Compoundpi Documentation

Release 0.3

Dave Hughes

May 23, 2014

Contents

This project provides a means for controlling multiple cameras attached to Raspberry Pi's all of which are attached to the same subnet. Broadcast UDP packets are utilized to permit near-simultaneous triggering of all attach cameras.

The code is licensed under the GPL v3 or above. Packages can be downloaded from PyPI. The source code can be obtained frmo GitHub, which also hosts the bug tracker. The documentation (which includes installation and quick start examples) can be read on ReadTheDocs.

CHAPTER 1

Table of Contents

1.1 Server Installation

The server component of Compound Pi can only be installed on the Raspberry Pi architecture. On Raspbian, the following command can be used to install the server package:

```
$ sudo apt-get install compoundpi-server
```

Warning: The Raspbian package will automatically install the *cpid* daemon in the boot sequence. This will make the camera inaccessible to other processes unless the daemon is manually stopped or prevented from starting.

On other platforms, the package can be installed from PyPI. Specify the `server` option to pull in all dependencies required by the server component:

```
$ sudo pip install "compoundpi[server]"
```

The PyPI package does not include init-scripts (because it can't). You will need to write these for your platform manually if you wish the daemon to start automatically on boot-up.

1.2 Client Installation

The client component of Compound Pi can be installed on any machine with Python available. On Ubuntu, the Waveform PPA can be used for simple installation:

```
$ sudo add-apt-repository ppa:waveform/ppa
$ sudo apt-get update
$ sudo apt-get install compoundpi-client
```

On other platforms, the package can be installed from PyPI. Specify the `client` option to pull in all dependencies required by the client component:

```
$ sudo pip install "compoundpi[client]"
```

Warning: Currently, the version of client and server must match exactly. The client will not work with a different version server (either older or newer).

1.3 Quick Start

By far the easiest method of configuring a fleet of Compound Pi servers is to get a single Pi running the *Compound Pi daemon* successfully, using an automatic network configuration, then clone its SD card for all the other Pis.

This quick start tutorial assumes you are using the Raspbian operating system on your Pis, and Ubuntu as your client.

1.3.1 Client Installation

Ensure your Ubuntu client machine is connected to the same network as your Pis (whether by Ethernet or Wifi doesn't matter). Then, execute the following to install the client and an NTP daemon:

```
$ sudo add-apt-repository ppa:waveform/ppa
$ sudo apt-get update
$ sudo apt-get install compoundpi-client ntp
```

The NTP daemon will most likely be installed to synchronize with an NTP pool on the Internet (e.g. `pool.ntp.org`). This is fine, but check that it's working with the following command line:

```
$ ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*aaaaaaa.aaaaaaa nn.nnn.nnn.nnn   3 u  109 1024  377    4.639   -2.101  21.233
```

1.3.2 Server Network Configuration

On the Pi you intend to clone, configure networking to use DHCP to automatically obtain an IP address. Edit the `/etc/network/interfaces` file and ensure that it looks similar to the following:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

This configuration should ensure that the first Ethernet and/or WiFi interfaces will pick up an address automatically from the local DHCP server. If you are using WiFi, complete the WiFi configuration by editing the `/etc/wpa_supplicant/wpa_supplicant.conf` file to look something like the following:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="my_wireless_ssid"
        psk="my_wireless_password"
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        auth_alg=OPEN
}
```

1.3.3 Server Installation

Execute the following command to install the Compound Pi server package and the NTP daemon (the latter is required for time-synchronized image capture):

```
$ sudo apt-get install compoundpi-server ntp
```

This should pull in all necessary dependencies, and automatically install an init-script which will start the Compound Pi daemon on boot-up. Test this by rebooting the Pi with a camera module attached. You should see the camera module's LED light up when the daemon starts. If it doesn't, the most likely culprit is the camera: try running **raspistill**, ensure you've activated the camera with **sudo raspi-config**, and ensure the CSI cable is inserted correctly. You can control the Compound Pi daemon as you would any other system daemon:

```
$ sudo service cpid stop
$ sudo service cpid start
$ sudo service cpid restart
```

Ideally, you want all your Pi servers to sync with the NTP time server you set up on your client. Edit the `/etc/ntp.conf` file and repalce the `server` lines with the IP address of your client (ideally you should configure your router to give your client a fixed address):

```
...
#server 0.debian.pool.ntp.org iburst
#server 1.debian.pool.ntp.org iburst
#server 2.debian.pool.ntp.org iburst
#server 3.debian.pool.ntp.org iburst
server 192.168.1.2
...
```

Restart the NTP daemon to use the new configuration:

```
$ sudo service ntp restart
```

1.3.4 Clone the SD Card

Once you've got a Pi running the Compound Pi daemon successfully, shut it down and place its SD card in any Linux machine with an SD card reader. Unmount any partitions that auto-mount, then figure out which device node represents the SD card. For example, the following would tell you that the SD card is sdd:

```
$ dmesg | tail | grep "Attached SCSI removable disk"
[    3.428459] sd 8:0:0:0: [sdd] Attached SCSI removable disk
```

Clone the SD card into a disk file:

