nebula

Release 0.0.1

Aug 08, 2023

Contents:

Nebula is a utility library design to aid development with embedded platforms through infrastructure management and orchestration. Targeted at using systems from the desktop as a standard developer or through a CI system. The majority of the supported functionality is built in pure python, or relies on existing packages built in pure python, making cross-platform support possible.

There are two main interfaces for nebula:

- **Module**: For python projects or projects using python for testing or tasking, the module interface is designed to complement existing infrastructure to enable boot file (uboot, bitstream, kernel) deployment automation to development systems.
- **CLI**: Built on top of invoke, the command-line interface simplifies common tasks typically done by a developer to build and deploy boot files to a development platform

CHAPTER 1

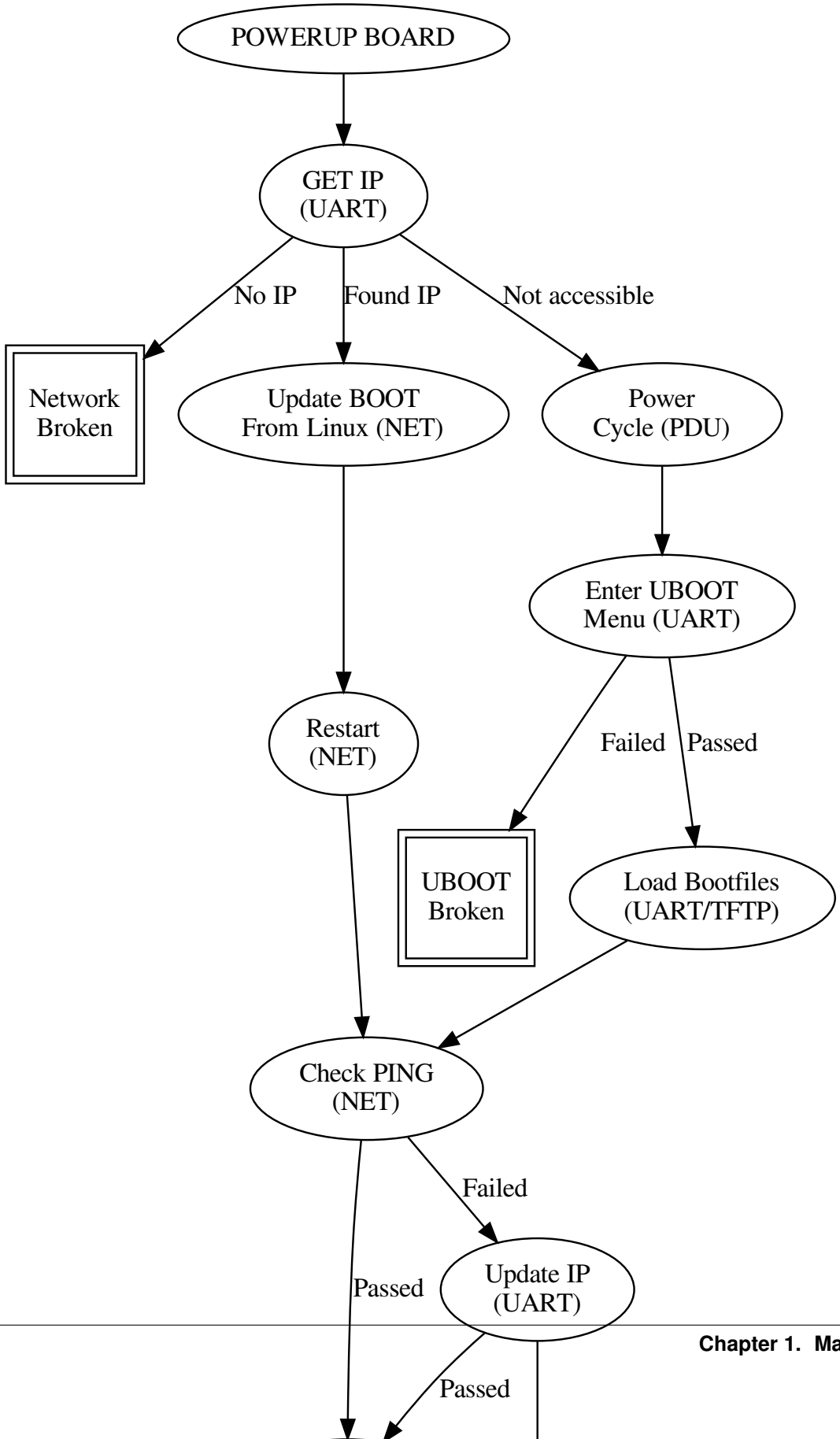
Managers

The manager class is designed to leverage many of the underlying classes (UART, Network, PDF, ...) together to effectively manage the state of development boards. Since in some cases not a single interface can be used to handle all failure modes, the manager class selectively uses the core classes to bring boards up and down, no matter their existing state or cause of failure.

1.1 Using Managers

1.2 Boot Flow

Below is the logic used to effectively load a boot files (bitstream, kernel, device tree) and test if the board is ready for driver specific or other tests that require a booted board.


POWERUP BOARD
GET IP (UART)
No IP
Found IP
Not accessible
Network Broken
Update BOOT From Linux (NET)
Power Cycle (PDU)
Enter UBOOT Menu (UART)
Restart (NET)
Failed
Passed
UBOOT Broken
Load Bootfiles (UART/TFTP)
Check PING (NET)
Failed
Passed
Update IP (UART)
Passed

CHAPTER 2

Command-Line

Invoke style tasks for Nebula's CLI.

2.1 Top-Level

```
Subcommands:

  gen-config                          Generate YAML configuration interactively
  show-log                            Show log for all following tasks
  update-config                       Update or read field of existing yaml
                                      config file
  build.repo                          Clone and build git project
  coverage.kernel                     Collect DUT gcov kernel logs and generate
                                      html report (Requires lcov to be
                                      installed locally)
  dl.bootfiles                        Download bootfiles for a specific
                                      development system
  dl.sdcard                           Download, verify, and decompress SD card
                                      image
  driver.check-iio-devices            Verify all IIO drivers appear on system
                                      as expected.
