Format

```
resource verify
```

Arguments

None

Return values

None

Example

```
> resource verify
Calculating Adler32...
...Adler32 from file: 0xF2156900, calculated:0xF2156900 in 26ms
INFO: res: stats: Resource file passed validation.
>

> resource verify
Calculating Adler32...
...Adler32 from file: 0xF2156900, calculated:0x5F1ED9F4 in 11ms
INFO: res: stats: Resource file failed validation.
>
```

Sensor Module

The Sensor Module allows the viewing of the current configuration and real-time values of all or one of Pleo's sensors.

```
Sensors:
   2 SENSOR_BATTERY
   3 SENSOR_IR
   6 SENSOR_HEAD
   7 SENSOR_CHIN
   8 SENSOR_BACK
   9 SENSOR_LEFT_LEG
  10 SENSOR_RIGHT_LEG
  11 SENSOR_LEFT_ARM
  12 SENSOR_RIGHT_ARM
  13 SENSOR_ARSE
  14 SENSOR_FRONT_LEFT
  15 SENSOR_FRONT_RIGHT
  16 SENSOR_BACK_LEFT
  17 SENSOR_BACK_RIGHT
  18 SENSOR_CARD_DETECT
  19 SENSOR_WRITE_PROTECT
  21 SENSOR_LIGHT
  23 SENSOR_OBJECT
  24 SENSOR_MOUTH
  26 SENSOR_SOUND_DIR
  27 SENSOR_LIGHT_CHANGE
  28 SENSOR_SOUND_LOUD
  29 SENSOR_TILT
  30 SENSOR_TERMINAL
  31 SENSOR_POWER_DETECT
  32 SENSOR_USB_DETECT
  33 SENSOR_WAKEUP
  34 SENSOR_BATTERY_TEMP
  35 SENSOR_CHARGER_STATE
  36 SENSOR_SHAKE
  37 SENSOR_LOUD_CHANGE
  38 SENSOR_BEACON
  39 SENSOR_BATTERY_CURRENT
  40 SENSOR_PACKET
```

SENSOR DISABLE

The sensor disable command disables a sensor, preventing it from causing any triggers to the script level.

Format

```
sensor disable <sensor_id>
```

Arguments

sensor_id: ID of sensor to disable

Return values

None

Example

```
> sensor disable 13
>
> sensor show 13
                 Aux   Debounce
         Trig   Trig     Time                    Trig
 Sensor  Level  Level   in ms.  Enabled  Trig  Time   TmSince
Count       Value  RawValue  Name
-----------------------------------------------------------------
-----------------
     13      0      0         0        0     0       0       0
0 (  0%)      0          0 SENSOR_ARSE
>
```

SENSOR ENABLE

The sensor enable command enables a specific sensor. When a sensor is enabled, its driver will be active and passing that information up the software stack all the way to any script that is executing in the Sensor VM.

Format

```
sensor enable <sensor_id>
```

Arguments

sensor_id: ID of sensor to enable

Return values

None

Example

```
> sensor enable 13
> sensor show 13
                 Aux   Debounce
         Trig   Trig     Time                    Trig
 Sensor  Level  Level   in ms.  Enabled  Trig  Time   TmSince
Count       Value  RawValue  Name
-----------------------------------------------------------------
-----------------
     13      0      0         0        1     0       0       0
0 (  0%)      0          0 SENSOR_ARSE
>
```

sensor set

The sensor set command sets the working parameters for a given sensor, such as trigger level, and de-bounce values.

Format

```
sensor set <sensor_id> <trigger_level> <aux_trigger_level>
<debounce> [enabled] [triggered]
```

Arguments

sensor_id: ID of sensor to change

trigger_level: value of trigger

aux_trigger_level: value of secondary trigger

debounce: currently ignored for Pleo firmware 1.0x and 1.1x

enabled: whether to enable this sensor. 0 means disable and 1 means enable

triggered: whether to set the trigger flag for this sensor. 0 will turn off the sensor trigger (false) and 1 will turn the trigger on (true).