```
$ sudo dd if=/dev/sdd of=server.img
```

This will take some considerable time to finish. Once it has done so, eject the source SD card and insert the target one in its place. Remember to unmount any partitions which auto-mount, then execute the reverse command:

```
$ sudo dd if=server.img of=/dev/sdd
```

Repeat this last step for all remaining target cards. Finally, install the SD cards in your set of Pi servers and boot them all to ensure their camera modules activate.

Warning: Ensure your target SD cards are the same size or larger than the source SD card. If they are larger, they will still appear the same size as the source after cloning because you the cloning also duplicates the partition table of the smaller device.

1.3.5 Testing the Servers

Back on the Ubuntu client machine, execute *cpi* to run the client. You will be presented with a command line like the following:

```
CompoundPi Client version 0.3
Type "help" for more information, or "find" to locate Pi servers
cpi>
```

Firstly, ensure that the network configuration is correct. The *config* command can be used to print the current configuration:

```
cpi> config
Setting       Value
------------- --------------
network       192.168.0.0/16
port          5647
bind          0.0.0.0:5647
timeout       5
capture_delay 0
capture_count 1
video_port    False
time_delta    0.25
output        /tmp
warnings      False
```

Assuming we're using a typical home router which gives out addresses in the 192.168.1.x network, this is incorrect. In order for broadcasts to work, the network *must* have the correct definition - it's no good having a superset configured (192.168.0.0/16 is a superset of 192.168.1.0/24). To correct the network definition, use the *set* command:

```
cpi> set network 192.168.1.0/24
cpi> config
Setting       Value
------------- --------------
network       192.168.1.0/24
port          5647
bind          0.0.0.0:5647
timeout       5
capture_delay 0
capture_count 1
video_port    False
time_delta    0.25
output        /tmp
warnings      False
```

To make permanent configuration changes, simply place them in a file named `~/.cpi.ini` like so:

```
[cpi]
network=192.168.1.0/24
timeout=10
output=~/Pictures
```

With the network configured correctly, you can now use *find* to locate your servers. If you run *find* on its own it will send out a broadcast ping and wait for a fixed number of seconds for servers to respond. If you know exactly how many servers you have, specify a number with the *find* command and it will warn you if it doesn't find that many servers (it will also finish faster if it does find the expected number of Pis):

```
cpi> find 2
Found 2 servers
```

You can query the status of your servers with the *status* command which will give you the basics for the camera configuration, the time according to the server, and the number of images currently stored in memory on the server. If you only want to query a specific set of servers you can give their addresses as a parameter:

```
cpi> status 192.168.1.154
Address        Mode        Shutter AWB    Exp  Meter   Flip Time Delta     #
-------------- ----------- ------- ------ ---- ------- ---- -------------- -
192.168.80.154 1280x720@30 auto    auto   auto average none 0:00:00        0
```

If any major discrepancies are detected (resolution, framerate, timestamp, etc.), the status command should notify you of them. The maximum discrepancy permitted in the timestamp is configured with the `time_delta` configuration setting.

To shoot an image, use the *capture* command:

```
cpi> capture
```

Finally, to download the captured images from all Pis, simply use the *download* command:

```
cpi> download
Downloaded image 0 from 192.168.1.154
Downloaded image 0 from 192.168.1.168
```

You can use the *config* and *set* commands to configure capture options, the download target directory, and so on.

Since version 0.3 a GUI client is also provided. The basic operations of the GUI client are essentially the same as the command line client, the only major difference being that download is performed automatically after capture. You can start the GUI client with the *cpigui* command.

1.3.6 Troubleshooting

Compound Pi provides some crude but effective tools for debugging problems. The first is simply that the daemon activates the camera by default. If you see a Pi server without the camera LED lit after boot-up, you know the daemon has failed to start for some reason.

The *identify* command is the main debugging tool provided by Compound Pi. If specified without any further parameters it will cause all discovered Pi servers to blink their camera LED for 5 seconds. Thus, if you run this command immediately after *find* you can quickly locate any Pi servers that were no discovered (typically this is due to misconfiguration of the network).

If *identify* is specified with one or more addresses, it will blink the LED on the specified Pi servers. This can be used to quickly figure out which address corresponds to which Pi (useful when dynamic addressing is used).

1.4 cpi

This is the Compound Pi client application which provides a command line interface through which you can query and interact with any Pi's running the *Compound Pi daemon* on your configured subnet. Use the *help* command within the application for information on the available commands.

The application can be configured via command line switches, a configuration file (defaults to `/etc/cpi.ini`, `/usr/local/etc/cpi.ini`, or `~/.cpid.ini`), or through the interactive command line itself.

1.4.1 Synopsis

