pibootctl attr: pibootctl.$_{version}$ $_{Documentation}$

Dave Jones

Sep 09, 2020

Contents

CHAPTER 1

Installation

If your distribution provides pibootctl then you should either find the utility is installed by default, or it should be installable via your package manager. For example:

```
$ sudo apt install pibootctl
```

It is strongly recommended to use a provided package rather than installing from PyPI as this will include configuration specific to your distribution. The utility can be removed via the usual mechanism for your package manager. For instance:

```
$ sudo apt purge pibootctl
```

1.1 Configuration

pibootctl looks for its configuration in three locations:

1. `/lib/pibootctl/pibootctl.conf`
2. `/etc/pibootctl.conf`
3. `~/.config/pibootctl.conf`

The last location is only intended for use by people developing pibootctl; for the vast majority of users the configuration should be provided by their distribution in one of the first two locations.

The configuration file is a straight-forward INI-style containing a single section titled "defaults". A typical configuration file might look like this:

Listing 1: pibootctl.conf

```
[defaults]
boot_path = /boot
store_path = pibootctl
package_name = pibootctl
comment_lines = on
backup = on
```

The configuration specifies several settings, but the most important are:

`boot_path` The mount-point of the boot partition (defaults to `/boot`).

`store_path` The path under which to store saved boot configurations, relative to `boot_path` (defaults to `pibootctl`).

`config_root` The "root" configuration file which is read first, relative to `boot_path` (defaults to `config.txt`). This is also the primary file that gets re-written when settings are changed.

`mutable_files` The set of files within a configuration that may be modified by the utility (defaults to `config.txt`). List multiple files on separate lines. Currently, this *must* include `config.txt`.

`comment_lines` If this is on, when lines in configuration files are no longer required, they will be commented out with a "#" prefix instead of being deleted. Defaults to off.

Note that, regardless of this setting, the utility will always search for commented lines to uncomment before writing new ones.

`reboot_required` The file which should be created in the event that the active boot configuration is changed.

`reboot_required_pkgs` The file to which the value of `package_name` should be appended in the event that the active boot configuration is changed.

`package_name` The name of the package which contains the utility. Used by `reboot_required_pkgs`.

`backup` If this is on (the default), any attempt to change the active boot configuration will automatically create a backup of that configuration if one does not already exist.

Line comments can be included in the configuration file with a `#` prefix. Another example configuration, typical for Ubuntu on the Raspberry Pi, is shown below:

Listing 2: pibootctl.conf

```
[defaults]
boot_path = /boot
store_path = pibootctl
mutable_files =
  config.txt
  syscfg.txt

reboot_required = /var/run/reboot-required
reboot_required_pkgs = /var/run/reboot-required.pkgs
package_name = pibootctl
backup = on
```

CHAPTER 2

User Manual

The `pibootctl` utility defines several commands which can be used to query and manipulate the boot configuration of the Raspberry Pi:

diff **(page 5)** Display the differences between the specified boot configuration and the current one, or another specified configuration.

get **(page 6)** Retrieve the value of specified setting(s).

help **(page 7)** The default command, which describes the specified command or configuration setting.

list **(page 9)** List the stored boot configurations.

load **(page 10)** Restore the named boot configuration to be used at the next boot.

remove **(page 11)** Delete the specified boot configuration.

rename **(page 12)** Rename the specified boot configuration.

save **(page 13)** Save the current boot configuration to the specified name.

set **(page 14)** Modify or reset the specified configuration setting(s).

show **(page 15)** Show the specified stored configuration.

status **(page 17)** Output the current boot configuration; by default this only prints modified settings.

Typically, the *status* (page 17) command is the first used, to determine the current boot configuration:

```
$ pibootctl status
+------------------------+-------+
| Name                   | Value |
|------------------------+-------|
| i2c.enabled            | on    |
| spi.enabled            | on    |
| video.overscan.enabled | off   |
+------------------------+-------+
```

After which the *save* (page 13) command might be used to take a backup of the configuration before editing it with the *set* (page 14) command:

```
$ sudo pibootctl save default
$ sudo pibootctl set camera.enabled=on gpu.mem=128
$ sudo pibootctl save cam
```

Note: Note that commands which modify the content of the boot partition (e.g. *save* (page 13) and *set* (page 14)) are executed with **sudo** as root privileges are typically required.

The configuration of **pibootctl** itself dictates where the stored configurations are placed on disk. By default this is under a "pibootctl" directory on the boot partition, but this can be changed in the **pibootctl** configuration. The application attempts to read its configuration from the following locations on startup:

- `/lib/pibootctl/pibootctl.conf`
- `/etc/pibootctl.conf`
- `$XDG_CONFIG_HOME/pibootctl.conf`

The final location is only intended for developers working on **pibootctl** itself. The others should be used by packages providing pibootctl on your chosen OS.

Stored boot configurations are simply PKZIP[1] files containing the files that make up the boot configuration (sometimes this is just the `config.txt` file, and sometimes other files may be included).

Note: In the event that your system is unable to boot (e.g. because of mis-configuration), you can restore a stored boot configuration simply by unzipping the stored configuration back into the root of the boot partition.

In other words, you can simply place your Pi's SD card in a Windows or MAC OS X computer which should automatically mount the boot partition (which is the only partition that these OS' will understand on the card), find the "pibootctl" folder and under there you should see all your stored configurations as .zip files. Unzip one of these into the folder above "pibootctl", overwriting files as necessary and you have restored your boot configuration.

The *diff* (page 5) command can be used to discover the differences between boot configurations:

```
$ pibootctl diff default
+------------------------+---------------+-------------+
| Name                   | <Current>     | default     |
|------------------------+---------------+-------------|
| boot.firmware.filename | 'start_x.elf' | 'start.elf' |
| boot.firmware.fixup    | 'fixup_x.dat' | 'fixup.dat' |
| camera.enabled         | on            | off         |
| gpu.mem                | 128 (Mb)      | 64 (Mb)     |
+------------------------+---------------+-------------+
```

Note: Some settings indirectly affect others. Even though we did not explicitly set `boot.firmware.filename`, setting `camera.enabled` affected its default value.

The *help* (page 7) command can be used to display the help screen for each sub-command:

```
$ pibootctl help save
usage: pibootctl save [-h] [-f] name

Store the current boot configuration under a given name.
```

(continues on next page)

[1] https://en.wikipedia.org/wiki/Zip_(file_format)

(continued from previous page)

```
positional arguments:
  name         The name to save the current boot configuration under; can
               include any characters legal in a filename

optional arguments:
  -h, --help   show this help message and exit
  -f, --force  Overwrite an existing configuration, if one exists
```

Additionally, *help* (page 7) will accept setting names to display information about the defaults and underlying commands each setting represents:

```
$ pibootctl help camera.enabled
      Name: camera.enabled
   Default: off
Command(s): start_x, start_debug, start_file, fixup_file

Enables loading the Pi camera module firmware. This implies that
start_x.elf (or start4x.elf) will be loaded as the GPU firmware rather than
the default start.elf (and the corresponding fixup file).

Note: with the camera firmware loaded, gpu.mem must be 64Mb or larger
(128Mb is recommended for most purposes; 256Mb may be required for complex
processing pipelines).
```

The *list* (page 9) command can be used to display the content of the configuration store, and *load* (page 10) to restore previously saved configurations:

```
$ pibootctl list
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| cam     | x      | 2020-03-11 21:29:56 |
| default |        | 2020-03-11 21:29:13 |
+---------+--------+---------------------+
$ sudo pibootctl load default
```

2.1 diff

2.1.1 Synopsis

```
pibootctl diff [-h] [--json | --yaml | --shell] [left] right
```

2.1.2 Description

Display the settings that differ between two stored boot configurations, or between one stored boot configuration and the current configuration.

2.1.3 Options

left
: The boot configuration to compare from, or the current configuration if omitted.

right
The boot configuration to compare against.

-h, --help
Show a brief help page for the command.

--json
Use JSON as the output format.

--yaml
Use YAML as the output format.

--shell
Use a tab-delimited output format suitable for the shell.

2.1.4 Usage

The **diff** command is used to display the differences between two boot configurations; either two stored configurations (if two names are supplied on the command line), or between the current boot configuration and a stored one (if one name is supplied on the command line):

```
$ sudo pibootctl save default
$ sudo pibootctl set video.hdmi0.group=1 video.hdmi0.mode=4
$ pibootctl diff default
+------------------+----------------+--------------------+
| Name             | <Current>      | default            |
|------------------+----------------+--------------------|
| video.hdmi0.group | 1 (CEA)        | 0 (auto from EDID) |
| video.hdmi0.mode  | 4 (720p @60Hz) | 0 (auto from EDID) |
+------------------+----------------+--------------------+
$ sudo pibootctl save 720p
$ pibootctl diff default 720p
+------------------+--------------------+----------------+
| Name             | default            | 720p           |
|------------------+--------------------+----------------|
| video.hdmi0.group | 0 (auto from EDID) | 1 (CEA)        |
| video.hdmi0.mode  | 0 (auto from EDID) | 4 (720p @60Hz) |
+------------------+--------------------+----------------+
```

For developers wishing to build on top of pibootctl, options are provided to produce the output in JSON (*--json* (page 6)), YAML (*--yaml* (page 6)), and shell-friendly (*--shell* (page 6)):

```
$ pibootctl diff --json default 720p
{"video.hdmi0.mode": {"right": 4, "left": 0}, "video.hdmi0.group":
{"right": 1, "left": 0}}
```

2.2 get

2.2.1 Synopsis

```
pibootctl get [-h] [--json | --yaml | --shell] setting [setting ...]
```

2.2.2 Description

Query the status of one or more boot configuration settings. If a single setting is requested then just that value is output. If multiple values are requested then both setting names and values are output.

This applies whether output is in the default, JSON, YAML, or shell-friendly styles.

2.2.3 Options

setting
: The name(s) of the setting(s) to query; if a single setting is given its value alone is output, if multiple settings are queried the names and values of the settings are output.

-h, --help
: Show a brief help page for the command.

--json
: Use JSON as the output format.

--yaml
: Use YAML as the output format.

--shell
: Use a var=value output format suitable for the shell.

2.2.4 Usage

The **get** command is primarily of use to those wishing to build something on top of **pibootctl**; for end users wishing to query the current boot configuration the *status* (page 17) command is of more use. When given a single setting to query the value of that setting is output on its own, in whatever output style is selected:

```
$ pibootctl get video.overscan.enabled
on
$ pibootctl get --json video.overscan.enabled
true
```

When given multiple settings, names and values of those settings are both output:

```
$ pibootctl get serial.enabled serial.baud serial.uart
+----------------+-------------------------+
| Name           | Value                   |
|----------------+-------------------------|
| serial.baud    | 115200                  |
| serial.enabled | on                      |
| serial.uart    | 0 (/dev/ttyAMA0; PL011) |
+----------------+-------------------------+
$ pibootctl get --json serial.enabled serial.baud serial.uart
{"serial.enabled": true, "serial.baud": 115200, "serial.uart": 0}
```

Note that wildcards are not permitted with this command, unlike with the *status* (page 17) command.

