
paradrop Documentation

Release 0.11.2

Paradrop Labs

Apr 23, 2018

Contents

1	Cloud computing vs. edge computing	3
2	Where is the vantage point for edge computing?	5
3	How does it work?	7
4	System Architecture	9
4.1	ParaDrop router	9
4.2	ParaDrop cloud controller	10
4.3	ParaDrop API	10
5	Quick Start	11
5.1	Create a ParaDrop account	11
5.2	Boot the router	11
5.3	Activate a ParaDrop router	12
5.4	Install a hello-world chute	12
6	Hardware Support	13
6.1	Virtual Machine	13
6.2	PC Engines APU2	16
6.3	Intel NUC	18
7	Developing Applications	21
7.1	Introduction	21
7.2	Developing Light Chutes	23
7.3	Getting Started with Go	25
7.4	Getting Started with Java	26
7.5	Getting Started with Node.js	28
7.6	Getting Started with Python	30
7.7	Tutorial: Sticky Board	31
8	Frequently Asked Questions	37
8.1	Issues with the hardware or operating system	37
9	How to Contribute	39
9.1	ParaDrop daemon development	39
9.2	Documentation and tests	40

10 Host API Reference	43
10.1 Host Configuration	43
10.2 Chute Configuration	48
10.3 Chute Management	53
10.4 Device Configuration	59
10.5 Device Information	61
11 pdtools	65
11.1 chute	65
11.2 cloud	67
11.3 device	70
11.4 group	79
11.5 help	79
11.6 list-groups	79
11.7 node	80
11.8 routers	92
11.9 store	93
11.10 wizard	95
12 Source Code Reference	97
12.1 Subpackages	97
12.2 Submodules	165
12.3 paradow.main module	165
12.4 paradow.plan_demo module	165
12.5 Module contents	165
13 ParaDrop - Enabling Edge Computing at the Extreme Edge	167
14 Getting Started	169
15 Where to go from here?	171
HTTP Routing Table	173
Python Module Index	175

ParaDrop is a platform for edge computing. This is best understood by comparison with the popular paradigm of cloud computing.

Cloud computing vs. edge computing

Cloud computing platforms such as Amazon EC2, Microsoft Azure, and Google Cloud Platform have grown in popularity as solutions for providing ubiquitous access to services across different user devices. Cloud computing has benefits for infrastructure providers, service providers, and end users. Infrastructure providers, i.e., cloud platform providers, take advantage of the economies of scale by managing and operating resources in a centralized manner. Cloud computing also provides reliable, scalable, and elastic resources to service providers. In addition, end users can access high-performance computing and large storage resources anywhere with Internet access at any time thanks to the cloud computing.

Despite all of the benefits of cloud computing, there are some inherent trade-offs in the approach. Cloud computing requires developers to host services and data on off-site data centers. That means that the computing and storage resources are spatially distant from end-users and out of their control, which raises issues related to network latency, security, and privacy. A growing number of high-quality services can benefit from computational tasks running closer to end-users, especially within their own home or office. By moving the computation closer to the users, at the edge of the network, services can take advantage of the lower latency to provide better responsiveness and user experience as well as conserve network bandwidth.

Where is the vantage point for edge computing?

There are various options for placing edge computing nodes within the network. ParaDrop takes the approach of placing the edge computing substrate within the WiFi access points (APs). The WiFi AP is uniquely suitable for edge computing for multiple reasons:

- WiFi APs are ubiquitous in homes and businesses and inexpensive to replace.
- WiFi APs are always on and available.
- WiFi APs reside directly on the data path between Internet services and end users.

CHAPTER 3

How does it work?

ParaDrop is a research effort to build a highly programmable edge computing platform. The name for the project draws inspiration from the military use case of airdropping resources into the battlefield wherever they are needed most. Similarly, ParaDrop enables users and developers to *paradrop* edge services into the edge of the network as needed. Based on previous research work exploring the advantages of edge computing, we focus on building a platform that is friendly to both users and developers alike.

ParaDrop provides a similar runtime environment as the cloud computing platform to developers so that developers can easily port their services from the cloud to ParaDrop in part or in whole. It does this through lightweight containerization powered by Docker, which is already immensely popular in the cloud computing space. Containers allow developers the flexibility to build services with the programming languages, libraries, and frameworks they prefer, while being less resource-intensive than virtual machines. On top of that, ParaDrop offers a well-defined API that developers can leverage to implement and deploy interesting capabilities that are only available at the edge.