```
cpi [-h] [--version] [-c CONFIG] [-q] [-v] [-l FILE] [-P] [-o PATH]
    [-n NETWORK] [-p PORT] [-b ADDRESS:PORT] [-t SECS]
    [--capture-delay SECS] [--capture-count NUM] [--video-port]
```

1.4.2 Description

-h, --help
show this help message and exit

--version
show program's version number and exit

-c CONFIG, **--config** CONFIG
specify a configuration file to load

-q, --quiet
produce less console output

-v, --verbose
produce more console output

-l FILE, **--log-file** FILE
log messages to the specified file

-P, --pdb
run under PDB (debug mode)

-o PATH, **--output** PATH
specifies the directory that downloaded images will be written to (default: `/tmp`)

-n NETWORK, **--network** NETWORK
specifies the network that the servers belong to (default: 192.168.0.0/16)

-p PORT, **--port** PORT
specifies the port that the servers will be listening on (default: 5647)

-b ADDRESS:PORT, **--bind** ADDRESS:PORT
specifies the address and port that the client listens on for downloads (default: 0.0.0.0:5647)

-t SECS, **--timeout** SECS
specifies the timeout (in seconds) for network transactions (default: 5)

--capture-delay SECS
specifies the delay (in seconds) used to synchronize captures. This must be less than the network timeout (default: 0)

--capture-count NUM
specifies the number of consecutive pictures to capture when requested (default: 1)

--video-port
if specified, use the camera's video port for rapid capture

1.4.3 Usage

The first command in a Compound Pi session is usually *find* to locate the servers on the specified subnet. If you know the number of servers available, specify it as an argument to the *find* command which will cause the command to return quicker in the case that all servers are found, or to warn you if less than the expected number are located.

The *status* command can be used to check that all servers have an equivalent camera configuration, and that time sync is reasonable.

The *capture* command is used to cause all located servers to capture an image. After capturing, use the *download* command to transfer all captured images to the client.

Finally, the *help* command can be used to query the available commands, and to obtain help on an individual command.

1.5 cpid

This is the server daemon for the Compound Pi application. Starting the application with no arguments starts the server in the foreground. The server can be configured through command line arguments or a configuration file (which defaults to `/etc/cpid.ini`, `/usr/local/etc/cpid.ini`, or `~/.cpid.ini`).

1.5.1 Synopsis

```
cpid [-h] [--version] [-c CONFIG] [-q] [-v] [-l FILE] [-P] [-b ADDRESS]
     [-p PORT] [-d] [-u UID] [-g GID] [--pidfile FILE]
```

1.5.2 Description

-h, --help
: show this help message and exit

--version
: show program's version number and exit

-c `CONFIG`, **--config** `CONFIG`
: specify a configuration file to load

-q, --quiet
: produce less console output

-v, --verbose
: produce more console output

-l `FILE`, **--log-file** `FILE`
: log messages to the specified file

-P, --pdb
: run under PDB (debug mode)

-b `ADDRESS`, **--bind** `ADDRESS`
: specifies the address to listen on for packets (default: 0.0.0.0)

-p `PORT`, **--port** `PORT`
: specifies the UDP port for the server to listen on (default: 5647)

-d, --daemon
: if specified, start as a background daemon

-u `UID`, **--user** `UID`
: specifies the user that the daemon should run as. Defaults to the effective user (typically root)

-g `GID`, **--group** `GID`
: specifies the group that the daemon should run as. Defaults to the effective group (typically root)

--pidfile `FILE`
: specifies the location of the pid lock file

1.5.3 Usage

The Compound Pi server is typically started at boot time by the init service. The Raspbian package includes an init script for this purpose. Users on other platforms will need to write their own init script.

When the server starts successfully it will initialize the camera and hold it open. This will prevent other applications from using the camera but also makes it easy to see that the server has started as the camera's LED will be lit (this is useful as Compound Pi servers are typically headless).

Note: If you explicitly set a user and/or group for the daemon (with the *`cpid -u`* and *`cpid -g`* options), be aware that using the Pi's camera typically requires membership of the `video` group. Furthermore, the specified user and group must have the ability to create and remove the pid lock file.

1.6 Client Commands

Each section below documents one of the commands available in the Compound Pi command line client. Many commands accept an address or list of addresses. Addresses must be specified in dotted-decimal format (no hostnames). Inclusive ranges of addresses are specified by two dash-separated addresses. Lists of addresses, or ranges of addresses are specified by comma-separating each list item.

The following table demonstrates various examples of this syntax:

Syntax	Expands To
`192.168.0.1`	192.168.0.1
`192.168.0.1-192.168.0.5`	192.168.0.1 192.168.0.2 192.168.0.3 192.168.0.4 192.168.0.5
`192.168.0.1,192.168.0.3`	192.168.0.1 192.168.0.3
`192.168.0.1,192.168.0.3-192.168.0.5`	192.168.0.1 192.168.0.3 192.168.0.4 192.168.0.5
`192.168.0.1-192.168.0.3,192.168.0.5`	192.168.0.1 192.168.0.2 192.168.0.3 192.168.0.5

It is also worth noting that if readline is installed (which it is on almost any modern Unix platform), the command line supports `Tab`-completion for commands and most parameters, including defined server addresses.

1.6.1 add

Syntax: add *addresses*

The *add* command is used to manually define the set of Pi servers to communicate with. Addresses can be specified individually, as a dash-separated range, or a comma-separated list of ranges and addresses.

See also: *find*, *remove*, *servers*.