  info.supported-boards               Print out list of supported design names
  jtag.reboot                         Reboot board using JTAG
  manager.update-boot-files           Update boot files through u-boot menu
                                      (Assuming board is running)
  manager.update-boot-files-jtag      Update boot files through JTAG
                                      (Assuming board is running)
  manager.recovery-device-manager     Recover device through many methods
                                      (Assuming board is running)
  net.check-dmesg                     Download and parse remote board's dmesg
                                      log
  net.restart-board                   Reboot development system over IP
  net.update-boot-files               Update boot files on SD Card over SSH
```

(continues on next page)

(continued from previous page)

```
  pdu.power-cycle                      Reboot board with PDU
  uart.get-carriername                 Get Carrier (FPGA) name of DUT from UART
                                       connection
  uart.get-ip                          Get IP of DUT from UART connection
  uart.get-mezzanine                   Get Mezzanine (FMC) name of DUT from UART
                                       connection
  uart.restart-board                   Reboot DUT from UART connection assuming
                                       Linux is accessible
  uart.set-dhcp                        Set board to use DHCP for networking from
                                       UART connection
  uart.set-local-nic-ip-from-usbdev    Set IP of virtual NIC created from DUT
                                       based on found MAC
  uart.set-static-ip                   Set Static IP address of board of DUT
                                       from UART connection
  uart.update-boot-files               Update boot files through u-boot menu
                                       (Assuming board is running)
```

```
Usage: nebula [--core-opts] gen-config [other tasks here ...]

Docstring:
  Generate YAML configuration interactively

Options:
  none
```

```
Usage: nebula [--core-opts] show-log [--options] [other tasks here ...]

Docstring:
  Show log for all following tasks

Options:
  -l STRING, --level=STRING   Set log level. Default is DEBUG
```

```
Usage: nebula [--core-opts] update-config [--options] [other tasks here ...]

Docstring:
  Update or read field of existing yaml config file

Options:
  -b STRING, --board-name=STRING
  -f STRING, --field=STRING          Field of section of yaml to update
  -s STRING, --section=STRING        Section of yaml to update
  -v STRING, --value=STRING          New field value. If none if given field is
                                     only printed
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     OS_SPECIFIC
```