System Architecture

This section describes some of the important architectural features of ParaDrop. Our discussion will cover three major aspects of the ParaDrop design.

- The ParaDrop router
- The ParaDrop cloud controller
- The ParaDrop API

4.1 ParaDrop router

The ParaDrop router is the key part of the ParaDrop platform. In addition to being a WiFi access point, it provides the substrate on which edge computing services can be deployed. We have built the reference ParaDrop routers based on the [PC Engines APU](#) and APU2 single board computer. The image below shows a router built with a PC Engines APU board.



In addition to the PC Engines APU2, the ParaDrop software implementation can be run on various other hardware platforms as well as virtual machines. Please refer to *Hardware Support* for more information about hardware setup for ParaDrop routers.

4.2 ParaDrop cloud controller

The ParaDrop cloud controller is hosted at paradrop.org and provides a central location for tracking and managing ParaDrop routers as well as a Chute Store for software distribution. Users can sign up for a free account. For end users and administrators, it provides a dashboard to configure and monitor the ParaDrop routers under their control. The dashboard enables users to manage the services (called *chutes*) running on their routers. For developers, it provides the interface through which applications can be registered as ParaDrop chutes available for installation on routers.

4.3 ParaDrop API

ParaDrop exports the platform's full capability through an API. Based on the functionality and location, the API can be divided into two parts: the cloud API and the edge API. The cloud API provides the management interfaces for applications to orchestrate the chutes from the cloud. Examples include resource permission management, chute deployment and management, and router configuration management. The edge API exports the local context information of the routers to the chutes to do some useful things locally. Examples include local wireless channel information and local wireless peripheral device access.

This section goes through the steps to create a ParaDrop account, activate a ParaDrop router, and install a hello-world chute on the router.

If you have received a device with ParaDrop already installed, you can start here. If you do not have a ParaDrop-enabled device, please visit the *Hardware Support* section.

5.1 Create a ParaDrop account

With a ParaDrop account, users can manage the resources of ParaDrop through a web frontend.

1. Signup at <https://paradrop.org/signup>. You will receive a confirmation email from paradrop.org after you finish the signup.
2. Confirm your registration in the email.

5.2 Boot the router

1. Using an Ethernet cable, connect the WAN port of the ParaDrop router to a modem, switch, or other device with access to the Internet.
2. Connect the power supply. To avoid malfunctioning due to arcing, it is recommended to connect the barrel connector to DC jack on the back of the router first and connect the adapter to a power outlet second.
3. Allow the router 1-2 minutes to start up, especially on the first boot.
4. Connect a device (laptop, phone, etc.) either to one of the LAN ports on the back of the router or to its WiFi network. Typically, the router will be preconfigured with an open ESSID called “ParaDrop”. If the WiFi network has a password, that information will be provided separately.

5.3 Activate a ParaDrop router

Activation associates the router with your account on paradrop.org so that you can manage the router and chutes through the cloud controller.

1. Login to paradrop.org.
2. Navigate to the Routers List page. If your router came with a Claim Token, enter that here and skip steps 3-5. Otherwise if you do not have a Claim Token, click Create Router. Give your router a unique name and an optional description to help you remember it and click Submit.
3. On the router page, find the “Router ID” and “Password” fields. You will need to copy this information to the router so that it can connect to the controller.
4. Open the router portal at http://<router_ip_address> (or <http://paradrop.io> if you are connected to the LAN port of the router or its WiFi network). You may be prompted for a username and password. The default login is “paradrop” with an empty password.
5. Click the “Activate the router with a ParaDrop Router ID” button and enter the information from the paradrop.org router page. If the activation was successful, you should see checkmarks appear on the “WAMP Router” and “ParaDrop Server” lines. You may need to refresh the page to see the update.
6. After you activate your router, you will see the router status is online at <https://paradrop.org/routers>.

5.4 Install a hello-world chute

1. Make sure you have an activated, online router.
2. Go to the Chute Store tab on paradrop.org. There you will find some public chutes such as the hello-world chute. You can also create your own chutes here.
3. Click on the hello-world chute, click Install, click your router’s name to select it, and finally, click Install.
4. That will take you to the router page again where you can click the update item to monitor its progress. When the installation is complete, an entry will appear under the Chutes list.
5. The hello-world chute starts a webserver, which is accessible at <http://<router-ip-address>/chutes/hello-world>. Once the installation is complete, test it in a web browser.