2.3 help

2.3.1 Synopsis

```
pibootctl help [-h] [command | setting]
```

2.3.2 Description

With no arguments, displays the list of **pibootctl** commands. If a command name is given, displays the description and options for the named command. If a setting name is given, displays the description and default value for that setting.

2.3.3 Options

-h, --help
Show a brief help page for the command.

command
The name of the command to output help for. The full command name must be given; abbreviations are not accepted.

setting
The name of the setting to output help for.

If the setting is not recognized, and contains an underscore ('_') character, the utility will assume it is a config.txt configuration command and attempt to output help for the setting that corresponds to it. If multiple settings correspond, their names will be printed instead.

2.3.4 Usage

The **help** command is the default command, and thus will be invoked if **pibootctl** is called with no other arguments. However it can also be used to retrieve help for specific commands:

```
$ pibootctl help ls
usage: pibootctl list [-h] [--json | --yaml | --shell]

List all stored boot configurations.

optional arguments:
  -h, --help  show this help message and exit
  --json      Use JSON as the format
  --yaml      Use YAML as the format
  --shell     Use a var=value or tab-delimited format suitable for the
              shell
```

Alternatively, it can be used to describe settings:

```
$ pibootctl help boot.debug.enabled
      Name: boot.debug.enabled
   Default: off
Command(s): start_debug, start_file, fixup_file

Enables loading the debugging firmware. This implies that start_db.elf (or
start4db.elf) will be loaded as the GPU firmware rather than the default
start.elf (or start4.elf). Note that the debugging firmware incorporates
the camera firmware so this will implicitly switch camera.enabled on if it
is not already.

The debugging firmware performs considerably more logging than the default
firmware but at a performance cost, ergo it should only be used when
required.
```

Finally, if you are more familiar with the "classic" boot configuration commands, it can be used to discover which **pibootctl** settings correspond to those commands:

```
$ pibootctl help start_file
start_file is affected by the following settings:

camera.enabled
boot.debug.enabled
boot.firmware.filename
```

2.4 list

2.4.1 Synopsis

```
pibootctl list [-h] [--json | --yaml | --shell]
```

2.4.2 Description

List all stored boot configurations.

2.4.3 Options

-h, --help
Show a brief help page for the command.

--json
Use JSON as the output format.

--yaml
Use YAML as the output format.

--shell
Use a tab-delimited output format suitable for the shell.

2.4.4 Usage

The **list** command is used to display the content of the store of boot configurations:

```
$ pibootctl list
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| 720p    | x      | 2020-03-10 11:33:24 |
| default |        | 2020-03-10 11:32:12 |
| dpi     |        | 2020-02-01 15:46:48 |
| gpi     |        | 2020-02-01 16:13:02 |
+---------+--------+---------------------+
```

If one (or more) of the stored configurations match the current boot configuration, this will be indicated in the "Active" column. Note that equivalence is based on a hash of all files in the configuration, not on the resulting settings. Hence a simple edit like, for example, reversing the order of two lines (which might not make any difference to the resulting settings) would be sufficient to mark the configuration as "different".

The "timestamp" of a stored configuration is the last modification date of that configuration (calculated as the latest modification date of all files within the configuration).

For developers wishing to build on top of pibootctl, options are provided to produce the output in JSON (*--json* (page 9)), YAML (*--yaml* (page 9)), and shell-friendly (*--shell* (page 9)). These combine with all aforementioned options as expected:

```
$ pibootctl list --json
[{"timestamp": "2020-02-01T15:46:48", "active": false, "name": "dpi"},
{"timestamp": "2020-03-10T11:32:12", "active": false, "name": "default"},
{"timestamp": "2020-02-01T16:13:02", "active": false, "name": "gpi"},
{"timestamp": "2020-03-10T11:33:24", "active": true, "name": "720p"}]
```

2.5 load

2.5.1 Synopsis

```
pibootctl load [-h] [--no-backup] name
```

2.5.2 Description

Overwrite the current boot configuration with a stored one.

2.5.3 Options

name
: The name of the boot configuration to restore

-h, --help
: Show a brief help page for the command.

--no-backup
: Don't take an automatic backup of the current boot configuration if one doesn't exist

2.5.4 Usage

The **load** command is used to replace the current boot configuration with one previously stored. Effectively this simply unpacks the PKZIP[2] of the stored boot configuration into the boot partition, overwriting existing files.

If the current boot configuration has not been stored (with the *save* (page 13) command), an automatically named backup will be saved first:

```
$ sudo pibootctl save default
$ sudo pibootctl set video.hdmi0.group=1 video.hdmi0.mode=4
$ sudo pibootctl load default
Backed up current configuration in backup-20200310-095646
```

This can be avoided with the *--no-backup* (page 10) option.

Warning: The command is written to guarantee that no files will ever be left half-written (files are unpacked to a temporary filename then atomically moved into their final location overwriting any existing file).

[2] https://en.wikipedia.org/wiki/Zip_(file_format)

However, the utility cannot guarantee that in the event of an error, the configuration as a whole is not half-written (i.e. that one or more files failed to unpack). In other words, in the event of failure you cannot assume that the boot configuration is consistent.

2.6 remove

2.6.1 Synopsis

```
pibootctl remove [-h] [-f] name
```

2.6.2 Description

Remove a stored boot configuration.

2.6.3 Options

name
The name of the boot configuration to remove.

-h, --help
Show a brief help page for the command.

-f, --force
Ignore errors if the named configuration does not exist.

2.6.4 Usage

The **remove** command is used to delete a stored boot configuration:

```
$ pibootctl list
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| 720p    | x      | 2020-03-10 11:33:24 |
| default |        | 2020-03-10 11:32:12 |
| dpi     |        | 2020-02-01 15:46:48 |
| gpi     |        | 2020-02-01 16:13:02 |
+---------+--------+---------------------+
$ sudo pibootctl remove gpi
$ pibootctl list
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| 720p    | x      | 2020-03-10 11:33:24 |
| default |        | 2020-03-10 11:32:12 |
| dpi     |        | 2020-02-01 15:46:48 |
+---------+--------+---------------------+
```

If, for scripting purposes, you wish to ignore the error in the case the specified stored configuration does not exist, use the *--force* (page 11) option:

```
$ pibootctl rm foo
unknown configuration foo
$ pibootctl rm -f foo
```

2.7 rename

2.7.1 Synopsis

```
pibootctl rename [-h] [-f] name to
```

2.7.2 Description

Rename a stored boot configuration.

2.7.3 Options

name
: The name of the boot configuration to rename.

to
: The new name of the boot configuration.

-h, --help
: Show a brief help page for the command.

-f, --force
: Overwrite the target configuration, if it exists.

2.7.4 Usage

The **rename** command can be used to change the name of a stored boot configuration:

```
$ pibootctl ls
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| 720p    | x      | 2020-03-10 11:33:24 |
| default |        | 2020-03-10 11:32:12 |
| dpi     |        | 2020-02-01 15:46:48 |
+---------+--------+---------------------+
$ sudo pibootctl rename default foo
$ pibootctl ls
+------+--------+---------------------+
| Name | Active | Timestamp           |
|------+--------+---------------------|
| 720p | x      | 2020-03-10 11:33:24 |
| dpi  |        | 2020-02-01 15:46:48 |
| foo  |        | 2020-03-10 11:32:12 |
+------+--------+---------------------+
```

As with *save* (page 13), any characters permitted in a filename are permitted in the new destination name.

If you wish to rename a configuration such that it overwrites an existing configuration you will need to use the *--force* (page 12) option:

```
$ sudo pibootctl load default
$ sudo pibootctl save foo
$ pibootctl ls
+---------+--------+---------------------+
| Name    | Active | Timestamp           |
|---------+--------+---------------------|
| 720p    |        | 2020-03-10 11:33:24 |
| default | x      | 2020-03-10 11:32:12 |
| dpi     |        | 2020-02-01 15:46:48 |
| foo     | x      | 2020-03-10 11:32:12 |
+---------+--------+---------------------+
$ sudo pibootctl mv foo default
[Errno 17] File exists: 'default.zip'
$ sudo pibootctl mv -f foo default
```

2.8 save

2.8.1 Synopsis

```
pibootctl save [-h] [-f] name
```

2.8.2 Description

Store the current boot configuration under a given name.

2.8.3 Options

name
: The name to save the current boot configuration under; can include any characters legal in a filename

-h, --help
: Show a brief help page for the command.

-f, --force
: Overwrite an existing configuration, if one exists

2.8.4 Usage

The **save** command is used to take a backup of the current boot configuration. In practice this creates a PKZIP[3] of the files that make up the boot configuration (`config.txt` et al.), and places it under the configured directory on the boot partition (usually `pibootctl`):

```
$ ls /boot/pibootctl
$ sudo pibootctl save foo
$ ls /boot/pibootctl
foo.zip
```

[3] https://en.wikipedia.org/wiki/Zip_(file_format)

Note that by default, you cannot overwrite saved configurations, but this can be overridden with the *--force* (page 13) option:

```
$ sudo pibootctl save foo
[Errno 17] File exists: 'foo.zip'
$ sudo pibootctl save -f foo
```

In the event that your system is rendered un-bootable, a boot configuration can be easily restored by extracting the PKZIP of a saved configuration into the boot partition (over-writing files as necessary). Alternatively you can use the *load* (page 10) command (if the system can boot). The *list* (page 9) command can be used to display all currently stored configurations.

2.9 set

2.9.1 Synopsis

```
pibootctl set [-h] [--no-backup] [--all | --this-model | --this-serial]
              [--json] [--yaml] [--shell]
              [name=[value] [name=[value] ...]]
```

2.9.2 Description

Change the value of one or more boot configuration settings. To reset the value of a setting to its default, simply omit the new value.

2.9.3 Options

name=[value]
: Specify one or more settings to change on the command line; to reset a setting to its default omit the value.

-h, --help
: Show a brief help page for the command.

--no-backup
: Don't take an automatic backup of the current boot configuration if one doesn't exist.

--all
: Set the specified settings on all Pis this SD card is used with. This is the default context.

--this-model
: Set the specified settings for this model of Pi only.

--this-serial
: Set the specified settings for this Pi's serial number only.

--json
: Use JSON as the input format.

--yaml
: Use YAML as the input format.

--shell
: Use a var=value input format suitable for the shell.

2.9.4 Usage

The **set** command can be used at the command line to update the boot configuration:

```
$ sudo pibootctl set video.overscan.enabled=off
Backed up current configuration in backup-20200309-230959
```

Note that, if no backup of the current boot configuration exists, a backup is automatically taken (unless *--no-backup* (page 14) is specified). Multiple settings can be changed at once, and settings can be reset to their default value by omitting the new value after the "=" sign:

```
$ sudo pibootctl set --no-backup serial.enabled=on serial.uart=
```

By default, settings are written into an "[all]" section in **config.txt** meaning that they will apply everywhere the SD card is moved. However, you can opt to make settings specific to the current model of Pi, or even the current Pi's serial number:

```
$ sudo pibootctl set --this-serial camera.enabled=on gpu.mem=128
```

In this case an appropriate section like "[0x123456789]" will be added and the settings written under there.