2.2 MANAGER

```
Usage: nebula [--core-opts] manager.update-boot-files [--options] [other tasks here ..
↪.]

Docstring:
  Update boot files through u-boot menu (Assuming board is running)

Options:
  -b STRING, --bootbinpath=STRING           Path to BOOT.BIN.
  -d STRING, --devtreepath=STRING           Path to devicetree.
  -f STRING, --folder=STRING                Resource folder containing
                                            BOOT.BIN, kernel, device tree, and
                                            system_top.bit. Overrides other
                                            setting
  -o STRING, --board-name=STRING
  -s STRING, --system-top-bit-path=STRING
  -u STRING, --uimagepath=STRING            Path to kernel image.
  -y STRING, --yamlfilename=STRING          Path to yaml config file. Default:
                                            /etc/default/nebula
```

```
Usage: nebula [--core-opts] manager.update-boot-files-jtag [--options] [other tasks␣
↪here ...]

Docstring:
  Update boot files through JTAG (Assuming board is running)

Options:
  -b STRING, --bootbinpath=STRING           Path to BOOT.BIN.
  -d STRING, --devtreepath=STRING           Path to devicetree.
  -f STRING, --folder=STRING                Resource folder containing BOOT.BIN,␣
↪kernel, device tree, and system_top.bit. Overrides other setting
  -o STRING, --board-name=STRING            Name of DUT design (Ex: zynq-zc706-
↪adv7511-fmcdaq2). Require for multi-device config files
  -s STRING, --system-top-bit-path=STRING   Path to system_top.bit
  -u STRING, --uimagepath=STRING            Path to kernel image.
  -y STRING, --yamlfilename=STRING          Path to yaml config file. Default: /etc/
↪default/nebula
```

```
Usage: nebula [--core-opts] manager.recovery-device-manager [--options] [other tasks␣
↪here ...]

Docstring:
  Recover device through many methods (Assuming board is running)

Options:
  -b STRING, --bootbinpath=STRING           Path to BOOT.BIN.
  -c         --sdcard                       No arguments required. If set, reference␣
↪files is obtained from SD card.
  -d STRING, --devtreepath=STRING           Path to devicetree.
  -f STRING, --folder=STRING                Resource folder containing BOOT.BIN,␣
↪kernel, device tree, and system_top.bit. Overrides other setting
  -o STRING, --board-name=STRING
  -s STRING, --system-top-bit-path=STRING   Path to system_top.bit
  -u STRING, --uimagepath=STRING            Path to kernel image.
```

(continues on next page)

(continued from previous page)

```
  -y STRING, --yamlfilename=STRING          Path to yaml config file. Default: /etc/
↪default/nebula
```

2.3 UART

```
Usage: nebula [--core-opts] uart.get-carriername [--options] [other tasks here ...]

Docstring:
  Get Carrier (FPGA) name of DUT from UART connection

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist, it will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.get-ip [--options] [other tasks here ...]

Docstring:
  Get IP of DUT from UART connection

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist is will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.get-mezzanine [--options] [other tasks here ...]

Docstring:
  Get Mezzanine (FMC) name of DUT from UART connection

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist, it will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.restart-board [--options] [other tasks here ...]

Docstring:
```

(continues on next page)