This section describes various hardware platforms that we support for running ParaDrop.

If this is your first time working with ParaDrop and you do not have access to any of the supported hardware platforms, we recommend starting with our pre-built virtual machine image.

With the hardware platform up and running, you are ready to activate the router in ParaDrop. The page [Quick Start](#) gives detailed information about that.

6.1 Virtual Machine

This will quickly take you through the process of bringing up a Hello World chute in a virtual machine on your computer.

NOTE: These instructions assume you are running Ubuntu. The steps to launch a virtual machine may be different for other environments.

6.1.1 Environment setup

These steps will download our router image and launch it in a virtual machine.

1. Install required packages:

```
sudo apt-get install qemu-kvm
```

2. Download the latest build of the Paradrop disk image. <https://paradrop.org/release/latest/paradrop-amd64.img.xz>

3. Extract the image:

```
xz -d paradrop-amd64.img.xz
```

4. Launch the VM:

```
sudo kvm -m 1024 \  
-netdev user,id=net0,hostfwd=tcp::8000-:8000,hostfwd=tcp::8022-:22,  
↪hostfwd=tcp::8080-:80 \  
-device virtio-net-pci,netdev=net0 -drive file=paradrop-amd64.img,format=raw
```

6.1.2 First Boot Setup

After starting the virtual machine for the first time, follow the instructions on the screen. When it prompts for an email address, enter *info@paradrop.io*. This sets up a user account on the router called *paradrop* and prepares the router to receive software upgrades. Allow the router 1-2 minutes to complete its setup before proceeding.

Please note: there is no username/password to log into the system console. Please follow the steps in the next sections to access your router through paradrop.org or through SSH.

6.1.3 Connecting to your Router with SSH

The router is running an SSH server, which is forwarded from localhost port 8022 with the `kvm` command above. The router does not accept password login by default, so you will need to have an RSA key pair available, and you can use the router configuration page to upload the public key and authorize it.

1. Open tools page on the router (<http://localhost:8080/#!/tools>). The page might prompt you for a username and password. If so, use the username “paradrop” and an empty password.
2. Find the SSH Keys section and use the text area to submit your public key. Typically, your public key file will be found at `~/.ssh/id_rsa.pub`. You can use `ssh-keygen` to generate one if you do not already have one. Copy the text from the file, and make sure the format resembles the example before submitting.
3. After the key has been accepted by the router, you can login with the command `ssh -p 8022 paradrop@localhost`.