For those wishing to build an interface on top of pibootctl, JSON, YAML, and shell-friendly formats can also be used to feed new values to the **set** command:

```
$ cat << EOF | sudo pibootctl set --json --no-backup
{"serial.enabled": true, "serial.uart": null}
EOF
```

2.10 show

2.10.1 Synopsis

```
pibootctl show [-h] [-a] [--json | --yaml | --shell] name [pattern]
```

2.10.2 Description

Display the specified stored boot configuration, or the sub-set of its settings that match the specified pattern.

2.10.3 Options

name
: The name of the boot configuration to display.

pattern
: If specified, only displays settings with names that match the specified pattern which may include shell globbing characters (e.g. *, ?, and simple [classes])

-h, --help
: Show a brief help page for the command.

-a, --all
: Include all settings, regardless of modification, in the output; by default, only settings which have been modified are included.

--json
Use JSON as the output format.

--yaml
Use YAML as the output format.

--shell
Use a var=value output format suitable for the shell.

2.10.4 Usage

The **show** command is the equivalent of the *status* (page 17) command for stored boot configurations. By default it displays only the settings in the specified configuration that have been modified from their default:

```
$ pibootctl show 720p
+------------------------+-----------------+
| Name                   | Value           |
|------------------------+-----------------|
| video.hdmi0.group      | 1 (CEA)         |
| video.hdmi0.mode       | 4 (720p @60Hz)  |
+------------------------+-----------------+
```

The full set of settings can be displayed (which is usually several pages long, and thus will implicitly invoke the system's pager) can be displayed with the *--all* (page 15) option:

```
$ pibootctl show 720p --all
+-----------------------------+----------+---------------------------------+
| Name                        | Modified | Value                           |
|-----------------------------+----------+---------------------------------|
...
| video.hdmi0.enabled         |          | auto                            |
| video.hdmi0.encoding        |          | 0 (auto; 1 for CEA, 2 for DMT)  |
| video.hdmi0.flip            |          | 0 (none)                        |
| video.hdmi0.group           | x        | 1 (CEA)                         |
| video.hdmi0.mode            | x        | 4 (720p @60Hz)                  |
| video.hdmi0.mode.force      |          | off                             |
| video.hdmi0.rotate          |          | 0                               |
| video.hdmi0.timings         |          | []                              |
| video.hdmi1.audio           |          | auto                            |
| video.hdmi1.boost           |          | 5                               |
...
```

Note that when *--all* (page 15) is specified, a "Modified" column is included in the output to indicate which settings are no longer default.

As with the *status* (page 17) command, the list of settings can be further filtered by specified a *pattern* with the command. The *pattern* can include any of the common shell wildcard characters:

- `*` for any number of any character
- `?` for any single character
- `[seq]` for any character in *seq*
- `[!seq]` for any character not in *seq*

For example:

```
$ pibootctl show --all 720p i2c.*
+-------------+----------+--------+
```

(continues on next page)

(continued from previous page)

```
| Name         | Modified | Value   |
|--------------+----------+---------|
| i2c.baud     |          | 100000  |
| i2c.enabled  |          | off     |
+--------------+----------+---------+
```

For developers wishing to build on top of pibootctl, options are provided to produce the output in JSON (*--json* (page 15)), YAML (*--yaml* (page 16)), and shell-friendly (*--shell* (page 16)). These combine with all aforementioned options as expected:

```
$ pibootctl show --json --all 720p i2c.*
{"i2c.baud": 100000, "i2c.enabled": false}
```

2.11 status

2.11.1 Synopsis

```
pibootctl status [-h] [-a] [--json | --yaml | --shell] [pattern]
```

2.11.2 Description

Output the current value of modified boot time settings that match the specified pattern (or all if no pattern is provided). The *--all* (page 17) option may be specified to output all boot settings regardless of modification state.

2.11.3 Options

pattern
: If specified, only displays settings with names that match the specified *pattern* which may include shell globbing characters (e.g. *, ?, and simple [classes]).

-h, --help
: Show a brief help page for the command.

-a, --all
: Include all settings, regardless of modification, in the output. By default, only settings which have been modified are included.

--json
: Use JSON as the output format.

--yaml
: Use YAML as the output format.

--shell
: Use a var=value format suitable for the shell.

2.11.4 Usage

By default, the **status** command only outputs boot time settings which have been modified:

```
$ pibootctl status
+-------------+-------+
| Name        | Value |
|-------------+-------|
| i2c.enabled | on    |
| spi.enabled | on    |
+-------------+-------+
```

The full set of settings (which is usually several pages long, and thus will implicitly invoke the system's pager) can be displayed with the *--all* (page 17) option:

```
$ pibootctl status --all
+-----------------------------+----------+--------------------------+
| Name                        | Modified | Value                    |
|-----------------------------+----------+--------------------------|
...
| i2c.baud                    |          | 100000                   |
| i2c.enabled                 | x        | on                       |
| i2s.enabled                 |          | off                      |
| serial.baud                 |          | 115200                   |
| serial.clock                |          | 48000000                 |
| serial.enabled              |          | on                       |
| serial.uart                 |          | 0 (/dev/ttyAMA0; PL011)  |
| spi.enabled                 | x        | on                       |
| video.cec.enabled           |          | on                       |
...
```

Note that when *--all* (page 17) is specified, a "Modified" column is included in the output to indicate which settings are no longer default.

The list of settings can be further filtered by specified a *pattern* with the command. The *pattern* can include any of the common shell wildcard characters:

- `*` for any number of any character
- `?` for any single character
- `[seq]` for any character in *seq*
- `[!seq]` for any character not in *seq*

For example:

```
$ pibootctl status --all i2c.*
+-------------+----------+--------+
| Name        | Modified | Value  |
|-------------+----------+--------|
| i2c.baud    |          | 100000 |
| i2c.enabled | x        | on     |
+-------------+----------+--------+
```

For developers wishing to build on top of pibootctl, options are provided to produce the output in JSON (*--json* (page 17)), YAML (*--yaml* (page 17)), and shell-friendly (*--shell* (page 17)). These combine with all aforementioned options as expected:

```
$ pibootctl status --json --all i2c.*
{"i2c.baud": 100000, "i2c.enabled": true}
```

CHAPTER 3

Development

If you wish to install a copy of pibootctl for development purposes, clone the git repository and set up a configuration to use the cloned directory as the source of the boot configuration:

```
$ sudo apt install python3-dev git virtualenvwrapper
$ cd
$ git clone https://github.com/waveform80/pibootctl.git
$ mkvirtualenv -p /usr/bin/python3 pibootctl
$ cd pibootctl
$ workon pibootctl
(pibootctl) $ make develop
(pibootctl) $ cat > ~/.config/pibootctl.conf << EOF
[defaults]
boot_path=.
store_path=store
reboot_required=
reboot_required_pkgs=
EOF
```

At this point you should be able to call the *pibootctl* (page 3) utility, and have it store the (empty) boot configuration as a PKZIP[4] file under the working directory:

```
$ pibootctl save foo
$ pibootctl ls
+------+--------+---------------------+
| Name | Active | Timestamp           |
|------+--------+---------------------|
| foo  | x      | 2020-03-08 22:40:28 |
+------+--------+---------------------+
```

To work on the clone in future simply enter the directory and use the **workon** command:

```
$ cd ~/pibootctl
$ workon pibootctl
```

To pull the latest changes from git into your clone and update your installation:

[4] https://en.wikipedia.org/wiki/Zip_(file_format)

```
$ cd ~/pibootctl
$ workon pibootctl
(pibootctl) $ git pull
(pibootctl) $ make develop
```

To remove your installation, destroy the sandbox and the clone:

```
(pibootctl) $ cd
(pibootctl) $ deactivate
$ rmvirtualenv pibootctl
$ rm -fr ~/pibootctl
```

3.1 Building the docs

If you wish to build the docs, you'll need a few more dependencies. Inkscape is used for conversion of SVGs to other formats, Graphviz and Gnuplot are used for rendering certain charts, and TeX Live is required for building PDF output. The following command should install all required dependencies:

```
$ sudo apt install texlive-xetex fonts-freefont-otf graphviz gnuplot inkscape
```

Once these are installed, you can use the "doc" target to build the documentation:

```
$ cd ~/pibootctl
$ workon pibootctl
(pibootctl) $ make doc
```

The HTML output is written to `build/html` while the PDF output goes to `build/latex`.

3.2 Test suite

If you wish to run the test suite, follow the instructions in *Development* (page 19) above and then make the "test" target within the sandbox:

```
$ cd ~/pibootctl
$ workon pibootctl
(pibootctl) $ make test
```

A tox[5] configuration is also provided that will test the utility against all supported Python versions:

```
$ cd ~/pibootctl
$ workon pibootctl
(pibootctl) $ pip install tox
...
(pibootctl) $ tox -p auto
```

Note: If developing under Ubuntu, the Dead Snakes PPA[6] is particularly useful for obtaining additional Python installations for testing.

[5] https://tox.readthedocs.io/en/latest/
[6] https://launchpad.net/~deadsnakes/+archive/ubuntu/ppa

CHAPTER 4

API

pibootctl (page 3) can be used both as a standalone application, and as an API within Python. The primary class of interest when using *pibootctl* (page 3) as an API is *Store* (page 38) in the *pibootctl.store* (page 38) module, but *pibootctl.main* (page 28) is useful for providing an instance of this, constructed from the pibootctl configuration.

The API is split into several modules, documented in the following sections:

4.1 pibootctl.exc

The *pibootctl.exc* (page 21) module defines the various exceptions used in the application:

exception pibootctl.exc.**InvalidConfiguration**(*errors*)
Error raised when an updated configuration fails to validate. All ValueError[7] exceptions raised during validation are available from the **errors** attribute which maps setting names to the ValueError[8] raised.

exception pibootctl.exc.**IneffectiveConfiguration**(*diff*)
Error raised when an updated configuration has been overridden by something in a file we're not allowed to edit. All settings which have been overridden are available from the **diff** attribute.

4.2 pibootctl.files

The *pibootctl.files* (page 21) module contains the *AtomicReplaceFile* (page 22) context manager, used to "safely" replace files by writing to a temporary file in the same directory, then moving the result over the target if no exception occurs within the block. The result is that external processes either see the "old" state of the file, or the "new" state, but nothing in between:

```
>>> from pathlib import Path
>>> from pibootctl.files import AtomicReplaceFile
>>> foo = Path('foo.txt')
>>> foo.write_text('foo')
```

(continues on next page)

[7] https://docs.python.org/3.5/library/exceptions.html#ValueError
[8] https://docs.python.org/3.5/library/exceptions.html#ValueError

(continued from previous page)

```
>>> foo.read_text()
'foo'
>>> with AtomicReplaceFile(foo, encoding='ascii') as f:
...     f.write('bar')
...     raise Exception('something went wrong!')
...
3
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
Exception: something went wrong!
>>> foo.read_text()
'foo'
```

class pibootctl.files.**AtomicReplaceFile**(*path*, *encoding=None*)

A context manager for atomically replacing a target file.