(continued from previous page)

```
  Reboot DUT from UART connection assuming Linux is accessible

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist is will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.set-dhcp [--options] [other tasks here ...]

Docstring:
  Set board to use DHCP for networking from UART connection

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist is will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -n STRING, --nic=STRING            Network interface name to set. Default is
                                     eth0
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.set-local-nic-ip-from-usbdev [--options] [other␣
↪tasks here ...]

Docstring:
  Set IP of virtual NIC created from DUT based on found MAC

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist it will override, if no
                                     yaml file exists and no address provided
                                     auto is used
  -b STRING, --board-name=STRING
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.set-static-ip [--options] [other tasks here ...]

Docstring:
  Set Static IP address of board of DUT from UART connection

Options:
  -a STRING, --address=STRING        UART device address (/dev/ttyACM0). If a
                                     yaml config exist it will override, if no
                                     yaml file exists and no address provided
                                     auto is used
```

(continues on next page)

(continued from previous page)

```
  -b STRING, --board-name=STRING
  -i STRING, --ip=STRING             IP Address to set NIC to
  -n STRING, --nic=STRING            Network interface name to set. Default is
                                     eth0
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

```
Usage: nebula [--core-opts] uart.update-boot-files [--options] [other tasks here ...]

Docstring:
  Update boot files through u-boot menu (Assuming board is running)

Options:
  -a STRING, --address=STRING                   UART device address
                                                (/dev/ttyACM0). If a yaml
                                                config exist is will override,
                                                if no yaml file exists and no
                                                address provided auto is used
  -b STRING, --board-name=STRING
  -d STRING, --devtreepath=STRING               Path to devicetree.
  -r, --reboot                                  Reboot board from linux console
                                                to get to u-boot menu. Defaut
                                                False
  -s STRING, --system-top-bit-filename=STRING
  -u STRING, --uimagepath=STRING                Path to kernel image.
  -y STRING, --yamlfilename=STRING              Path to yaml config file.
                                                Default: /etc/default/nebula
```

2.4 NETWORK

```
Usage: nebula [--core-opts] net.check-dmesg [--options] [other tasks here ...]

Docstring:
  Download and parse remote board's dmesg log
  Three log files will be produced:
      dmesg.log - Full dmesg
      dmesg_err.log - dmesg errors only
      dmesg_warn.log - dmesg warnings only

Options:
  -b STRING, --board-name=STRING
  -i STRING, --ip=STRING           IP address of board
  -p STRING, --password=STRING     Password for board. Default: analog
  -u STRING, --user=STRING         Board username. Default: root
```

```
Usage: nebula [--core-opts] net.restart-board [--options] [other tasks here ...]

Docstring:
  Reboot development system over IP
```

(continues on next page)

(continued from previous page)

```
Options:
  -b STRING, --board-name=STRING
  -i STRING, --ip=STRING            IP address of board
  -p STRING, --password=STRING      Password for board. Default: analog
  -u STRING, --user=STRING          Board username. Default: root
```

```
Usage: nebula [--core-opts] net.update-boot-files [--options] [other tasks here ...]

Docstring:
  Update boot files on SD Card over SSH

Options:
  -b STRING, --bootbinpath=STRING   Path to BOOT.BIN. Optional
  -d STRING, --devtreepath=STRING   Path to devicetree. Optional
  -i STRING, --ip=STRING            IP address of board. Default from yaml
  -m STRING, --uimagepath=STRING    Path to kernel image. Optional
  -o STRING, --board-name=STRING
  -p STRING, --password=STRING      Password for board. Default: analog
  -u STRING, --user=STRING          Board username. Default: root
```

2.5 DOWNLOADER

```
Usage: nebula [--core-opts] dl.bootfiles [--options] [other tasks here ...]