```
cpi> add 192.168.0.1
cpi> add 192.168.0.1-192.168.0.10
cpi> add 192.168.0.1,192.168.0.5-192.168.0.10
```

1.6.2 awb

Syntax: awb *mode [addresses]*

The *awb* command is used to set the AWB mode of the camera on all or some of the defined servers. The mode can be one of the following:

- auto
- cloudy
- flash
- fluorescent
- horizon
- incandescent
- shade
- sunlight
- tungsten

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *exposure*, *metering*.

```
cpi> awb auto
cpi> awb fluorescent 192.168.0.1
cpi> awb sunlight 192.168.0.1-192.168.0.10
```

1.6.3 capture

Syntax: capture *[addresses]*

The *capture* command causes the servers to capture an image. Note that this does not cause the captured images to be sent to the client. See the *download* command for more information.

If no addresses are specified, a broadcast message to all defined servers will be used in which case the timestamp of the captured images are likely to be extremely close together. If addresses are specified, unicast messages will be sent to each server in turn. While this is still reasonably quick there will be a measurable difference between the timestamps of the last and first captures.

See also: *download*, *clear*.

```
cpi> capture
cpi> capture 192.168.0.1
cpi> capture 192.168.0.50-192.168.0.53
```

1.6.4 clear

Syntax: clear *[addresses]*

The *clear* command can be used to clear the in-memory image store on the specified Pi servers (or all Pi servers if no address is given). The *download* command automatically clears the image store after successful transfers so this command is only useful in the case that the operator wants to discard images without first downloading them.

See also: *download*, *capture*.

```
cpi> clear
cpi> clear 192.168.0.1-192.168.0.10
```

1.6.5 config

Syntax: config

The *config* command is used to display the current client configuration. Use the related *set* command to alter the configuration.

See also: *set*.

```
cpi> config
```

1.6.6 download

Syntax: download *[addresses]*

The *download* command causes each server to send its captured images to the client. Servers are contacted consecutively to avoid saturating the network bandwidth. Once images are successfully downloaded from a server, they are wiped from the server.

See also: *capture*, *clear*.

```
cpi> download
cpi> download 192.168.0.1
```

1.6.7 exit

Syntax: exit|quit

The *exit* command is used to terminate the application. You can also use the standard UNIX `Ctrl+D` end of file sequence to quit.

1.6.8 exposure

Syntax: exposure *mode [addresses]*

The *exposure* command is used to set the exposure mode of the camera on all or some of the defined servers. The mode can be one of the following:

- antishake
- auto
- backlight
- beach
- fireworks
- fixedfps
- night
- nightpreview
- snow
- sports
- spotlight
- verylong

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *awb*, *metering*.

```
cpi> exposure auto
cpi> exposure night 192.168.0.1
cpi> exposure backlight 192.168.0.1-192.168.0.10
```

1.6.9 find

Syntax: find *[count]*

The *find* command is typically the first command used in a client session to locate all Pis on the configured subnet. If a count is specified, the command will display an error if the expected number of Pis is not located.

See also: *add*, *remove*, *servers*, *identify*.

```
cpi> find
cpi> find 20
```

1.6.10 flip

Syntax: flip *value [addresses]*

The *flip* command is used to set the picture orientation on all or some of the defined servers. The following values can be specified:

- none
- horizontal
- vertical
- both

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*.

```
cpi> flip none
cpi> flip vertical 192.168.0.1
cpi> flip both 192.168.0.1-192.168.0.10
```

1.6.11 framerate

Syntax: framerate *rate [addresses]*

The *framerate* command is used to set the capture framerate of the camera on all or some of the defined servers. The rate can be specified as an integer, a floating-point number, or as a fractional value. The framerate of the camera influences the capture mode that the camera uses. See the camera hardware chapter of the picamera documentation for more information.

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *resolution*.

```
cpi> framerate 30
cpi> framerate 90 192.168.0.1
cpi> framerate 15 192.168.0.1-192.168.0.10
```

1.6.12 help

Syntax: help *[command]*

The 'help' command is used to display the help text for a command or, if no command is specified, it presents a list of all available commands along with a brief description of each.

1.6.13 identify

Syntax: identify *[addresses]*

The *identify* command can be used to locate a specific Pi server (or servers) by their address. It sends a command causing the camera's LED to blink on and off for 5 seconds. If no addresses are specified, the command will be sent to all defined servers (this can be useful after the *find* command to determine whether any Pi's failed to respond due to network issues).

See also: *find*.

```
cpi> identify
cpi> identify 192.168.0.1
cpi> identify 192.168.0.3-192.168.0.5
```

1.6.14 iso

Syntax: iso *value [addresses]*

The *iso* command is used to set the emulated ISO value of the camera on all or some of the defined servers. The value can be specified as an integer number between 0 and 1600, or `auto` which leaves the camera to determine the optimal ISO value.

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *exposure*.

```
cpi> iso auto
cpi> iso 100 192.168.0.1
cpi> iso 800 192.168.0.1-192.168.0.10
```

1.6.15 levels

Syntax: levels *brightness contrast saturation [addresses]*

The *levels* command is used to simultaneously set the brightness, contrast, and saturation levels on all or some of the defined servers. Each level is specified as an integer number between 0 and 100. The default for each level is 50.

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*.