Uses tempfile.NamedTemporaryFile()[9] to construct a temporary file in the same directory as the target file. The associated file-like object is returned as the context manager's variable; you should write the content you wish to this object.

When the context manager exits, if no exception has occurred, the temporary file will be renamed over the target file atomically (and sensible permissions will be set, i.e. 0666 & umask). If an exception occurs during the context manager's block, the temporary file will be deleted leaving the original target file unaffected and the exception will be re-raised.

Parameters

- **path** (*str*[10] *or pathlib.Path*[11]) – The full path and filename of the target file. This is expected to be an absolute path.
- **encoding** (*str*[12]) – If None[13] (the default), the temporary file will be opened in binary mode. Otherwise, this specifies the encoding to use with text mode.

4.3 pibootctl.formatter

The *pibootctl.formatter* (page 22) module contains some generic text formatting routines, including the *TableWrapper* (page 22) class (akin to TextWrapper[14] but specific to table output), *TransMap* (page 25) for partially formatting templates, and the *render()* (page 27) function: a crude markup renderer.

class pibootctl.formatter.**TableWrapper**(*width=70*, *header_rows=1*, *footer_rows=0*, *cell_separator=' '*, *internal_line='-'*, *internal_separator=' '*, *borders=('', '', '', '')*, *corners=('', '', '', '')*, *internal_borders=('', '', '', '')*, *align=None*, *format=None*)

Similar to TextWrapper[15], this class provides facilities for wrapping text to a particular width, but with a focus on table-based output.

The constructor takes numerous arguments, but typically you don't need to specify them all (or at all). A series of dictionaries are provided with "common" configurations: *pretty_table* (page 24), *curvy_table* (page 24), *unicode_table* (page 25), and *curvy_unicode_table* (page 25). For example:

[9] https://docs.python.org/3.5/library/tempfile.html#tempfile.NamedTemporaryFile
[10] https://docs.python.org/3.5/library/stdtypes.html#str
[11] https://docs.python.org/3.5/library/pathlib.html#pathlib.Path
[12] https://docs.python.org/3.5/library/stdtypes.html#str
[13] https://docs.python.org/3.5/library/constants.html#None
[14] https://docs.python.org/3.5/library/textwrap.html#textwrap.TextWrapper
[15] https://docs.python.org/3.5/library/textwrap.html#textwrap.TextWrapper

```
>>> from pibootctl.formatter import *
>>> wrapper = TableWrapper(width=80, **curvy_table)
>>> data = [
... ('Name', 'Length', 'Position'),
... ('foo', 3, 1),
... ('bar', 3, 2),
... ('baz', 3, 3),
... ('quux', 4, 4)]
>>> print(wrapper.fill(data))
,------+--------+----------.
| Name | Length | Position |
|------+--------+----------|
| foo  | 3      | 1        |
| bar  | 3      | 2        |
| baz  | 3      | 3        |
| quux | 4      | 4        |
`------+--------+----------'
```

The *TableWrapper* (page 22) instance attributes (and keyword arguments to the constructor) are as follows:

width
(default 70) The maximum number of characters that the table can take up horizontally. *TableWrapper* (page 22) guarantees that no output line will be longer than *width* (page 23) characters.

header_rows
(default 1) The number of rows at the top of the table that will be separated from the following rows by a horizontal border (*internal_line* (page 23)).

footer_rows
(default 0) The number of rows at the bottom of the table that will be separated from the preceding rows by a horizontal border (*internal_line* (page 23)).

cell_separator
(default `' '`) The string used to separate columns of cells.

internal_line
(default `'-'`) The string used to draw horizontal lines inside the table for *header_rows* (page 23) and *footer_rows* (page 23).

internal_separator
(default `' '`) The string used within runs of *internal_line* (page 23) to separate columns.

borders
(default `('', '', '', '')`) A 4-tuple of strings which specify the characters used to create the left, top, right, and bottom borders of the table respectively.

corners
(default `('', '', '', '')`) A 4-tuple of strings which specify the characters used for the top-left, top-right, bottom-right, and bottom-left corners of the table respectively.

internal_borders
(default `('', '', '', '')`) A 4-tuple of strings which specify the characters used to interrupt runs of the *borders* (page 23) characters to draw row and column separators. Like *borders* (page 23) these are the left, top, right, and bottom characters respectively.

align
A callable accepting three parameters: 0-based row index, 0-based column index, and the cell data. The callable must return a character indicating the intended alignment of data within the cell. "<" for left justification, "^" for centered alignment, and ">" for right justification

(as in `str.format()`[16]). The default is to left align everything.

format
A callable accepting three parameters: 0-based row index, 0-based column index, and the cell data. The callable must return the desired string representation of the cell data. The default simply calls `str`[17] on everything.

TableWrapper (page 22) also provides similar public methods to `TextWrapper`[18]:

wrap(*data*)
Wraps the table *data* returning a list of output lines without final newlines. *data* must be a sequence of row tuples, each of which is assumed to be the same length.

If the current *width* (page 23) does not permit at least a single character per column (after taking account of the width of borders, internal separators, etc.) then `ValueError`[19] will be raised.

fill(*data*)
Wraps the table *data* returning a string containing the wrapped output.

pibootctl.formatter.**pretty_table**
Uses simple ASCII characters to produce a typical "box-like" table appearance:

```
>>> from pibootctl.formatter import *
>>> wrapper = TableWrapper(width=80, **pretty_table)
>>> data = [
... ('Name', 'Length', 'Position'),
... ('foo', 3, 1),
... ('bar', 3, 2),
... ('baz', 3, 3),
... ('quux', 4, 4)]
>>> print(wrapper.fill(data))
+------+--------+----------+
| Name | Length | Position |
|------+--------+----------|
| foo  | 3      | 1        |
| bar  | 3      | 2        |
| baz  | 3      | 3        |
| quux | 4      | 4        |
+------+--------+----------+
```

pibootctl.formatter.**curvy_table**
Uses simple ASCII characters to produce a "round-edged" table appearance:

```
>>> from pibootctl.formatter import *
>>> wrapper = TableWrapper(width=80, **curvy_table)
>>> data = [
... ('Name', 'Length', 'Position'),
... ('foo', 3, 1),
... ('bar', 3, 2),
... ('baz', 3, 3),
... ('quux', 4, 4)]
>>> print(wrapper.fill(data))
,------+--------+----------.
| Name | Length | Position |
|------+--------+----------|
| foo  | 3      | 1        |
```

(continues on next page)

[16] https://docs.python.org/3.5/library/stdtypes.html#str.format
[17] https://docs.python.org/3.5/library/stdtypes.html#str
[18] https://docs.python.org/3.5/library/textwrap.html#textwrap.TextWrapper
[19] https://docs.python.org/3.5/library/exceptions.html#ValueError

(continued from previous page)

```
| bar  | 3      | 2         |
| baz  | 3      | 3         |
| quux | 4      | 4         |
`------+--------+-----------'
```

pibootctl.formatter.**unicode_table**

Uses unicode box-drawing characters to produce a typical "box-like" table appearance:

```
>>> from pibootctl.formatter import *
>>> wrapper = TableWrapper(width=80, **unicode_table)
>>> data = [
... ('Name', 'Length', 'Position'),
... ('foo', 3, 1),
... ('bar', 3, 2),
... ('baz', 3, 3),
... ('quux', 4, 4)]
>>> print(wrapper.fill(data))

 Name   Length   Position

 foo    3        1
 bar    3        2
 baz    3        3
 quux   4        4
```

pibootctl.formatter.**curvy_unicode_table**

Uses unicode box-drawing characters to produce a "round-edged" table appearance:

```
>>> from pibootctl.formatter import *
>>> wrapper = TableWrapper(width=80, **curvy_unicode_table)
>>> data = [
... ('Name', 'Length', 'Position'),
... ('foo', 3, 1),
... ('bar', 3, 2),
... ('baz', 3, 3),
... ('quux', 4, 4)]
>>> print(wrapper.fill(data))

 Name   Length   Position

 foo    3        1
 bar    3        2
 baz    3        3
 quux   4        4
```

class pibootctl.formatter.**TransMap**(***kw*)

Used with `str.format_map()`[20] to substitute only a subset of values in a given template, passing the rest through for later processing. For example:

```
>>> '{foo}{bar}'.format_map(TransMap(foo=1))
'1{bar}'
>>> '{foo:02d}{bar:02d}{baz:02d}'.format_map(TransMap(foo=1, baz=3))
'01{bar:02d}03'
```

[20] https://docs.python.org/3.5/library/stdtypes.html#str.format_map

Note: One exception is that the `!a` conversion is not handled correctly. This is erroneously converted to `!r`. Unfortunately there's no solution to this; it's a side-effect of the means by which the `!a` conversion is performed.

`class pibootctl.formatter.FormatDict(`*data*, *key_title='Key'*, *value_title='Value'*, *sort_key=None*`)`

Used to format *data*, a `dict`[21], in a format acceptable as input to the *`render()`* (page 27) function. The *key_title* and *value_title* strings provide the cells for the single header row.

This class is intended to be used within a string for `str.format()`[22]. For example:

```
>>> from pibootctl.formatter import FormatDict
>>> d = {'foo': 100, 'bar': 200}
>>> print('An example table:\n\n{s}'.format(s=FormatDict(d)))
An example table:

| Key | Value |
| foo | 100 |
| bar | 200 |
```

The format specification in the format string can be used to request different kinds of output, for instance:

```
>>> f = FormatDict({'foo': 100, 'bar': 200})
>>> print('An example list:\n\n{f:list}'.format(f=f))
An example list:

* foo = 100
* bar = 200
>>> print('An example reference list:\n\n{f:refs}'.format(f=f))
An example reference list:

[foo]: 100
[bar]: 200
```

The default format specification is "table", naturally.

If the values are tuples that should be expanded into multiple columns, set *value_title* to a tuple with the corresponding column titles:

```
>>> from pibootctl.formatter import FormatDict
>>> d = {'foo': (1, 100), 'bar': (2, 200)}
>>> print('An example table:\n\n{s}'.format(s=FormatDict(d,
... value_title=('col1', 'col2'))))
An example table:

| Key | col1 | col2 |
| foo | 1 | 100 |
| bar | 2 | 200 |
```

Tuple values are only supported for table output.

Note: In Python versions before 3.7, you may need to use `collections.OrderedDict`[23] to ensure output of the elements of *data* in a particular order. Alternatively, you may specify a *sort_key*

[21] https://docs.python.org/3.5/library/stdtypes.html#dict
[22] https://docs.python.org/3.5/library/stdtypes.html#str.format
[23] https://docs.python.org/3.5/library/collections.html#collections.OrderedDict

value which will be applied to the key values of the dict to sort them prior to output.

pibootctl.formatter.**render**(*text*, *width=70*, *list_space=False*, *table_style=None*)
A crude renderer for a crude markup language intended for formatting documentation for the console.