Return values

None

Example

```
> sensor set 21 35
```

sensor setlevel

The sensor setlevel command allows setting the current value of a sensor. Setting a new value may cause a trigger to occur, which may result in the sensor script being called.

Format

```
sensor setlevel <sensor_id> <level>
```

Arguments

sensor_id: ID of sensor to change

level: new value for this sensor

Return values

None

Example

```
> sensor setlevel 29 4
INFO: monitor: sensor 29 set to 4 at t=5716889
```

SENSOR SHOW

The sensor show command prints the current state for a particular sensor, or all sensors.

Format

```
sensor show [sensor_id]
```

Arguments

sensor_id: ID of sensor to display. If omitted, all sensors will be shown

Return values

Displays all data related to a sensor.

Example

```
> sensor show 29
                 Aux   Debounce
         Trig   Trig     Time                    Trig
 Sensor  Level  Level   in ms.  Enabled  Trig  Time   TmSince
Count       Value  RawValue  Name
-----------------------------------------------------------------
-----------------
     29      0      0         0        1     0    2569     785
2 (  0%)      2          2 SENSOR_TILT
```

SENSOR WATCH

The sensor watch command continuously prints the current state of a sensor, allowing an easy way to view sensor changes as they happen.

Format

```
sensor watch <sensor_id>
```

Arguments

sensor_id: ID of sensor to watch

Return values

Current sensor status

Example

```
> sensor watch 29
                 Aux   Debounce
         Trig   Trig     Time                    Trig
 Sensor  Level  Level   in ms.  Enabled  Trig  Time   TmSince
Count       Value  RawValue  Name
-----------------------------------------------------------------
-----------------
>    29      0      0         0        1     0 5716890 5714321
3 (  0%)      2          2 SENSOR_TILT
```

Sound Module

The Sound module is an interface into the sound playback system of Pleo. It allows showing all available sounds, playing sounds and setting the global volume and pitch settings for the Sound System.

SOUND COMMAND

The sound command allows the loading and playback of a Sound Command, or .UCF file. A Sound Command is a list of possible sounds with properties that allow the firmware to select one of them based on the current state of the system.

Only Sound Commands from the currently loaded application can be executed. The current list of available Commands can be seen via a resource show command.

Format

```
sound command <command_id|command_name>
```

Arguments

command_id: ID of Command to play

command_name: name of Command to play

Return values

None

Example

```
> sound command sound_cmd_test

INFO: sound: find_entry: returns entry 1 with weight 267
INFO: sound: Playing sound file sigh
```

sound play

The sound play command allows the playback of an individual sound file – either .USF or .WAV – or the playing of a sound that is contained within an application (URF on SD Card or URF in DataFlash).

Note this command can NOT play Sound Commands. See sound command for that functionality.

Note that Sound ID's between different applications may be the same, but the actual sound name may be different. That is, the range of sound ID's is the same for all Pleo applications.

Format

```
sound play <id> | <name>
```

Arguments

id: ID of sound to play

name: name of sound to play

Return values

None

Example

```
> sound play 4096
INFO: low: sound: ts 6492127: load sound file do_re_mi
INFO: sound: Playing sound file do_re_mi
Playing sound file 4096
INFO: low: sound: ts 6492888: sound 4096 stopped.
INFO: sound: sound 2124448 ended.

> sound play do_re_mi
INFO: sound: Playing sound file do_re_mi
INFO: low: sound: ts 6552004: sound 0 stopped.
INFO: sound: sound 2124448 ended.
```

SOUND SHOW

The sound show command will list all available sounds in the currently loaded application, including their ID's and names.

Format

```
sound show
```

Arguments

None

Return values

Lists all sounds in the resource file

Example

```
> sound show
sound: ID=4096, name=do_re_mi
>
```

SOUND SPEED

The sound speed command will set the sound playback speed globally. That is, all future sound playback will occur at the speed if successfully set.