6.1.4 Managing Virtual Machines Using `virt-manager`

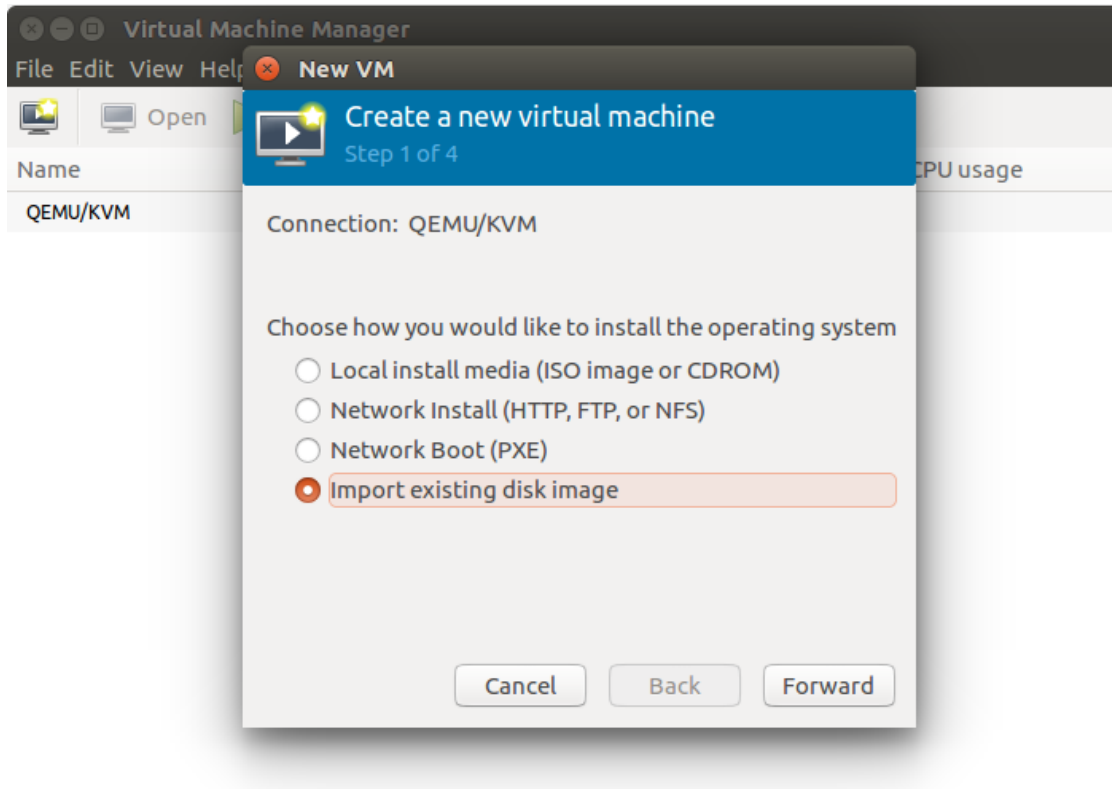
Even though many developers prefer command line tools to manage virtual machines, some developers like to use GUI tools. In addition, GUI tools are convenient to support some advanced features, e.g., assigning some peripheral devices (USB WiFi dongle) from host to virtual machines. We recommend using “`virt-manager`” to run ParaDrop virtual machines. If you have not installed it on Ubuntu, you can use below command to install it:

```
sudo apt-get install virt-manager
```

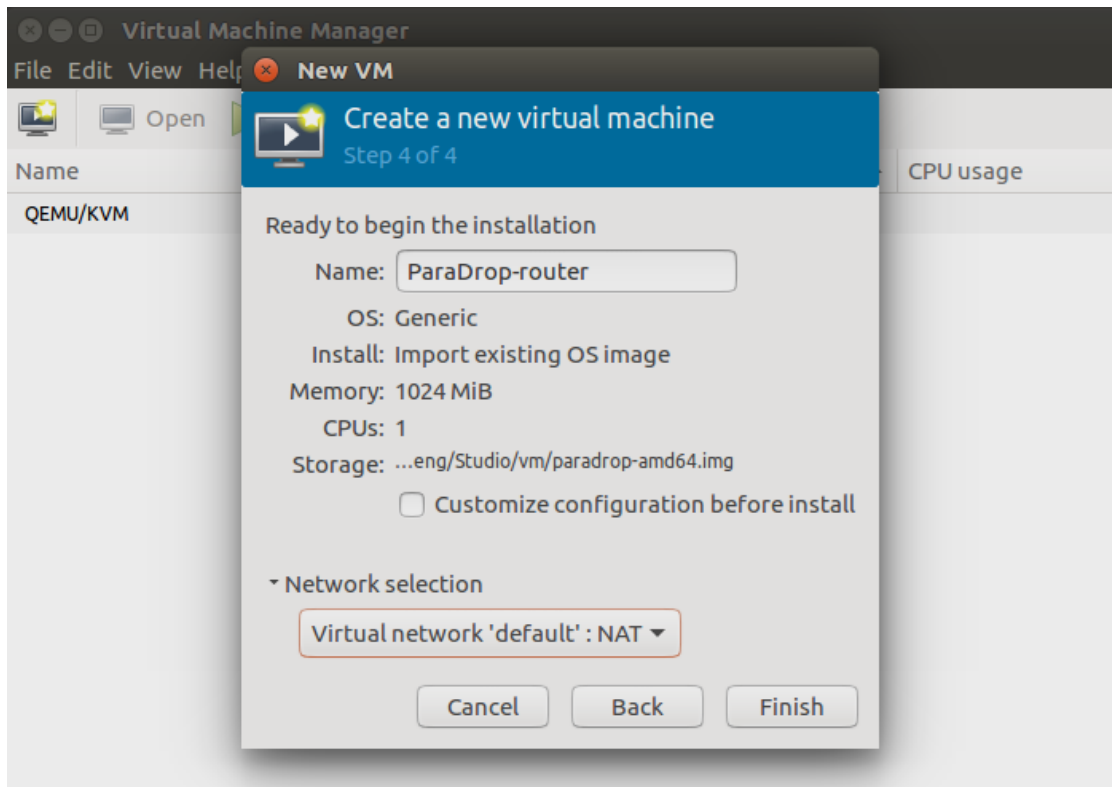
Then we can start `virt-manager` with below command:

```
sudo virt-manager
```

We can create a VM with the ParaDrop disk image.