The markup recognized by this routine is as follows:

```
* Paragraphs must be separated by at least one blank line. They will be
  wrapped to *width*.

* Items in bulleted lists must start with an asterisk. No list nesting
  is permitted, but items may span several lines (without blank lines
  between them). Items will be wrapped to *width* and indented
  appropriately.

* Lines beginning and ending with a pipe character are assumed to be
  table rows. Pipe characters also delimit columns within the row. The
  first row is assumed to be a header row and will be separated from
  the rest.

An example table is shown below:

| Command | Description |
| cd | changes the current directory |
| ls | lists the content of a directory |
| cp | copies files |
| mv | renames files |
| rm | removes files |
```

4.4 pibootctl.info

The *pibootctl.info* (page 27) module contains some simple routines for determining information about the Pi that the application is running on.

pibootctl.info.**get_board_revision**()
Return the Pi's board revision as an unsigned 32-bit integer number. This is the same number as reported under "Revision" in `/proc/cpuinfo`.

pibootctl.info.**get_board_serial**()
Return the Pi's serial number as an unsigned 64-bit integer number. This can also be queried as "Serial" under `/proc/cpuinfo`.

pibootctl.info.**get_board_type**()
Return a string indicating the overall model of the Pi, e.g. "pi0w", "pi2", or "pi3+". This is derived from the result of *get_board_revision()* (page 27) according to the Pi's revision codes table[24].

pibootctl.info.**get_board_types**()
Return a set of strings used for matching the model of Pi against configuration sections according to the conditional filters table[25].

pibootctl.info.**get_board_mem**()
Return the amount of memory (in megabytes) present on the Pi, according to the model returned by *get_board_revision()* (page 27).

[24] https://www.raspberrypi.org/documentation/hardware/raspberrypi/revision-codes/README.md
[25] https://www.raspberrypi.org/documentation/configuration/config-txt/conditional.md

4.5 pibootctl.main

The *pibootctl.main* (page 28) module defines the *Application* (page 28) class, and an instance of this called *main* (page 28). Instances of *Application* (page 28) are callable and thus *main* (page 28) is the entry-point for the *pibootctl* (page 3) script.

From an API perspective, this module is primarily useful for providing an instance of the *Store* (page 38) class:

```
from pibootctl.main import main
from pibootctl.store import Store, Current, Default

store = main.store
store[Current] = store['foo']
```

pibootctl.main.main
The instance of *Application* (page 28) which is the entry-point for the *pibootctl* (page 3) script.

class pibootctl.main.Application
An instance of this class (*main* (page 28)) is the entry point for the application. The instance is callable, accepting the command line arguments as its single (optional) argument. The arguments will be derived from sys.argv[26] if not provided:

```
>>> from pibootctl.main import main
>>> try:
...     main(['-h'])
... except SystemExit:
...     pass
usage:  [-h] [--version]
{help,?,status,dump,get,set,save,load,diff,show,cat,list,ls,remove,rm,rename,mv}
...
```

Warning: Calling *main* (page 28) will raise SystemExit[27] in several cases (usually when requesting help output). It will also replace the system exception hook (sys.excepthook()[28]).

This is intended and by design. If you wish to use *pibootctl* (page 3) as an API, you are better off investigating the *Store* (page 38) class, or treating *pibootctl* (page 3) as a self-contained script and calling it with subprocess[29].

backup_if_needed()
Tests whether the active boot configuration is also present in the store (by checking for the calculated hash). If it isn't, constructs a unique filename (backup-<timestamp>) and saves a copy of the active boot configuration under it.

do_diff()
Implementation of the *diff* (page 5) command.

do_get()
Implementation of the *get* (page 6) command.

do_help()
Implementation of the *help* (page 7) command.

do_list()
Implementation of the *list* (page 9) command.

[26] https://docs.python.org/3.5/library/sys.html#sys.argv
[27] https://docs.python.org/3.5/library/exceptions.html#SystemExit
[28] https://docs.python.org/3.5/library/sys.html#sys.excepthook
[29] https://docs.python.org/3.5/library/subprocess.html#module-subprocess

do_load()
Implementation of the *load* (page 10) command.

do_remove()
Implementation of the *remove* (page 11) command.

do_rename()
Implementation of the *rename* (page 12) command.

do_save()
Implementation of the *save* (page 13) command.

do_set()
Implementation of the *set* (page 14) command.

do_show()
Implementation of the *show* (page 15) command.

do_status()
Implementation of the *status* (page 17) command.

static invalid_config(*exc)
Generates the error message for unhandled *InvalidConfiguration* (page 21) exceptions. These are caused when a configuration fails to validate, and have an **errors** attribute listing all the exceptions that occurred during validation.

mark_reboot_required()
Writes the necessary files to indicate that the system requires a reboot.

static overridden_config(*exc)
Generates the error message for unhandled *IneffectiveConfiguration* (page 21) exceptions. These are caused when a boot configuration is split across multiple files; the application is permitted to modify a file before the final one, but a later file overrides a value the application has tried to set in the file it is permitted to modify.

static permission_error(*exc)
Generates the error message for unhandled PermissionError[30] exceptions. As these are very likely to be caused by non-root execution, this is customzied to warn about this in the event that the effective UID is not 0.

commands
A dictionary mapping command names to their sub-parser.

config
Returns the script's configuration as derived from the files in the three pre-defined locations (see *pibootctl* (page 3) for more information). Returns a Namespace[31] containing the parsed configuration.

parser
The parser for all the sub-commands that the script accepts. The parser's defaults are derived from the configuration obtained from *config* (page 29). Returns the newly constructed argument parser.

store
The *Store* (page 38) containing the current and stored boot configurations.

4.6 pibootctl.parser

The *pibootctl.parser* (page 29) module provides the *BootParser* (page 30) class for parsing the boot configuration of the Raspberry Pi.

[30] https://docs.python.org/3.5/library/exceptions.html#PermissionError
[31] https://docs.python.org/3.5/library/argparse.html#argparse.Namespace

The output of this class consists of derivatives of *BootLine* (page 30) (*BootSection* (page 31), *BootCommand* (page 31), etc.) and *BootFile* (page 31) instances.

class pibootctl.parser.**BootParser**(*path*)

Parser for the files used to configure the Raspberry Pi's bootloader.

The *path* specifies the container of all files that make up the configuration. It be one of:

- a str[32] or a Path[33] in which case the path specified must be a directory
- a ZipFile[34]
- a dict[35] mapping filenames to *BootFile* (page 31) instances; effectively the output of *files* (page 30) after parsing

add(*filename*, *encoding=None*, *errors=None*)

Adds the auxilliary *filename* under *path* (page 30) to the configuration. This is used to update the *hash* (page 30) and *files* (page 30) of the parsed configuration to include files which are referenced by the boot configuration but aren't themselves configuration files (e.g. EDID data, and the kernel cmdline.txt).

If specified, *encoding* and *errors* are as for open()[36]. If *encoding* is None[37], the data is assumed to be binary and the method will return the content of the file as a bytes[38] string. Otherwise, the content of the file is assumed to be text and will be returned as a list[39] of str[40].

parse(*filename='config.txt'*)

Parse the boot configuration on *path* (page 30). The optional *filename* specifies the "root" of the configuration, and defaults to `config.txt`.

If parsing is successful, this will update the *files* (page 30), *hash* (page 30), *timestamp* (page 30), and *config* (page 30) attributes.

config

The parsed configuration; a sequence of *BootLine* (page 30) instances (or derivatives of *BootLine* (page 30)), after *parse()* (page 30) has been successfully called.

files

The content of all parsed files; a mapping of filename to *BootFile* (page 31) objects.

hash

After *parse()* (page 30) is successfully called, this is the SHA1 hash of the complete configuration in parsed order (i.e. starting at "config.txt" and proceeding through all included files).

path

The path under which all configuration files can be found. This may be a Path[41] instance, or a ZipFile[42], or a dict[43].

timestamp

The latest modified timestamp on all files that were read as a result of calling *parse()* (page 30).

class pibootctl.parser.**BootLine**(*filename*, *linenum*, *conditions*, *comment=None*)

Represents a line in a boot configuration. This is effectively an abstract base class and should never appear in output itself. Provides four attributes:

[32] https://docs.python.org/3.5/library/stdtypes.html#str
[33] https://docs.python.org/3.5/library/pathlib.html#pathlib.Path
[34] https://docs.python.org/3.5/library/zipfile.html#zipfile.ZipFile
[35] https://docs.python.org/3.5/library/stdtypes.html#dict
[36] https://docs.python.org/3.5/library/functions.html#open
[37] https://docs.python.org/3.5/library/constants.html#None
[38] https://docs.python.org/3.5/library/functions.html#bytes
[39] https://docs.python.org/3.5/library/stdtypes.html#list
[40] https://docs.python.org/3.5/library/stdtypes.html#str
[41] https://docs.python.org/3.5/library/pathlib.html#pathlib.Path
[42] https://docs.python.org/3.5/library/zipfile.html#zipfile.ZipFile
[43] https://docs.python.org/3.5/library/stdtypes.html#dict

filename
A str[44] indicating the path (relative to the configuration's root) of the file containing the line.

linenum
The 1-based line number of the line.

conditions
A *BootConditions* (page 32) specifying the filters in effect for this configuration line.

comment
Any comment that appears after other content on the line, or None[45] if no comment was present

class pibootctl.parser.BootSection(*filename*, *linenum*, *conditions*, *section*, *comment=None*)
A derivative of *BootLine* (page 30) for `[conditional sections]` in a boot configuration. Adds a single attribute:

section
The criteria of the section (everything between the square brackets).

Note: The `conditions` for a *BootSection* (page 31) instance *includes* the filters defined by that section.

---

class pibootctl.parser.BootCommand(*filename*, *linenum*, *conditions*, *command*, *params*, *hdmi=None*, *comment=None*)
A derivative of *BootLine* (page 30) which represents a command in a boot configuration, e.g. "disable_overscan=1". Adds several attributes:

command
The title of the command; characters before the first "=" in the line.

params
The value of the command; characters after the first "=" in the line. As a special case, the "initramfs" command has two values and thus if *command* (page 31) is "initramfs" then this attribute will be a 2-tuple.

hdmi
The HDMI display that the command applies to. This is usually None[46] unless the command has an explicit hdmi suffix (":" separated after the *command* (page 31) title but before the "="), or the command appears in an [HDMI:1] section.

class pibootctl.parser.BootInclude(*filename*, *linenum*, *conditions*, *include*, *comment=None*)
A derivative of *BootLine* (page 30) representing an "include" command in a boot configuration. Adds a single attribute:

include
The name of the file to be included.

class pibootctl.parser.BootFile
Represents a file in a boot configuration.

filename
A str[47] representing the file's path relative to the boot configuration's container (whatever that may be: a path, a zip archive, etc.)

timestamp
A datetime[48] containing the last modification timestamp of the file.