Format

```
sound speed <percent>
```

Arguments

percent: a percentage of normal speed (100). Minimum is 25% and maximum is 200%

Return values

Status

Example

```
> sound speed 200

Set playback speed to 200% of normal.
```

sound volume

The sound volume command will set the sound volume adjustment globally. That is, all future sounds will play at the new volume.

Format

```
sound volume <percent>
```

Arguments

percent: percentage of normal volume (100%). Minimum is 0% and maximum is 200%

Return values

Status

Example

```
> sound volume 150

Set playback volume to 150% of normal.
>
```

Stats Module

The Stats module is used to view a number of statistics in different modules.

STATS ALL

The stats all command will show all results for all stats modules in one shot.

Format

```
stats all
```

Arguments

None

Return values

All stats information

Example

```
> stats all
```

stats cpu

The stats cpu command will show a breakdown of CPU usage per module in the firmware.

Format

```
stats cpu
```

Arguments

None

Return values

Statistics

Example

```
> stats cpu
CPU usage in milliseconds:
  Description            Count       Time    Percent
  ------------------------------------------------
  Initialization             1        853    0.0120%
  Monitor             11529103    2440026   34.4601%
  Low Level           11529103    4093230   57.8081%
  High Level          23058205     228079    3.2211%
    Script               32384        119    0.0016%
    Motion               32386        207    0.0029%
    Drive                16192      80963    1.1434%
    Sensor                   0          0    0.0000%
    Property             16192       1443    0.0203%
    Animation            16192         18    0.0002%
    Behavior                 0          0    0.0000%
    Sound                16193        161    0.0022%
    Camera               16192        618    0.0087%
    Resource                 0          0    0.0000%
    Application         113348       6259    0.0883%
  Virtual Machine       113348       3454    0.0487%
    Initialization           1        102    0.0014%
    Execution           113352       2697    0.0380%
    Native Calls             5         47    0.0006%
    IO                       0          0    0.0000%
  Filesystem               270       3003    0.0424%
  SPI Waiting                0          0    0.0000%
  ------------------------------------------------
  Total                        7080720       100%
>
```

STATS FS

The stats fs command displays information about files being opened and closed.

Format

```
stats fs
```

Arguments

None

Return values

See below

Example

```
> stats fs
Number of open files = 1
Files opened = 24
Files closed = 23
Files failed = 6
>
```

stats hist

The stats hist command displays a breakdown of the duration in time for each iteration through the main loop. The main loop is the central function that calls out to all subsystems with the firmware.

This command gives you a good idea of whether there may be some issue with the system. That is, a large number of samples at a high time period indicates the system is unbalanced.

Format

```
stats hist
```

Arguments

None

Return values

See below

Example

```
> stats hist
Main loop execution time histogram:
Bin    Time Range (ms)    Count  Percent
 0.   1000... 5000       84     0%
 1.    750... 999        2     0%
 2.    500... 749        3     0%
 3.    250... 499        2     0%
 4.    100... 249       23     0%
 5.     75...  99        0     0%
 6.     50...  74        0     0%
 7.     25...  49       23     0%
 8.     10...  24     6705     0%
 9.      5...   9     2284     0%
10.      0...   4   4658236    40%
>
```

stats mem

The stats mem command displays stack and heap information for the main Atmel firmware. This allows the user to see whether the firmware is running out of space, or that stack depth has become too deep.

Format

```
stats mem
```

Arguments

None

Return values

See below

Example

```
> stats mem
Stack and heap separation = 3020 bytes; min so far 3020 bytes
Heap address 0x0020F198 stack address 0x0020FD64!
>
```

STATS POWER

The stats power command displays details about the power system.