Docstring:
  Download bootfiles for a specific development system

Options:
  -a STRING, --board-name=STRING
  -b STRING, --branch=STRING         Name of branches to get related files. It can
                                     be from Linux+HDL folders or from the
                                     boot partition folder.
                                     For Linx+HDL, enterstring [<linuxbranch>,
↪<hdlbranch>].
                                     For boot partition, enter [boot_partition,
↪<bootpartitionbranch>].
                                     This is only used for http and artifactory␣
↪sources.
                                     Default is [boot_partition, master]
  -f, --firmware                     No arguments required. If set Pluto
                                     firmware is downloaded from GitHub. Branch
                                     name is used as release name. Design name
                                     must be pluto or m2k
  -o STRING, --source-root=STRING    Location of source boot files. Dependent on␣
↪source.
                                     For http and artifactory sources this is a IP or␣
↪domain name (no http://)
  -s STRING, --source=STRING         Boot file download source. Options are:
                                     local_fs, http, artifactory, remote.
                                     Default: local_fs
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
```

(continues on next page)

(continued from previous page)

```
                                    /etc/default/nebula
```

```
Usage: nebula [--core-opts] dl.sdcard [--options] [other tasks here ...]

Docstring:
  Download, verify, and decompress SD card image

Options:
  -r STRING, --release=STRING   Name of release to download. Default is 2019_R1
```

2.6 PDU

```
Usage: nebula [--core-opts] pdu.power-cycle [--options] [other tasks here ...]

Docstring:
  Reboot board with PDU

Options:
  -a STRING, --password=STRING       Password of PDU service (optional)
  -b STRING, --board-name=STRING
  -d STRING, --pduip=STRING          IP address of PDU (optional)
  -o STRING, --outlet=STRING         Outlet index of which dev board is
                                     connected
  -p STRING, --pdutype=STRING        Type of PDU used. Current options:
                                     cyberpower, vesync
  -u STRING, --username=STRING       Username of PDU service (optional)
  -y STRING, --yamlfilename=STRING   Path to yaml config file. Default:
                                     /etc/default/nebula
```

2.7 JTAG

```
Usage: nebula [--core-opts] jtag.reboot [--options] [other tasks here ...]

Docstring:
  Reboot board using JTAG


Options:
  -b STRING, --board_name=STRING
  -c STRING, --custom-vivado-path=STRING   Full path to vivado settings64 file.
                                           When set ignores vivado version
  -v STRING, --vivado-version=STRING       Set vivado version. Defauts to 2019.1
  -y STRING, --yamlfilename=STRING         Path to yaml config file. Default:
                                           /etc/default/nebula
```

2.8 INFO

```
Usage: nebula [--core-opts] info.supported-boards [--options] [other tasks here ...]

Docstring:
  Print out list of supported design names


Options:
  -f STRING, --filter=STRING   Required substring in design names
```

2.9 DRIVER

```
Usage: nebula [--core-opts] driver.check-iio-devices [--options] [other tasks here ...
↪]

Docstring:
  Verify all IIO drivers appear on system as expected.
  Exception is raised otherwise

Options:
  -b STRING, --board-name=STRING
  -i STRING, --iio-device-names=STRING   List of IIO driver names to check on board
  -u STRING, --uri=STRING                URI of board running iiod with drivers
                                         to check
  -y STRING, --yamlfilename=STRING       Path to yaml config file. Default:
                                         /etc/default/nebula
```

CHAPTER 3

pytest

Using nebula through a fixture

```
import nebula

@pytest.fixture(scope="session", autouse=True)
def load_boot_file(request):
    ### Before test

    # Bring up board
    print("Board bring up")
    cfg = request.config.getoption("--configfilename")
    m = nebula.manager(configfilename=cfg)
    m.start_tests()

    ############################
    yield
    ############################

    ### After test
    print("Board bring down")

    # Put board into good state
    m.stop_tests()
```

CHAPTER 4

Configuration

Main configuration is done through a main YAML file. If sections are not filled out they will be not set at run-time unless set on the command-line. Not setting certain parameters may limit functionality since some interfaces are required in certain board failure modes. Below is a complete example with documentation for each setting.