[44] https://docs.python.org/3.5/library/stdtypes.html#str
[45] https://docs.python.org/3.5/library/constants.html#None
[46] https://docs.python.org/3.5/library/constants.html#None
[47] https://docs.python.org/3.5/library/stdtypes.html#str
[48] https://docs.python.org/3.5/library/datetime.html#datetime.datetime

> **Note:** This is rounded down to a 2-second precision as that is all that PKZIP[49] archives support.

content
A bytes[50] string containing the complete content of the file.

encoding
None[51] if the file is a binary file. Otherwise, specifies the name of the character encoding to be used when reading the file.

errors
None[52] if the file is a binary file. Otherwise, specifies the character replacement strategy to be used with erroneous characters encountered when reading the file.

`class pibootctl.parser.BootConditions`
Represents the set of conditional filters that apply to a given *BootLine* (page 30). The class implements methods necessary to compare instances as if they were sets.

For example:

```
>>> cond_all = BootConditions()
>>> cond_pi3 = BootConditions(pi='pi3')
>>> cond_pi3p = BootConditions(pi='pi3p')
>>> cond_serial = BootConditions(pi='pi3', serial=0x12345678)
>>> cond_all == cond_pi3
False
>>> cond_all >= cond_pi3
True
>>> cond_pi3 > cond_pi3p
True
>>> cond_serial < cond_pi3
True
>>> cond_serial < cond_pi3p
False
```

pi
The model of pi that the section applies to. See conditional filters[53] for details of valid values. This represents sections like `[pi3]`.

hdmi
The index of the HDMI port (0 or 1) that settings within this section will apply to, if no index-suffix is provided by the setting itself. This represents sections like `[HDMI:0]`.

edid
The EDID of the display that the section applies to. This represents sections like `[EDID=VSC-TD2220]`.

serial
The serial number of the Pi that settings within this section will apply to, stored as an int[54]. This represents sections like `[0x12345678]`.

gpio
The GPIO number and state that must be matched for settings in this section to apply, stored as a (gpio, state) tuple. This represents sections like `[gpio2=0]`.

[49] https://en.wikipedia.org/wiki/Zip_(file_format)
[50] https://docs.python.org/3.5/library/functions.html#bytes
[51] https://docs.python.org/3.5/library/constants.html#None
[52] https://docs.python.org/3.5/library/constants.html#None
[53] https://www.raspberrypi.org/documentation/configuration/config-txt/conditional.md
[54] https://docs.python.org/3.5/library/functions.html#int

none
If this is True[55] then a [none] section has been encountered and no settings apply.

suppress_count
This is a "suppression count" used to track sections within included files that are currently disabled (because the include occurred within a section that itself is disabled).

evaluate(*section*)
Calculates a new conditional state (based upon the current conditional state) from the specified *section* criteria. Returns a new *BootConditions* (page 32) instance.

generate(*context=None*)
Given *context*, a *BootConditions* (page 32) instance representing the currently active conditional sections, this method yields the conditional secitons required to set the conditions to this instance. If *context* is not specified, it defaults to conditions equivalent to [any], which is the default in the Pi bootloader.

For example:

```
>>> current = BootConditions(pi='pi2', gpio=(4, True))
>>> wanted = BootConditions()
>>> print('\n'.join(wanted.generate(current)))
[all]
>>> wanted = BootConditions(pi='pi4')
>>> print('\n'.join(wanted.generate(current)))
[all]
[pi4]
>>> current = BootConditions(pi='pi2')
>>> print('\n'.join(wanted.generate(current)))
[pi4]
>>> current = BootConditions(none=True)
>>> print('\n'.join(wanted.generate(current)))
[all]
[pi3]
```

Note: The yielded strings do *not* end with a line terminator.

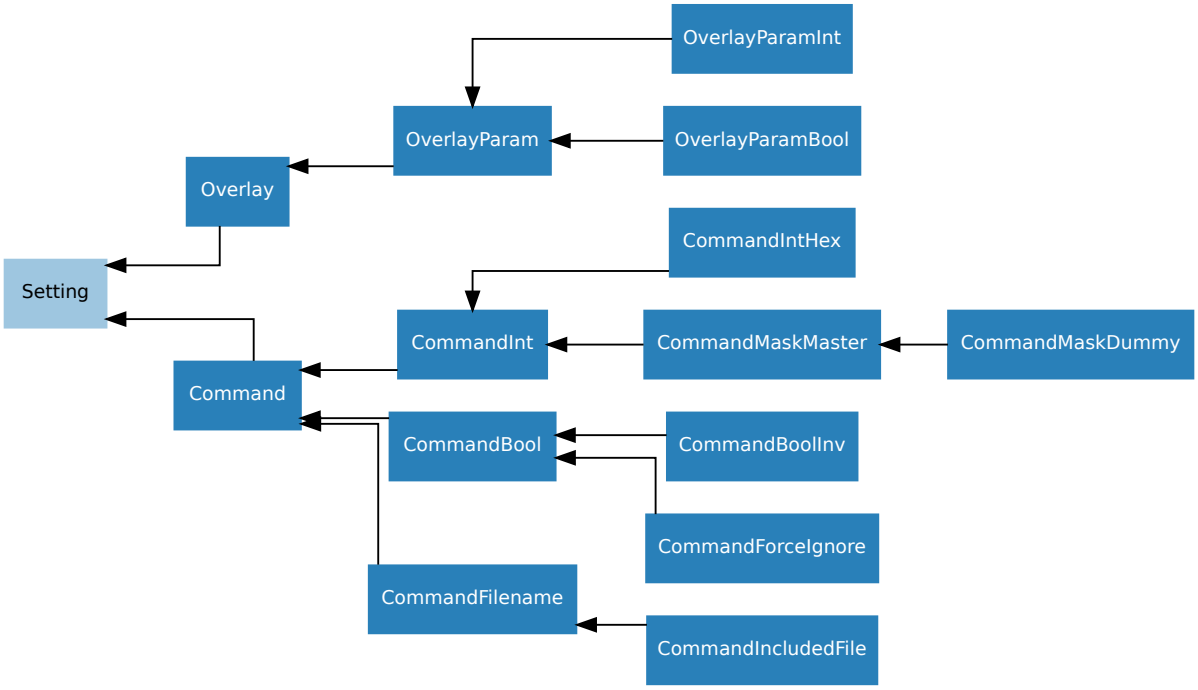
suppress()
If the current boot conditions are not *enabled* (page 33), returns a new *BootConditions* (page 32) instance with the suppression count incremented by one. This is used during parsing to disable all conditionals in suppressed includes.

enabled
Returns True[56] if parsed items are currently effective. If this is False[57], parsed items are ignored.

4.7 pibootctl.setting

The *pibootctl.setting* (page 33) module defines all the classes used to represent boot configuration settings:

[55] https://docs.python.org/3.5/library/constants.html#True
[56] https://docs.python.org/3.5/library/constants.html#True
[57] https://docs.python.org/3.5/library/constants.html#False



The base of the hierarchy is *Setting* (page 34) but this is effectively an abstract class and it is rare that anyone will need to use it directly. Rather you should derive from one of the concrete implementations below it like *OverlayParam* (page 36), *Command* (page 36), or one of the type-specializations like *CommandBool* (page 37), *CommandInt* (page 36), etc.

Note: For the sake of brevity, only the generic classes defined in *pibootctl.setting* (page 33) are documented here. There are also specialization classes specific to individual settings defined (for cases of complex inter-dependencies, e.g. how the Bluetooth enabled status affects the default serial UART).

Developers are advised to familiarize themselves with the full range of classes in this module before defining additional ones.

---

class pibootctl.setting.Setting(*name*, ***, *default=None*, *doc=''*)

Represents a configuration setting.

Each setting has a *name* which uniquely identifies the setting, a *default* value, and an optional *doc* string. The life-cycle of a typical setting in the scenario where the active boot configuration is being changed is:

- *extract()* (page 34) the value of a setting from parsed configuration lines
- *update()* (page 35) the value of a setting from user-provided values
- *validate()* (page 35) a setting in the wider context of a configuration
- generate *output()* (page 35) to represent the setting in a new config.txt

Optionally:

- *hint* (page 35) may be queried to describe a value in human-readable terms

extract(*config*)

Given a *config* which must be a sequence of *BootLine* (page 30) items, yields each line that potentially affects the setting's value (including those currently disabled by conditionals), and the new value that the line produces (or `None`[58] indicating that the value is now, or is still, the default state).

[58] https://docs.python.org/3.5/library/constants.html#None

Note: This method is *not* influenced by conditionals that disable a line. In this case the method must still yield the line and the value it would produce (were it enabled). The caller will deal with the fact the line is currently disabled (but needs to be aware of such lines for the configuration mutator).

For this reason (and others) this method must *not* affect *value* (page 36) directly; the caller will handle mutating the value when required.

output()
Yields lines of configuration to represent the current state of the setting (taking in account the context of other *Settings* (page 40)).

update(*value*)
Given a *value*, returns it transformed to the setting's native type (typically an int[59] or bool[60] but can be whatever type is appropriate).

The *value* may be a regular type (str[61], int[62], None[63], etc.) as deserialized from one of the input formats (JSON or YAML). Alternatively, it may be a *UserStr* (page 42), indicating that the value is a string given by the user on the command line and should be interpreted by the setting accordingly.

Note: Note to implementers: the method must *not* affect *value* (page 36) directly; the caller will handle this.

validate()
Validates the setting within the context of the other *Settings* (page 40). Raises ValueError[64] in the event that the current value is invalid. May optionally use *ValueWarning* (page 37) to warn about dangerous or inappropriate configurations.

default
The default value of this setting. The default may be sensitive to the wider context of *Settings* (page 40) (i.e. the default of one setting can change depending on the current value of other settings).

doc
A description of the setting, used as help-text on the command line.

hint
Provides a human-readable interpretation of the state of the setting. Used by the "dump" and "show" commands to provide translations of default and current values.

Must return None[65] if no explanation is available or necessary. Otherwise, must return a str[66].

key
Returns a tuple of strings which will be used to order the output of *output()* (page 35) in the generated configuration.

Note: The output of this property *must* be unique for each setting, unless a setting delegates all its output to another setting.

[59] https://docs.python.org/3.5/library/functions.html#int
[60] https://docs.python.org/3.5/library/functions.html#bool
[61] https://docs.python.org/3.5/library/stdtypes.html#str
[62] https://docs.python.org/3.5/library/functions.html#int
[63] https://docs.python.org/3.5/library/constants.html#None
[64] https://docs.python.org/3.5/library/exceptions.html#ValueError
[65] https://docs.python.org/3.5/library/constants.html#None
[66] https://docs.python.org/3.5/library/stdtypes.html#str

lines
Returns the *BootLine* (page 30) items which (if enabled by conditionals) affected the value of the setting, in the reverse order they were encountered while parsing (so the first *enabled* item holds the current value).

modified
Returns `True`[67] when the setting has been modified. Note that it is *not* sufficient to simply compare *value* (page 36) to *default* (page 35) as some defaults are context- or platform-specific.

name
The name of the setting. This is a dot-delimited list of strings; note that the individual components do not have to be valid identifiers. For example, "boot.kernel.64bit".

value
Returns the current value of the setting (or the *default* (page 35) if the setting has not been *modified* (page 36)).

class pibootctl.setting.Overlay(*name*, ***, *overlay*, *default=False*, *doc=''*)
Represents a boolean setting that is "on" if the represented *overlay* is present, and "off" otherwise.

overlay
The name of the overlay this parameter affects.

class pibootctl.setting.OverlayParam(*name*, ***, *overlay='base'*, *param*, *default=None*, *doc=''*)
Represents a *param* to a device-tree *overlay*. Like *Setting* (page 34), this is effectively an abstract base class to be derived from.

param
The name of the parameter within the base overlay that this setting represents.

class pibootctl.setting.OverlayParamInt(*name*, ***, *overlay='base'*, *param*, *default=0*, *doc=''*, *valid=None*)
Represents an integer parameter to a device-tree overlay.