Format

```
stats power
```

Arguments

None

Return values

See below

Example

```
> stats power
                  Battery Voltage = 74, Min 74, Max 75 (tenths
of a volt)
              Battery Temperature = 28, Min 24, Max 28 (degrees
C)
                  Battery Current = 285, Min 257, Max 364
(milliamperes)
             Battery sensor value = 66 (% capacity remaining)
      Battery sensor direct value = 66, Min 0, Max 66
              Battery Voltage A/D = 658, Min 652, Max 667
          Battery Temperature A/D = 286, Min 282, Max 329
              Battery Current A/D = 40, Min 36, Max 51
            Thermistor Resistance = 8774, Min 8748, Max 10596
(ohms)
      Battery Internal Resistance = 0, Min 100000, Max 0
(milliohms)
        Battery Capacity Consumed = 588 (mAh)
                       Powered up = true
                  RTK Adjustments = 0
                            RTK IK = 0
                            RTK VB = 0
Power monitor reset early warning = 0 (/LLO)
>
```

stats sd

The stats sd command will display low-level register and speed information for any SD Card that is currently inserted. This is a debugging feature to test the SD Card portion of the firmware.

The output includes CID, MID and CSD registers of the currently inserted SD Card. Definitions of these registers can be found in the SD Card specifications.

If no SD Card is present, then the output will so indicate.

Format

```
stats sd
```

Arguments

None

Return values

See below

Example

```
> stats sd
CID: 03 53 44 53 44 31 32 38 80 60 37 19 C3 00 74 BD
 MID:  3 OID: SD PNM: SDSD MDT: 2007/4
CSD: 00 26 00 32 5F 59 83 C0 FE FA 4F FF 92 40 40 B5
SDHC speed: 25000000Hz
>
```

stats sys

The stats sys command displays detailed data that is persisted to DataFlash about the life form.

Format

```
stats sys
```

Arguments

None

Return values

See below

Example

```
> stats sys
System Statistics:
Sensor Triggers Disabled/Reset Name
     0        0          0 SENSOR_RESERVED
     1        0          0 SENSOR_RESERVED
     2       32          0 SENSOR_BATTERY
     3        0          0 SENSOR_IR
     4        0          0 SENSOR_IR_ACTIVITY
     5        0          0 SENSOR_RESERVED
     6        2          0 SENSOR_HEAD
     7        2          0 SENSOR_CHIN
     8        0          0 SENSOR_BACK
     9        0          0 SENSOR_LEFT_LEG
    10       38          0 SENSOR_RIGHT_LEG
    11        0          0 SENSOR_LEFT_ARM
    12        0          0 SENSOR_RIGHT_ARM
    13        0          1 SENSOR_ARSE
    14        0          0 SENSOR_FRONT_LEFT
    15        0          0 SENSOR_FRONT_RIGHT
    16        0          0 SENSOR_BACK_LEFT
    17        2          0 SENSOR_BACK_RIGHT
    18       31          0 SENSOR_CARD_DETECT
    19        6          0 SENSOR_WRITE_PROTECT
    20     1231          0 SENSOR_LEFT_LOUD
    21       25          0 SENSOR_LIGHT
    22     1317          0 SENSOR_RIGHT_LOUD
    23       71          0 SENSOR_OBJECT
    24        0          0 SENSOR_MOUTH
    25        0          0 SENSOR_RESERVED
    26       86          0 SENSOR_SOUND_DIR
    27        0          0 SENSOR_LIGHT_CHANGE
    28     1285          0 SENSOR_SOUND_LOUD
    29       39          4 SENSOR_TILT
    30      513          0 SENSOR_TERMINAL
    31        0          0 SENSOR_POWER_DETECT
    32        4          0 SENSOR_USB_DETECT
```

```
    33        0           0 SENSOR_WAKEUP
    34        0           0 SENSOR_BATTERY_TEMP
    35        0           0 SENSOR_CHARGER_STATE
    36        2           0 SENSOR_SHAKE
    37      206           0 SENSOR_SOUND_LOUD_CHANGE
    38        0           0 SENSOR_BEACON
    39        0           0 SENSOR_BATTERY_CURRENT
    40    28316           0 SENSOR_PACKET
Joint Movements DistanceMoved Stalls Displacements Name
    0         1            25      0        0 J_RIGHT_SHOULDER
    1         1           255      0        0 J_RIGHT_ELBOW
    2         2            33      0        0 J_LEFT_SHOULDER
    3         1           101      1        0 J_LEFT_ELBOW
    4         2            42      1        0 J_LEFT_HIP
    5         2            61      0        0 J_LEFT_KNEE
    6         1            11      1        0 J_RIGHT_HIP
    7        12          1966      0        0 J_RIGHT_KNEE
    8         2            34      1        0 J_TORSO
    9         0             0      0        0 J_TAIL_HORIZONTAL
   10         1           129      0        0 J_TAIL_VERTICAL
   11         2            76      0        0 J_NECK_HORIZONTAL
   12         4           226      0        0 J_NECK_VERTICAL
ID: USSF Revision: 4
Pleo S/N: 110400168080; DataFlash Security ID: 0B373317
Total run time (seconds): 25664
Stats write count: 85, last written: 25589
Atmel: resets:19, upgrade count: 0
NXP:   resets:20, upgrade count: 0, upgrade failures: 0,
crashes: 0
MotorController resets: 0; Comm Errors: TSB0=161421 TSB1=0
TSB2=0 TSB3=0
SD card insertions: 20, removals: 11
HL loads: 34, unloads: 15
Motion files played: 0
Amp enabled count: 2
Sound files played: 2
Sound output seconds: 0
Times powered down: 4, powered up: 0
Sleep count: controlled: 0, forced: 4
Charge cycle count: 0, aborted charges: 0, overtemp charges: 0
Cycle OTemp UpRun DnRun StVolt EndVolt StChg EndChg StIntRes
EndIntRes
   -7     0   300     0     60      77  1718   1695        0
0
   -6     0  1200     0     77      76  1695   1672        0
0
   -5     0  3907     0     76      76  1672   1387        0
0
   -4     0  2402     0     76      76  1387   1340        0
0
   -3     0   448     0     76      77  1340   1317        0
0
   -2     0  1209     0     77      75  1317   1222        0
0
   -1     0  1799     0     75      76  1222   1104        0
0
    0     0  2402     0     76      76  1104   1081        0
0
```

```
    1      0  4071      0      76       74   1081     764       682
686
    2      0  7857      0      74       74    764     167       686
0
Overall Average Charge Statistics:
    0      0 25595      0       0        0      0       0         0
0

>
```