```
cpi> levels 50 50 50
cpi> levels 70 50 50 192.168.0.1
cpi> levels 40 60 70 192.168.0.1-192.168.0.10
```

1.6.16 metering

Syntax: metering *mode [addresses]*

The *metering* command is used to set the metering mode of the camera on all or some of the defined servers. The mode can be one of the following:

- average
- backlit
- matrix
- spot

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *awb*, *exposure*.

```
cpi> metering average
cpi> metering spot 192.168.0.1
cpi> metering backlit 192.168.0.1-192.168.0.10
```

1.6.17 quit

Syntax: exit|quit

The *exit* command is used to terminate the application. You can also use the standard UNIX `Ctrl+D` end of file sequence to quit.

1.6.18 remove

Syntax: remove *addresses*

The *remove* command is used to remove addresses from the set of Pi servers to communicate with. Addresses can be specified individually, as a dash-separated range, or a comma-separated list of ranges and addresses.

See also: *add*, *find*, *servers*.

```
cpi> remove 192.168.0.1
cpi> remove 192.168.0.1-192.168.0.10
cpi> remove 192.168.0.1,192.168.0.5-192.168.0.10
```

1.6.19 resolution

Syntax: resolution *width x height [addresses]*

The *resolution* command is used to set the capture resolution of the camera on all or some of the defined servers. The resolution of the camera influences the capture mode that the camera uses. See the camera hardware chapter of the picamera documentation for more information.

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *framerate*.

```
cpi> resolution 640x480
cpi> resolution 1280x720 192.168.0.54
cpi> resolution 1280x720 192.168.0.1,192.168.0.3
```

1.6.20 servers

Syntax: servers

The *servers* command is used to list the set of servers that the client expects to communicate with. The content of the list can be manipulated with the *find*, *add*, and *remove* commands.

See also: *find*, *add*, *remove*.

```
cpi> servers
```

1.6.21 set

Syntax: set *name value*

The *set* command is used to alter the value of a client configuration variable. Use the related *config* command to view the current configuration.

See also: *config*.

```
cpi> set timeout 10
cpi> set output ~/Pictures/
cpi> set capture_count 5
```

1.6.22 shutter

Syntax: shutter *speed [addresses]*

The *shutter* command is used to set the shutter speed of the camera on all or some of the defined servers. The speed can be specified as a floating-point number (in milli-seconds), or `auto` which leaves the camera to determine the shutter speed. The *framerate* of the camera limits the shutter speed that can be set. For example, if framerate is 30fps, then shutter speed cannot be slower than 33.333ms.

If no address is specified then all currently defined servers will be targetted. Multiple addresses can be specified with dash-separated ranges, comma-separated lists, or any combination of the two.

See also: *status*, *resolution*, *framerate*.

```
cpi> shutter auto
cpi> shutter 33.333 192.168.0.1
cpi> shutter 100 192.168.0.1-192.168.0.10
```

1.6.23 status

Syntax: status *[addresses]*

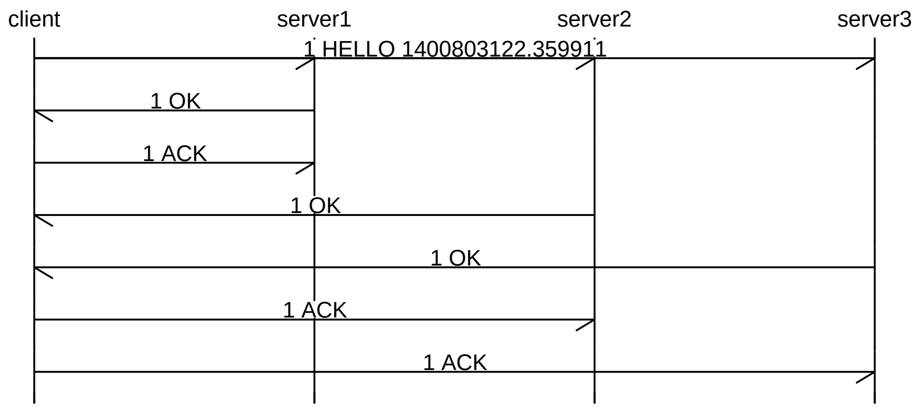
The *status* command is used to retrieve configuration information from servers. If no addresses are specified, then all defined servers will be queried.

See also: *resolution*, *framerate*.