```
### Main external resources
system-config:
  - tftpserverip: 192.168.10.1 # TFTP server IP address
  - tftpserverroot: /tftpboot # Directory of TFTP share on server
### Local hardware settings
board-config:
  - board-name: zynq-zc706-adv7511-fmcdaq2
  - monitoring-interface: uart # Console monitoring type. Options: uart, netconsole
  - allow-jtag: False
### Downloader Settings
downloader-config:
  - reference_boot_folder: zynq-zc706-adv7511-fmcdaq2 # Location of bootfiles
  - hdl_folder: daq2_zc706
### Driver Settings
driver-config:
  - iio_device_names:
    - ad7291
    - ad9523-1
    - axi-ad9680-hpc
    - axi-ad9144-hpc
### UART settings
uart-config:
  - address: /dev/ttyUSB0 # File descriptor for UART console (auto will try to find,␣
↪but linux only)
  - baudrate: 115200 # UART baudrate in bits per second
  - logfilename: zc706-daq2.log # Filename for logging output of console
### Network/Ethernet settings
network-config:
  - dutip: 192.168.10.2 # IP address of development board
### Power distribution unit settings
```

(continues on next page)

(continued from previous page)

```
pdu-config:
  - pduip: 192.168.86.35 # IP address of power distribution unit
  - outlet: 1 # Outlet number of development board
  - pdu_type: cyberpower # PDU device type. Options: cyberpower, vesync
```

Each section of the yaml file applies to specific classes of nebula, and follow the convention <*classname*>**-config**, except for system. Therefore, you can modify any class property during initialization through the yaml file. For example if you wanted to change the *bootargs* setting, which is the kernel bootargs set over UART, you would have the following in your yaml:

```
uart-config:
  - bootargs: console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlycon rootfstype=ext4␣
↪rootwait
```

If settings exist in the yaml file within a **-config** block that does not has an existing property, this will cause an exception. This is designed to avoid defining settings which do not change behavior.

4.1 Generation

If you use the CLI interface through the **nebula gen-config** command to interactively generate this yaml file.

CHAPTER 5

Built-In Tests

Core tests are provided for basic validation of a given platform. Currently these are focused around libIIO, JESD, and some basic OS level checks.

5.1 Driver Tests

5.2 Operating System Tests

CHAPTER 6

Examples

6.1 Kernel Coverage Testing

This example will explain how gcov can be used to get coverage traces of specific drivers.

6.1.1 Building the kernel

The kernel must be built with certain configurations and enabled and certain Makefiles updated so the specific driver is selected for coverage monitoring. For complete doc look at the kernel.org doc .

In short, configure the kernel with:

```
CONFIG_DEBUG_FS=y
CONFIG_GCOV_KERNEL=y
```

select the gcc's gcov format, default is autodetect based on gcc version:

```
CONFIG_GCOV_FORMAT_AUTODETECT=y
```

To select specific drivers or folders update the necessary Makefiles as:

- For a single file (e.g. main.o):

  ```
  GCOV_PROFILE_main.o := y
  ```

- For all files in one directory:

  ```
  GCOV_PROFILE := y
  ```

To exclude files from being profiled even when CONFIG_GCOV_PROFILE_ALL is specified, use:

```
GCOV_PROFILE_main.o := n
```

and:

```
GCOV_PROFILE := n
```

Only files which are linked to the main kernel image or are compiled as kernel modules are supported by this mechanism.

6.1.2 Collecting logs and generating HTML reports

Once the remote kernel is booted and tests have been run. To collect the gcov traces and to generate the HTML report use the CLI API:

```
nebula coverage.kernel --ip <ip of DUT> --linux_build_dir <Build directory of kernel>
```

This will create a directory called **html** with the generated html report.

Note: For the reports to be generated correctly the necessary compiler must be on path which was used to build the kernel and lcov must be installed.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search