stats toshiba

The stats toshiba command displays detailed usage data for the four Toshiba MCUs that are used for motor control.

Format

```
stats toshiba
```

Arguments

None

Return values

See below

Example

```
> stats toshiba
MotorController Stats
Enable = 1, Timer Enable 1, Stall Detect Enable 1
*** MotorController0: Good 1, Configured 1, Comm Failure 0,
Output Bits 01
Xfers: good 899892, bad 0, faults 0
  CH  Vectors RegPolls LatePolls LateDuration Moving Enabled
DirectPWM Joint Name
   0       0   224907         0            0      0       0
0     1 J_RIGHT_ELBOW
   1       0   224906         0            0      0       0
0     0 J_RIGHT_SHOULDER
   2       0   224904         0            0      0       0
0    12 J_NECK_VERTICAL
   3       0   224907         0            0      0       0
0    11 J_NECK_HORIZONTAL
*** MotorController1: Good 1, Configured 1, Comm Failure 0,
Output Bits 01
Xfers: good 899897, bad 0, faults 0
  CH  Vectors RegPolls LatePolls LateDuration Moving Enabled
DirectPWM Joint Name
   0       0   224909         0            0      0       0
0     3 J_LEFT_ELBOW
   1       0   224907         0            0      0       0
0     2 J_LEFT_SHOULDER
   2       0   224905         0            0      0       0
0    13 J_HEAD
   3       0   224908         0            0      0       0
0     8 J_TORSO
*** MotorController2: Good 1, Configured 1, Comm Failure 0,
Output Bits 01
Xfers: good 899900, bad 0, faults 0
  CH  Vectors RegPolls LatePolls LateDuration Moving Enabled
DirectPWM Joint Name
   0       0   224910         0            0      0       0
0     5 J_LEFT_KNEE
```

```
   1       0   224907         0            0      0       0
0     4 J_LEFT_HIP
   2       0   224906         0            0      0       0
0     7 J_RIGHT_KNEE
   3       0   224909         0            0      0       0
0     6 J_RIGHT_HIP
*** MotorController3: Good 1, Configured 1, Comm Failure 0,
Output Bits 01
Xfers: good 450087, bad 0, faults 0
  CH  Vectors RegPolls LatePolls LateDuration Moving Enabled
DirectPWM Joint Name
   0       0   224911         0            0      0       0
0    10 J_TAIL_VERTICAL
   1       0   224909         0            0      0       0
0     9 J_TAIL_HORIZONTAL
   2       0        0         0            0      0       0
0    -1 J_NONE
   3       0        0         0            0      0       0
0    -1 J_NONE
>
```