The *valid* parameter may optionally provide a dictionary mapping valid integer values for the command to string explanations, to be provided by the basic *hint* (page 35) implementation.

class pibootctl.setting.OverlayParamBool(*name*, ***, *overlay='base'*, *param*, *default=False*, *doc=''*)
Represents a boolean parameter to the base device-tree overlay.

class pibootctl.setting.Command(*name*, ***, *command=None*, *commands=None*, *default=None*, *doc=''*, *index=None*)
Represents a string-valued configuration *command* or *commmands* (one of these must be specified, but not both). If multiple *commands* are represented, only the first will be generated by `output()` in this base class.

This is also the base class for most simple-valued configuration commands (integer, boolean, etc).

commands
The configuration commands that this setting represents.

index
The index of this setting for multi-valued settings (e.g. settings which apply to HDMI outputs).

class pibootctl.setting.CommandInt(*name*, ***, *command=None*, *commands=None*, *default=0*, *doc=''*, *index=0*, *valid=None*)
Represents an integer-valued configuration *command* or *commands*.

The *valid* parameter may optionally provide a dictionary mapping valid integer values for the command to string explanations, to be provided by the basic *hint* (page 35) implementation.

[67] https://docs.python.org/3.5/library/constants.html#True

class pibootctl.setting.CommandIntHex(*name*, ***, *command=None*, *commands=None*, *default=0*, *doc=''*, *index=0*, *valid=None*)
An integer-valued configuration *command* or *commands* that are typically represented in hexidecimal (like memory addresses).

class pibootctl.setting.CommandBool(*name*, ***, *command=None*, *commands=None*, *default=False*, *doc=''*, *index=0*)
Represents a boolean-valued configuration *command* or *commands*.

class pibootctl.setting.CommandBoolInv(*name*, ***, *command=None*, *commands=None*, *default=False*, *doc=''*, *index=0*)
Represents a boolean-valued configuration *command* or *commands* with inverted logic, e.g. video.overscan.enabled represents the `disable_overscan` setting and therefore its value is always the opposite of the actual written value.

class pibootctl.setting.CommandForceIgnore(*name*, ***, *force*, *ignore*, *doc=''*, *index=0*)
Represents the tri-valued configuration values with *force* and *ignore* commands, e.g. `hdmi_force_hotplug` and `hdmi_ignore_hotplug`.

For these cases, when both commands are "0" the setting is considered to have the value `None`[68] (which in most cases means "determine automatically"). When the *force* command is "1", the setting is `True`[69] and thus when the *ignore* command is "1", the setting is `False`[70]. When both are "1" (a contradictory setting) the final setting encountered takes precedence.

force
The boolean command that forces this setting on.

ignore
The boolean command that forces this setting off.

class pibootctl.setting.CommandMaskMaster(*name*, ***, *mask*, *command=None*, *commands=None*, *default=0*, *doc=''*, *index=0*, *valid=None*, *dummies=()*)
Represents an integer bit-mask setting as several settings. The "master" setting is the only one that produces any output. It defines the suffixes of its *dummies* (instances of *CommandMaskDummy* (page 37) which parse the same setting but produce no output of their own).

The *mask* specifies the integer bit-mask to be applied to the underlying value for this setting. The right-shift will be calculated from this. Single-bit masks will be represented as boolean values rather than integers.

class pibootctl.setting.CommandMaskDummy(*name*, ***, *mask*, *command=None*, *commands=None*, *default=0*, *doc=''*, *index=0*, *valid=None*, *dummies=()*)
Represents portions of integer bit-masks which are subordinate to a *CommandMaskMaster* (page 37) setting.

class pibootctl.setting.CommandFilename(*name*, ***, *command=None*, *commands=None*, *default=None*, *doc=''*, *index=None*)
Represents settings that contain a filename affected by the os_prefix command. The *filename* (page 37) returns the full filename incorporating the value of "boot.prefix" (if set), and *hint* (page 35) outputs a suitable message including the full path.

filename
The full filename represented by the value, after concatenating it with the value of "boot.prefix".

class pibootctl.setting.CommandIncludedFile(*name*, ***, *command=None*, *commands=None*, *default=None*, *doc=''*, *index=None*)
Represents settings that reference a file which should be included in any stored boot configuration.

[68] https://docs.python.org/3.5/library/constants.html#None
[69] https://docs.python.org/3.5/library/constants.html#True
[70] https://docs.python.org/3.5/library/constants.html#False

exception pibootctl.setting.ValueWarning
Warning class used by *Setting.validate()* (page 35) to warn about dangerous or inappropriate configurations.

4.8 pibootctl.settings

The *pibootctl.settings* (page 38) module defines the template of all settings stored by the *pibootctl.store.Settings* (page 40) class. Users of the API never have any need for this module, but developers wishing to extend the set of settings will need to modify the *SETTINGS* (page 38) set.

pibootctl.settings.SETTINGS
A dict[71] mapping setting names to *pibootctl.setting.Setting* (page 34) instances which represents the complete set of settings that the application handles.

4.9 pibootctl.store

The *pibootctl.store* (page 38) module defines classes which control a store of Raspberry Pi boot configurations, or the active boot configuration.

The main class of interest is *Store* (page 38). From an instance of this, one can access derivatives of *BootConfiguration* (page 39) for the purposes of manipulating the store of configurations, or the active boot configuration itself. Each *BootConfiguration* (page 39) contains an instance of *Settings* (page 40) which maps setting names to *Setting* (page 34) instances.

See *pibootctl.main* (page 28) for information on obtaining an instance of *Store* (page 38).

pibootctl.store.Current
The key of the active boot configuration in instances of *Store* (page 38).

pibootctl.store.Default
The key of the default (empty) boot configuration in instances of *Store* (page 38).

class pibootctl.store.Store(*boot_path*, *store_path*, *config_root='config.txt'*, *mutable_files=frozenset({'config.txt'})*, *comment_lines=False*)
A mapping representing all boot configurations (current, default, and stored).

Acts as a mapping keyed by the name of the stored configuration, or the special values *Current* (page 38) for the current boot configuration, or *Default* (page 38) for the default (empty) configuration. The values of the mapping are derivatives of *BootConfiguration* (page 39) which provide the parsed *Settings* (page 40), along with some other attributes.

The mapping is mutable and this can be used to manipulate stored boot configurations. For instance, to store the current boot configuration under the name "foo":

```
>>> store = Store('/boot', 'pibootctl')
>>> store["foo"] = store[Current]
```

Setting the item with the key *Current* (page 38) overwrites the current boot configuration:

```
>>> store[Current] = store["serial"]
```

Note that items retrieved from the store are effectively immutable; modifying them (even internally) does *not* modify the content of the store. To modify the content of the store, you must request a *mutable()* (page 39) copy of a configuration, modify it, and assign it back:

```
>>> foo = store["foo"].mutable()
>>> foo.update({"serial.enabled": True})
>>> store["serial"] = foo
```

[71] https://docs.python.org/3.5/library/stdtypes.html#dict

The same applies to the current boot configuration item:

```
>>> current = store[Current].mutable()
>>> current.update({"camera.enabled": True, "gpu.mem": 128})
>>> store[Current] = current
```

Items can be deleted to remove them from the store, with the obvious exception of the items with the keys *Current* (page 38) and *Default* (page 38) which cannot be removed (attempting to do so will raise a KeyError[72]). Furthermore, the item with the key *Default* (page 38) cannot be modified either.

Parameters

- **boot_path** (*str*[73]) – The path on which the boot partition is mounted.
- **store_path** (*str*[74]) – The path (relative to *boot_path*) under which stored configurations will be saved.
- **config_root** (*str*[75]) – The filename of the "root" of the configuration, i.e. the first file read by the parser, and the file in which certain commands (e.g. start_x) *must* be placed. Currently, this should always be "config.txt", the default.
- **mutable_files** (*set*[76]) – The set of filenames which *MutableConfiguration* (page 40) instances are permitted to change. By default this is just "config.txt".
- **comment_lines** (*bool*[77]) – If True[78], then *MutableConfiguration* (page 40) will comment out lines no longer required with a # prefix. When False[79] (the default), such lines will be deleted instead. When adding lines, regardless of this setting, the utility will search for, and uncomment, commented out lines which match the required output.

active
Returns the key of the active configuration, if any. If no configuration is currently active, returns None[80].

class pibootctl.store.**BootConfiguration**(*path*, *config_root='config.txt'*, *mutable_files=frozenset({'config.txt'})*, *comment_lines=False*)
Represents a boot configuration, as parsed from *config_root* (default "config.txt") on the boot partition (presumably mounted at *path*, a Path[81] instance).

mutable()
Return a *MutableConfiguration* (page 40) based on the parsed content of this configuration.

Note that mutable configurations are not backed by any files on disk, so nothing is actually re-written until the updated mutable configuration is assigned back to something in the *Store* (page 38).

config_root
The root file of the boot configuration. This is currently always "config.txt".

files
A mapping of filenames to *BootFile* (page 31) instances representing all the files that make up the boot configuration.

[72] https://docs.python.org/3.5/library/exceptions.html#KeyError
[73] https://docs.python.org/3.5/library/stdtypes.html#str
[74] https://docs.python.org/3.5/library/stdtypes.html#str
[75] https://docs.python.org/3.5/library/stdtypes.html#str
[76] https://docs.python.org/3.5/library/stdtypes.html#set
[77] https://docs.python.org/3.5/library/functions.html#bool
[78] https://docs.python.org/3.5/library/constants.html#True
[79] https://docs.python.org/3.5/library/constants.html#False
[80] https://docs.python.org/3.5/library/constants.html#None
[81] https://docs.python.org/3.5/library/pathlib.html#pathlib.Path

hash
The SHA1 hash that identifies the boot configuration. This is obtained by hashing the files of the boot configuration in parsing order.

path
The path (or archive or entity) containing all the files that make up the boot configuration.

settings
A *Settings* (page 40) instance containing all the settings extracted from the boot configuration.

timestamp
The last modified timestamp of the boot configuration, as a datetime[82].

class pibootctl.store.StoredConfiguration(*path*, *config_root='config.txt'*, *mutable_files=frozenset({'config.txt'})*, *comment_lines=False*)
Represents a boot configuration stored in a ZipFile[83] specified by *path*. The starting file of the configuration is given by *config_root*. All other parameters are as in *BootConfiguration* (page 39).

class pibootctl.store.MutableConfiguration(*path*, *config_root='config.txt'*, *mutable_files=frozenset({'config.txt'})*, *comment_lines=False*)
Represents a changeable boot configuration.