```
cpi> status
```

1.7 Network Protocol

The Compound Pi network protocol is UDP-based, utilizing broadcast or unicast packets for commands, and unicast packets for responses. File transfers (as initiated by the *download* command in the client) are TCP-based. The diagram below shows a typical conversation between a Compound Pi client and three servers involving a broadcast PING packet and the resulting responses:



All messages are encoded as ASCII text. Command messages consist of a non-zero positive integer sequence number followed by a single space, followed by the command in capital letters, optionally followed by space separated parameters for the command. The following are all valid examples of command messages:

```
1 HELLO 1400803122.359911

2 CLEAR

3 CAPTURE 1 0

4 STATUS

5 LIST

6 SEND 0 5647

7 FOO
```

In other words, the generic form of a command message is:

```
<sequence-number> <command> [parameter1] [parameter2]...
```

Response messages (from the servers to the client) consist of a non-zero positive integer sequence number (copied from the corresponding command), followed by a single space, followed by `OK` if the command's execution was successful, optionally followed by a new-line character (ASCII character 10), and any data the response is expected to include. For example:

```
1 OK
VERSION 0.3

2 OK

3 OK

4 OK
RESOLUTION 1280 720
FRAMERATE 30
SHUTTERSPEED 0
AWB auto
EXPOSURE auto 0
ISO 0
METERING average
LEVELS 50 0 0
FLIP 0 0
TIMESTAMP 1400803173.991651
IMAGES 1

5 OK
IMAGE 0 1400803173.012543 8083879

6 OK
```

In the case of an error, the response message consists of a non-zero positive integer sequence number (copied from the corresponding command), followed by a single space, followed by `ERROR`, followed by a new-line character (ASCII character 10), followed by a description of the error that occurred:

```
7 ERROR
Unknown command FOO
```

In other words, the general form of a response message is:

```
<sequence-number> OK
<data>
```

Or, if an error occurred:

```
<sequence-number> ERROR
<error-description>
```

Sequence numbers start at 1 (0 is reserved), and are incremented on each command, except for *ACK* and *HELLO*. The sequence number for a response indicates which command the response is associated with and likewise the sequence number for *ACK* indicates the response that the *ACK* terminates. The *HELLO* command, being the command that begins a session specifies a new starting sequence number for the server.

As UDP is an unreliable protocol, some mechanism is required to compensate for lost, unordered, or duplicated packets. All transmissions (commands and responses) are repeated with random delays. The sequence number associated with a client command permits servers to ignore repeated commands that they have already seen. Likewise, the sequence number of the server response permits clients to ignore repeated responses they have already seen.

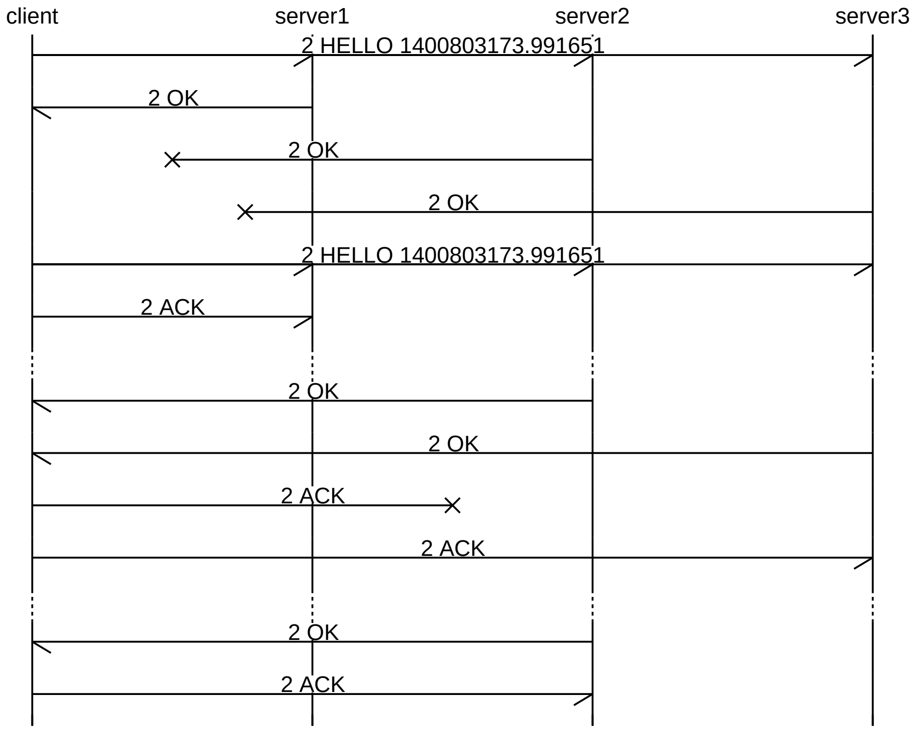
Commands are repeated by the client until it has received a response from the targetted server(s) (all located servers on the subnet in the case of broadcast messages), or until a timeout has elapsed (5 seconds by default).

Responses are repeated by a server until it receives an ACK from the client with a corresponding sequence number, or until a timeout has elapsed (5 seconds by default).

An exception to the above is the *HELLO* command. Because this command sets a new sequence number, servers cannot use the sequence number to detect repeated packets. Hence, the *HELLO* command includes the timestamp at the client issuing it as a command parameter. Servers must use this timestamp to detect stale or repeated instances of this messsage. The timestamp can be assumed to be incrementing (like a monotonic clock); in the current implementation it isn't but this doesn't matter much given how rarely this message is issued in a session.

1.7.1 Example

In the following example, the client broadcasts a *HELLO* command to three servers. The servers all respond with an OK response, but only the packet from server1 makes it back to the client. The server resends the HELLO command but this is ignored by the servers as they've seen the included timestamp before. The client responds to server1 with an *ACK*. The other servers (after a random delay) now retry their OK responses and both get through this time. The client responds with an ACK for server3, but the ACK for server2 is lost. After another random delay, server2 once again retries its OK response, causing the client to send another ACK which succeeds this time:



The following sections document the various commands that the server understands and the expected responses.

1.7.2 ACK

Syntax: ACK

The *ACK* command is sent by the client to acknowledge receipt of a response from a server. It is special in that its sequence number must match the sequence number of the response that it acknowledges (it is the only command that does not increment the sequence number on the client).

It is also special in that its implementation is effectively optional: a client doesn't *have* to acknowledge receipt of a server's response; after 5 seconds, the server will stop retrying its responses anyway, but an *ACK* command is nonetheless useful to reduce the congestion of the network with useless response retries. It is also the only client message which is not automatically repeated (as its only purpose is to silence the auto-repeating of a response in order to reduce network congestion).

When a server receives the *ACK* command, it must stop retrying responses with the same sequence number as the ACK command. No other response should be sent.

1.7.3 AWB

Syntax: AWB *mode [red blue]*

The *AWB* command changes the camera's auto-white-balance mode which is provided as a lower case string. If the string is `'off'` then manual red and blue gains may additionally be specified as floating point values between 0.0 and 8.0.

An OK response is expected with no data.

1.7.4 BLINK

Syntax: BLINK

The *BLINK* command should cause the server to identify itself for the purpose of debugging. In this implementation, this is accomplished by blinking the camera's LED for 5 seconds.

An OK response is expected with no data.

1.7.5 CAPTURE

Syntax: CAPTURE *[count [video-port [sync]]]*

The *CAPTURE* command should cause the server to capture one or more images from the camera. The parameters are as follows:

count Specifies the number of images to capture. If specified, this must be a non-zero positive integer number. If not specified, defaults to 1.

video-port Specifies which port to capture from. If unspecified, or 0, the still port should be used (resulting in the best quality capture, but may cause significant delay between multiple consecutive shots). If 1, the video port should be used.

sync Specifies the timestamp at which the capture should be taken. The timestamp's form is UNIX time: the number of seconds since the UNIX epoch specified as a dotted-decimal. The timestamp must be in the future, and it is important for the server's clock to be properly synchronized in order for this functionality to operate correctly. If unspecified, the capture should be taken immediately upon receipt of the command.

The image(s) taken in response to the command should be stored locally on the server until their retrieval is requested by the *SEND* command. The timestamp at which the image was taken must also be stored. Storage in this implementation is simply in RAM, but implementations are free to use any storage medium they see fit.

An OK response is expected with no data.

1.7.6 CLEAR

Syntax: CLEAR

The *CLEAR* command deletes all images from the server's local storage. As noted above in *CAPTURE*, implementations are free to use any storage medium, but the current implementation simply uses a list in RAM.

An OK response is expected with no data.

1.7.7 EXPOSURE

Syntax: EXPOSURE *mode compensation*

The *EXPOSURE* command changes the camera's exposure mode and compensation value. The mode is provided as a lower case string. The compensation value is an integer number between -24 and 24.

An OK response is expected with no data.

1.7.8 FLIP

Syntax: FLIP *horizontal vertical*

The *FLIP* command changes the camera's orientation. The horizontal and vertical parameters must be integer numbers which will be interpreted as booleans (0 being false, anything else true).

An OK response is expected with no data.

1.7.9 FRAMERATE

Syntax: FRAMERATE *num[/denom]*

The *FRAMERATE* command changes the camera's configuration to use the specified framerate which is given either as an integer number between 1 and 90 or as a fraction consisting of an integer numerator and denominator separated by a forward-slash.

An OK response is expected with no data.

1.7.10 HELLO

Syntax: HELLO *timestamp*

The *HELLO* command is sent by the client's *find* command in order to locate Compound Pi servers. The server must send the following string in the data portion of the OK response indicating the version of the protocol that the server understands:

```
VERSION 0.3
```

The server must use the sequence number of the command as the new starting sequence number (i.e. HELLO resets the sequence number on the server). For this reason, the sequence number cannot be used to detect repeated HELLO commands. Instead the timestamp parameter should be used for this purpose: the timestamp can be assumed to be incrementing hence HELLO commands from a particular host with a timestamp less than or equal to one already seen can be ignored.

Warning: As Compound Pi is a project in its infancy, the protocol version is currently the project's version and no attempt will be made to preserve backward (or forward) compatibility in the protocol until version 1.0 is released. In the current version, the client crudely compares the version in the response with its own version and rejects anything that doesn't match precisely.

1.7.11 ISO

Syntax: ISO *level*

The *ISO* command changes the camera's emulated ISO level. The new level is provided as an integer number where 0 indicates automatic ISO level.

An OK response is expected with no data.

1.7.12 LEVELS

Syntax: LEVELS *brightness contrast saturation*

The *LEVELS* command changes the camera's brightness, contrast, and saturation levels. The new levels are given as integer numbers between 0 and 50 for brightness, or -100 to 100 for contrast and saturation.

An OK response is expected with no data.

1.7.13 LIST

Syntax: LIST

The *LIST* command causes the server to respond with a new-line separated list detailing all locally stored images. Each line in the data portion of the response has the following format:

```
IMAGE <number> <timestamp> <size>
```

For example, if five images are stored on the server the data portion of the OK response may look like this:

```
IMAGE 0 1398618927.307944 8083879
IMAGE 1 1398619000.53127 7960423
IMAGE 2 1398619013.658935 7996156
IMAGE 3 1398619014.122921 8061197
IMAGE 4 1398619014.314919 8053651
```

The `number` portion of the line is a zero-based integer index for the image which can be used with the *SEND* command to retrieve the image data. The `timestamp` portion is in UNIX-time format: a dotted-decimal value of the number of seconds since the UNIX epoch. Finally, the `size` portion is an integer number indicating the number of bytes in the image.

1.7.14 METERING

Syntax: METERING *mode*

The *METERING* command changes the camera's light metering mode. The new mode is provided as a lower case string.

An OK response is expected with no data.

1.7.15 RESOLUTION

Syntax: RESOLUTION *width height*

The *RESOLUTION* command changes the camera's configuration to use the specified capture resolution which is two integer numbers giving the width and height of the new resolution.

An OK response is expected with no data.

1.7.16 SEND

Syntax: SEND *index port*

The *SEND* command causes the specified image to be sent from the server to the client. The parameters are as follows:

index Specifies the zero-based index of the image that the client wants the server to send. This must match one of the indexes output by the *LIST* command.

port Specifies the TCP port on the client that the server should connect to in order to transmit the image data. This is given as an integer number (never a service name).

Assuming *index* refers to a valid image index, the server must connect to the specified TCP port on the client, send the bytes of the image, and finally close the connection. The server must also send an OK response with no data.

1.7.17 SHUTTERSPEED

Syntax: SHUTTERSPEED *speed*

The *SHUTTERSPEED* command changes the camera's configuration to use the specified shutter speed which is given as an integer number between 0 and 1000000 (where 0 indicates automatic shutter speed).

An OK response is expected with no data.

1.7.18 STATUS

Syntax: STATUS

The *STATUS* command causes the server to send the client information about its current configuration. Specifically, the response must contain the following lines in its data portion, in the order given below:

```
RESOLUTION <width> <height>
FRAMERATE <num>[/denom]
SHUTTERSPEED <speed>
AWB <awb_mode>
EXPOSURE <exposure_mode> <exposure_comp>
ISO <iso>
METERING <metering_mode>
LEVELS <brightness> <contrast> <saturation>
FLIP <hflip> <vflip>
TIMESTAMP <time>
IMAGES <images>
```

Where:

<width> <height> Gives the camera's currently configured capture resolution

<num>[/denom] Gives the camera's currently configured framerate as an integer number or fractional value

<speed> Gives the camera's currently configured shutter speed as an integer number between 0 and 1000000 (where 0 indicates automatic)

<awb_mode> Gives the camera's current auto-white-balance mode as a lower case string

<exposure_mode> Gives the camera's current exposure mode as a lower case string

<exposure_comp> Gives the camera's current exposure compensation value as an integer number between -24 and 24 (each increment represents 1/6th of a stop)

<iso> Gives the camera's current ISO setting as an integer number between 0 and 1600 (where 0 indicates automatic)

<metering_mode> Gives the camera's current light metering mode as a lower case string

<brightness> Gives the camera's current brightness setting as an integer value between 0 and 100 (50 is the default)

<contrast> Gives the camera's current contrast setting as an integer between -100 and 100 (0 is the default)

<saturation> Gives the camera's current saturation setting as an integer between -100 and 100 (0 is the default)

<hflip>* and *<vflip> Gives the camera's orientation as 1 or 0 (indicating the flip is or is not active respectively)

<time> Gives the timestamp at which the *STATUS* command was received in UNIX time format (a dotted-decimal number of seconds since the UNIX epoch).

<images> Gives the number of images currently stored locally by the server.

For example, the data portion of the OK response may look like the following:

```
RESOLUTION 1280 720
FRAMERATE 30
SHUTTERSPEED 0
AWB auto
EXPOSURE auto 0
ISO 0
METERING average
```

```
LEVELS 50 0 0
FLIP 0 0
TIMESTAMP 1400803173.991651
IMAGES 1
```

1.8 Change log

1.8.1 Release 0.3 (2014-05-23)

Several major enhancements in this release:

- A GUI client (cpigui) is now included. This is currently undocumented, but should be pretty intuitive to anyone familiar with the command line interface (#3)
- Both clients and the server now support many more camera settings including white-balance, exposure, ISO, shutter speed, etc (#12)
- All UDP messages (client and server) are now retried to ensure reliability, particularly during multiple unicast messages (#13)

1.8.2 Release 0.2 (2014-04-27)

Several improvements in this release:

- The network protocol has been changed to enhance its reliability when dealing with lots of Pis on unreliable networks (like Wifi)
- The status command has been enhanced to warn of configuration discrepancies.
- Lots more work on the docs

1.8.3 Release 0.1 (2014-04-15)

Initial release

1.9 License

CHAPTER 2

Indices and Tables

- *genindex*
- *search*