stats vm

The stats vm displays detailed information about the four Pawn VMs in the firmware.

Note: This command is only useful in Pleo 1.0.x firmware. For Pleo 1.1, see the vm info command.

Format

```
stats vm
```

Arguments

None

Return values

See below

Example

```
> stats vm
VM statistics not enabled
>
```

Toshiba Module

The Toshiba module allows interacting with the four Toshiba motor controllers.

> **Warning!** These commands are dangerous and may render your Pleo inoperable. They are only documented here for completeness and should not be used by the general public.

TOSHIBA AD

The toshiba ad command sets the motor controller K and N for A/D filters.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

FORMAT

```
toshiba ad <ctlr_num> <K> <N>
```

ARGUMENTS

ctlr_num: motor controller number from 0 to 3

K: numeric value from 1 to N; default is 1

N: numeric value from 1 to 255; default is 8

RETURN VALUES

The current A/D filter values for the specified controller number, or all if omitted.

EXAMPLE

```
> toshiba ad
A/D Filter Coefficients:
MotorController K N
0               1 8
1               1 8
2               1 8
3               1 8
>
```

TOSHIBA IN

The toshiba in command queries input bits from the given motor controller.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

FORMAT

```
toshiba in <ctlr_num>
```

ARGUMENTS

ctlr_num: numeric value from 0 to 3

RETURN VALUES

Returns the current digital inputs from the specified motor controller. These digital inputs are used for signals such as the touch sensors and the foot switches.

EXAMPLE

```
> toshiba in 0
MotorController 0: Input Bits = 0x01
>
```

TOSHIBA OUT

The toshiba out command sets the specified output pins in the specified motor controller. They are generally used to control and reset the touch sensor ICs.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

FORMAT

```
toshiba out <ctlr_num> <P0.0> <P0.1>
```

ARGUMENTS

ctlr_num: a numeric value from 0 to 3

P0.0: a value of 0 or 1; sets the P0.0 pin on the specified controller; default is 1

P0.1: a value of 0 or 1; sets the P0.1 pin on the specified controller; default is 0

RETURN VALUES

Returns a confirmation string.

EXAMPLE

```
> toshiba out 0 1 0
MotorController 0: Set Outputs p0_0 = 1, p0_1 = 0
>
```

TOSHIBA RAM

The toshiba ram command dumps command RAM from the specified motor controller.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

Format

```
toshiba ram <ctlr_num> <address> <length>
```

Arguments

ctlr_num: a numeric value from 0 to 3

address: a numeric value from 0 to 255

length: a numeric value from 0 to 31

Return values

The data stored in the command RAM.

Example

```
> toshiba ram 0 0 1

00: 1C
>
```

TOSHIBA RESET

The toshiba reset command resets all motor controllers.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

FORMAT

```
toshiba reset [h|f]
```

ARGUMENTS

h: leave high

f: float

RETURN VALUES

None.

EXAMPLE

```
> toshiba reset 1 1
Resetting MotorControllers... leaving MC_RESET asserted high
Done.
>
```

Utility Commands

The Utility commands are a collection of other commands that do not fit cleanly in any other module.

access

The access command sets the current protection level for the commands in the monitor. Each command has an associated access level, and if the currently selected access level is less than the commands access level, it will not be shown or executed.

Format

```
access <pass_phrase>
```

Arguments

pass_phrase: the code necessary to enter a particular access level. These are not documented, and may change between versions of the firmware.