Do not construct instances of this class directly; they are typically constructed from a *base* *BootConfiguration* (page 39), by calling *mutable()* (page 39).

Mutable configurations can be changed with the *update()* (page 40) method which will also validate the new configuration, and check that the settings were not overridden by later files. No link is maintained between the original *BootConfiguration* (page 39) and the mutable copy. This implies that nothing is re-written on disk when the mutable configuration is updated. The resulting configuration must be assigned back to something in the *Store* (page 38) in order to re-write disk files.

update(*values*, *context*)
Given a mapping of setting names to new values, updates the values of the corresponding settings in this configuration. If a value is None[84], the setting is reset to its default value.

class pibootctl.store.Settings(*items=None*)
Represents all settings in a boot configuration; acts like an ordered mapping of names to *Setting* (page 34) objects.

copy()
Returns a distinct copy of the configuration that can be updated without affecting the original.

diff(*other*)
Returns a set of (self, other) setting tuples for all settings that differ between *self* and *other* (another *Settings* (page 40) instance). If a particular setting is missing from either side, its entry will be given as None[85].

filter(*pattern*)
Returns a copy of the configuration which only contains settings with names matching *pattern*, which may contain regular shell globbing patterns.

modified()
Returns a copy of the configuration which only contains modified settings.

[82] https://docs.python.org/3.5/library/datetime.html#datetime.datetime
[83] https://docs.python.org/3.5/library/zipfile.html#zipfile.ZipFile
[84] https://docs.python.org/3.5/library/constants.html#None
[85] https://docs.python.org/3.5/library/constants.html#None

4.10 pibootctl.term

The *pibootctl.term* (page 41) module contains various utilities for determining the type of terminal the script is running under (*term_is_dumb()* (page 41), *term_is_utf8()* (page 42), and *term_size()* (page 42)), for directing terminal output through the system's *pager()* (page 42), and for constructing an overall *ErrorHandler* (page 41) for the script.

class pibootctl.term.**ErrorHandler**

Global configurable application exception handler. For "basic" errors (I/O errors, keyboard interrupt, etc.) just the error message is printed as there's generally no need to confuse the user with a complete stack trace when it's just a missing file. Other exceptions, however, are logged with the usual full stack trace.

The configuration can be augmented with other exception classes that should be handled specially by treating the instance as a dictionary mapping exception classes to *ErrorAction* (page 41) tuples (or any 2-tuple, which will be converted to an *ErrorAction* (page 41)).

For example:

```
>>> from pibootctl.term import ErrorAction, ErrorHandler
>>> import sys
>>> sys.excepthook = ErrorHandler()
>>> sys.excepthook[KeyboardInterrupt]
(None, 1)
>>> sys.excepthook[SystemExit]
(None, <function ErrorHandler.exc_value at 0x7f6178915e18>)
>>> sys.excepthook[ValueError] = (sys.excepthook.exc_message, 3)
>>> sys.excepthook[Exception] = ("An error occurred", 1)
>>> raise ValueError("foo is not an integer")
foo is not an integer
```

Note the lack of a traceback in the output; if the example were a script it would also have exited with return code 3.

clear()

Remove all pre-defined error handlers.

static exc_message(*exc_type*, *exc_value*, *exc_tb*)

Extracts the message associated with the exception (by calling `str`[86] on the exception instance). The result is returned as a one-element list containing the message.

static exc_value(*exc_type*, *exc_value*, *exc_tb*)

Returns the first argument of the exception instance. In the case of `SystemExit`[87] this is the expected return code of the script.

static syntax_error(*exc_type*, *exc_value*, *exc_tb*)

Returns the message associated with the exception, and an additional line suggested the user try the `--help` option. This should be used in response to exceptions indicating the user made an error in their command line.

class pibootctl.term.**ErrorAction**(*message*, *exitcode*)

Named tuple dictating the action to take in response to an unhandled exception of the type it is associated with in *ErrorHandler* (page 41). The *message* is an iterable of lines to be output as critical error log messages, and *exitcode* is an integer to return as the exit code of the process.

Either of these can also be functions which will be called with the exception info (type, value, traceback) and will be expected to return an iterable of lines (for *message*) or an integer (for *exitcode*).

[86] https://docs.python.org/3.5/library/stdtypes.html#str
[87] https://docs.python.org/3.5/library/exceptions.html#SystemExit

pibootctl.term.**term_is_dumb**()
Returns True[88] if stdout is something other than a TTY (e.g. a file redirection or a pipe).

pibootctl.term.**term_is_utf8**()
Returns True[89] if the code-set of the current locale is 'UTF-8'.

pibootctl.term.**term_size**()
Returns the size of the console as a (rows, cols) tuple.

pibootctl.term.**pager**(*enable=None*)
Used as a context manager to redirect stdout to the system's pager utility ("pager", "less", or "more" are all attempted, in that order).

By default (when *enable* is None[90]), stdout will only be redirected if stdout is connected to a TTY. If *enable* is True[91] stdout will always be redirected, and likewise when *enable* is False[92] the function will do nothing.

For example, the following script should print "Hello, world!", piping the result through the system's pager:

```
from pibootctl.term import pager
with pager():
    print("Hello, world!")
```

4.11 pibootctl.userstr

The *pibootctl.userstr* (page 42) module provides the *UserStr* (page 42) class which represents unparsed user input on the command line.

The module also provides a variety of functions for converting input (either from JSON, YAML, or other structured formats, or from unparsed *UserStr* (page 42)) into common types (*to_bool()* (page 42), *to_int()* (page 42), *to_str()* (page 43), etc).

class pibootctl.userstr.**UserStr**
Type used to represent a value expressed as a string on the command line. In other words, any value bearing this type is a string representation of some other type (possibly str[93], int[94], None[95], etc.)

Primarily used by various conversion routines (*to_bool()* (page 42), *to_str()* (page 43), etc.) to determine whether a value is a string parsed from some serialization format (like JSON or YAML) which should be treated as a string literal.

Note: The blank *UserStr* (page 42) is special in that it *always* represents None[96] in conversions.

---

pibootctl.userstr.**to_bool**(*s*)
Converts the *UserStr* (page 42) (or other type) *s* to a bool[97]. Various "typical" string representations of true and false are accepted including "true", "yes", and "on", along with their counter-parts "false", "no", and "off". Literal None[98] passes through unchanged, and a blank *UserStr* (page 42) will convert to None[99].

[88] https://docs.python.org/3.5/library/constants.html#True
[89] https://docs.python.org/3.5/library/constants.html#True
[90] https://docs.python.org/3.5/library/constants.html#None
[91] https://docs.python.org/3.5/library/constants.html#True
[92] https://docs.python.org/3.5/library/constants.html#False
[93] https://docs.python.org/3.5/library/stdtypes.html#str
[94] https://docs.python.org/3.5/library/functions.html#int
[95] https://docs.python.org/3.5/library/constants.html#None
[96] https://docs.python.org/3.5/library/constants.html#None
[97] https://docs.python.org/3.5/library/functions.html#bool
[98] https://docs.python.org/3.5/library/constants.html#None
[99] https://docs.python.org/3.5/library/constants.html#None

pibootctl.userstr.to_int(*s*)
: Converts the *UserStr* (page 42) (or other type) *s* to a int[100]. As with all *UserStr* (page 42) conversions, blank string inputs are converted to None[101], and literal None[102] passes through unchanged. Otherwise, decimal integers and hexi-decimal integers prefixed with "0x" are accepted.

pibootctl.userstr.to_float(*s*)
: Converts the *UserStr* (page 42) (or other type) *s* to a float[103]. As with all *UserStr* (page 42) conversions, blank string inputs are converted to None[104], and literal None[105] passes through unchanged. Otherwise, typical floating point values (optionally prefixed with sign, optionally suffixed with an exponent) are accepted.

pibootctl.userstr.to_str(*s*)
: Converts the *UserStr* (page 42) (or other type) *s* to a str[106]. Blank *UserStr* (page 42) are converted to None[107], and literal None[108] passes through unchanged. Everything else is simply passed to the str[109] constructor.

pibootctl.userstr.to_list(*s*, *sep=', '*)
: Converts the *UserStr* (page 42) (or other type) *s* to a list[110] based on the separator character *sep* (which defaults to ","). Blank *UserStr* (page 42) are converted to None[111], and literal None[112] passes through unchanged. Everything else is passed to the list[113] constructor. This ensures that the result is always a unique reference.

[100] https://docs.python.org/3.5/library/functions.html#int
[101] https://docs.python.org/3.5/library/constants.html#None
[102] https://docs.python.org/3.5/library/constants.html#None
[103] https://docs.python.org/3.5/library/functions.html#float
[104] https://docs.python.org/3.5/library/constants.html#None
[105] https://docs.python.org/3.5/library/constants.html#None
[106] https://docs.python.org/3.5/library/stdtypes.html#str
[107] https://docs.python.org/3.5/library/constants.html#None
[108] https://docs.python.org/3.5/library/constants.html#None
[109] https://docs.python.org/3.5/library/stdtypes.html#str
[110] https://docs.python.org/3.5/library/stdtypes.html#list
[111] https://docs.python.org/3.5/library/constants.html#None
[112] https://docs.python.org/3.5/library/constants.html#None
[113] https://docs.python.org/3.5/library/stdtypes.html#list

CHAPTER 5

Changelog

5.1 Release 0.5 (2020-09-09)

- Rewrote the configuration setting code to always target `config.txt` as several settings don't work in included files (e.g. `start_x`).
- Added `comment_lines` configuration option to permit commenting out lines instead of deleting them
- Enhanced the configuration setting code to search for and uncomment existing lines in preference to writing new ones
- Added `--this-model` and `--this-serial` options to permit adding settings in new conditional sections

5.2 Release 0.4 (2020-03-31)

- Handle unrecognized commands correctly in the "help" command
- Implemented loading settings with the `--shell` style
- Improved help output for reference lists
- Fixed all legal stuff (added copyright headers where required, re-licensed to GPL 3+)

5.3 Release 0.3 (2020-03-27)

- Added full bash completion support

5.4 Release 0.2 (2020-03-26)

- The application now reports which lines overrode a setting when the "ineffective setting" error occurs
- Added the max_framebuffers setting, and detection for the vc4-*-v3d overlays

- Fixed restoring the default configuration in which config.txt doesn't exist (i.e. when config.txt should be deleted or blanked; the prior version simply left the old config.txt in place incorrectly)
- Various documentation fixes

5.5 Release 0.1.1 (2020-03-13)

- Fixed broken build on Bionic

5.6 Release 0.1 (2020-03-13)

- Initial release.
- Please note that as this is a pre-v1 release, API stability is not yet guaranteed.

CHAPTER 6

License

This file is part of pibootctl.

pibootctl is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

pibootctl is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with pibootctl. If not, see <https://www.gnu.org/licenses/>.

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W