Return values

None

Example

```
> access
```

clear

The clear command clears the terminal screen. This simply sends a Ctrl-L (the ASCI clear screen command code) back to the terminal, so it needs to be set up to handle this character sequence. If not, this command has no effect.

Format

```
clear
```

Arguments

None

Return values

None

Example

```
> clear
```

config

The config command displays the current configuration information.

Format

```
config
```

Arguments

None

Return values

See below

Example

```
> config
Pleo Hardware Configuration:
  Hardware ID: 0704120B-37331F27-01001A07-FFFF17FF (0B373317)
  Serial Number: 110400168080
  Board Revision: 17-8
  Board Revision ADC: 520
Build Configuration:
  Firmware built for: unknown
  Build type: DEBUG
  Pleo main ARM version: 00000 - May 19 2008 19:46:33
  Pleo head ARM version: 8128 - Mar 19 2008 06:25:46
  Pleo PM (DF) version: 0 - Jan 01 1970 00:00:00
  Bootloader version: 8210:8219M - May 12 2008 10:55:19
  Compiled in:   Thumb mode
  Resource folders:  on
Acroname Toshiba Configuration:
  TSB0: Version 28
  TSB1: Version 28
  TSB2: Version 28
  TSB3: Version 28
VM Configuration:
  Incremental mode:   off
  Code swapping:      false
  malloc mode:        false
  main vm image:      4096
  sensors vm image:   3072
  auxiliary vm image: 2560
  user vm image:      2048
>
```

exec

The exec command executes the specified binary code file, passing with optional parameters. The specific parameters depend on what binary code is being executed. For example, if it is a firmware upgrade component, it will likely take a file name as a parameter.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

Format

```
exec <file>
```

Arguments

file: .bin file to load into memory and execute

Return values

None

Example

Intentionally left blank

help

The `help` command displays help for a command. Typing help without any commands will display a list of available commands.

Format

```
help [command]
```

Arguments

command: command or command group to display help on

Return values

Help text

Example

```
> help motion
```

passthru

The `passthru` command puts the Atmel firmware into a mode where the external UART monitor connection is redirected directly to the NXP UART connection, allowing direct interaction with the NXP firmware.

Format

```
passthru [port0_baud] [port1_baud]
```

Arguments

port0_baud: baud rate to set the Atmel UART port. Defaults to 19200

port1_baud: baud rate to set the Atmel to NXP UART. Defaults to 19200

Return values

None

Example

```
> passthru 9600 9600
```

power

The `power` command forces the power up or power down sequences.

Format

```
power [up | down]
```

Arguments

up: perform power up sequence

down: perform power down sequence

Return values

None

Example

```
> access 3
> power down
```

RECEIVE

The `receive` command will capture the next series of monitor input to a file. This is used to take receive data over the monitor and write it to a (binary) file.

> Note This command was designed for internal use only and is deprecated. It will be removed in a future firmware version

FORMAT

```
receive <filename> <total_byte_count> [chunk_size]
```

ARGUMENTS

filename: name of file to save to

total_byte_count: number of bytes to capture to file

chunk_size: ?

RETURN VALUES

Returns success or failure conditions

EXAMPLE

```
> receive b:test.txt 234
```

RESET

The reset command resets the main Atmel processor.

Warning: This will not do any of the proper shutdown procedures, so data may be lost!

Format

```
reset
```

Arguments

None

Return values

None

Example

```
> reset
```

sdcard

The `sdcard` command simulates the removal and insertion of an SD card.

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

Format

```
sdcard [on | off]
```

Arguments

on: will simulate an SD Card insertion event

off: will simulate an SD Card removal event

Return values

None

Example

```
> sdcard on
```

SPI

The `spi` command is used to set the SPI bus frequency for various devices on the SPI bus. In Pleo this includes the Toshiba motor controllers, the Atmel DataFlash and the SD Card.

See this page for an overview of what SPI is:
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Warning! This command is dangerous and may render your Pleo inoperable. It is only documented here for completeness and should not be used by the general public.

Format

```
spi <dev_num> <frequency>
```

Arguments

dev_num: ID of device. 0 for Toshiba, 1 for DataFlash and 2 for SD Card

frequency: clock frequency to use for this device, in cycles

Return values

None

Example

```
> spi 2 12000000
```

time

The time command is used to display the time since Pleo was powered up.

Format

```
time
```

Arguments

None

Return values

Current time

Example

```
> time
h:m:s:ms 0:05:20.320, tms 320320
us: 318991930
```

upgrade

The `upgrade` command loads a new firmware image from the SD card. This is done by jumping directly to the built in Pleo bootloader. The bootloader is designed to look for an Atmel ARM7 image named ‘pleo.img’ at the root of the SD Card. If this image is found, it will be written to the Atmel flash. If not found, it will wait until an SD Card with this image is inserted, or a USB cable is connected – at which point it switches to a DFU (device firmware upgrade) mode.

Format

```
upgrade
```

Arguments

None

Return values

None

Example

```
> upgrade
Image upgrade: pleo.img
Looking for image in SD card!
```

xmodem

The `xmodem` command is used to upload a file using xmodem protocol.

Note: This command is unsupported and deprecated. It will be removed in a future firmware.

This command was initially put in for internal development. Its functionality has been replaced with the OBEX support. Use of this command is highly discouraged.

Format

```
Xmodem filename length
```

Arguments

filename: name of file to save the data to

length: the number of bytes that will be included in this transfer

Return values

None

Example

```
> xmodem test.wav 1234
```

VM Module

The VM module is used to load, unload and call functions in the fourth Pawn VM, named the User VM. This is convenient to test whether individual scripts are compiled properly and will execute well within the Pleo firmware.

VM CALL

The vm call command will call a named function within a script that was loaded using vm load.

Format

```
vm call <function> [param]
```

Arguments

function: name of function to call

param: optional parameter to pass to the function. these vary depending on the specific function being called. currently limited to one parameter.

Return values

Displays any return value from the called function

Example

```
> vm call init
```

VM INFO

The vm info will display information about all the running VMs, the code pool, etc.

Format

```
vm info
```

Arguments

None

Return values

See below

Example

```
> vm info
block 0x200 at 0x203C80 is    4 bytes
block 0x201 at 0x203C8C is  280 bytes
block 0x202 at 0x203DAC is  164 bytes
total used is 448 bytes out of 8192 bytes
Sensor VM:
  Allocated:  3072
Main VM:
  Allocated:  4096
     Header:   224
       Data:   244
      Stack:   512
      Total:   980
       Free:  3116
Behavior VM:
  Allocated:  2560
User/Init VM:
  Allocated:  2048
>
```

VM LOAD

The vm load command will load the given script into the fourth Pawn VM, called the User VM. The script will be processed according to the standard rules. That is, if there is an init, it will be called right after loading. After the init function completes, and there is a main, it will be called.

Format

```
vm load <script_id>
```

Arguments

script_id: ID or name of script to load. Currently has to be part of an application.

Return values

Any output from loaded script

Example

```
> resource show
sound: ID=4096, name=do_re_mi
script: ID=16384, name=main
script: ID=16385, name=neckTail
script: ID=16386, name=jointN
property: ID=20, name=platform, value=4
property: ID=32, name=attn_track_weight, value=1
property: ID=35, name=attn_track_mindist, value=3
property: ID=33, name=attn_hold_flags, value=0
property: ID=37, name=attn_p2p_timeout, value=300000
property: ID=34, name=cam_img_progress, value=0
> vm load jointN
Start joint test!
```

VM UNLOAD

The vm unload command will unload any script that is in the fourth User VM. If there is a close function in the script, it will be called.

Format

```
vm unload
```

Arguments

None

Return values

Any messages from scripts close function.

Example

```
> vm unload
```

Appendix A Windows Pleo USB Driver

Please refer to the PDK for the latest version of this INF and the associated drivers. It is located in {pdkroot}/tools/drivers.