Below is the configuration of the VM.



After that, we can boot the VM and configure the first boot as we do when run the VM with command line tools. However, the VM will have an IP address 192.168.122.x, so we can access <http://<IP address of the VM>> to access

the portal to upload ssh keys, and then login to it directly with the IP address.

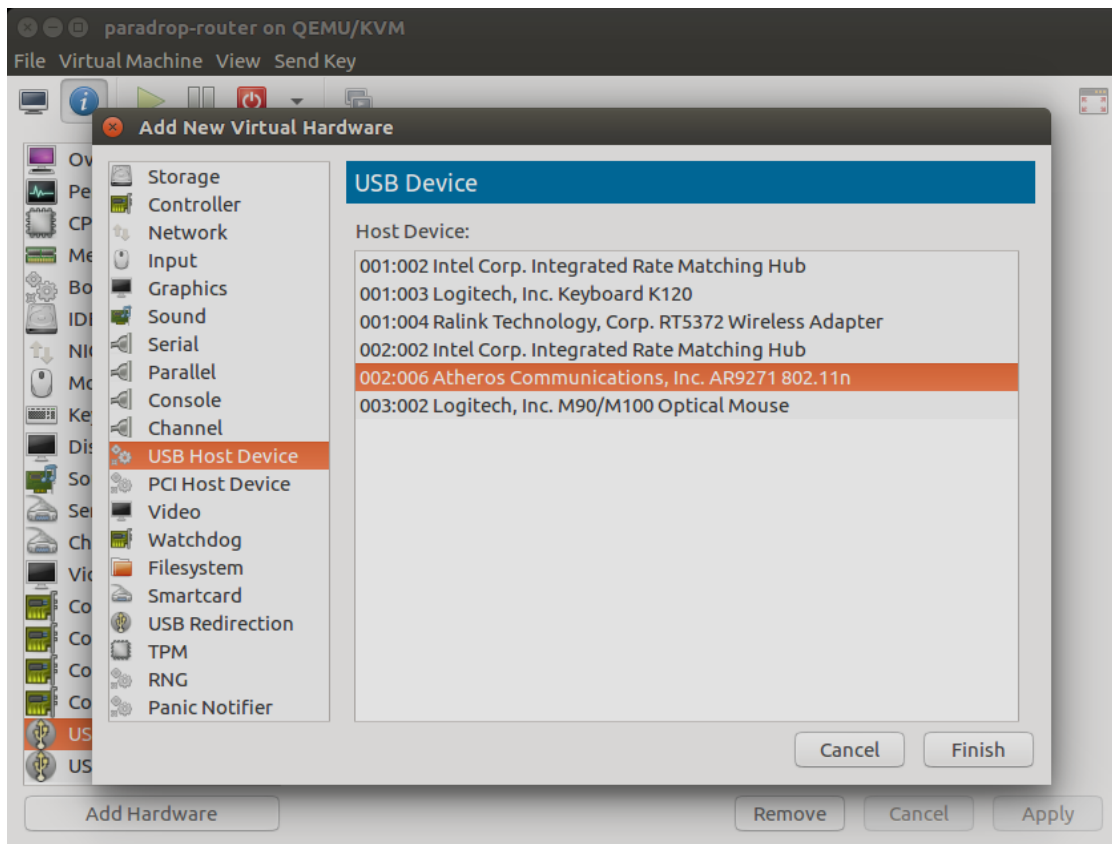
We can assign the USB WiFi dongle from the Host to the ParaDrop VM so that the ParaDrop running on the VM can support features related to WiFi. Before we do that, we need to disable the WiFi device for Host. We can do that with “rflist” command. Run below command to get the number of the WiFi device:

```
rflist list
```

Suppose the number of the WiFi device we want to assign to the ParaDrop VM is 2, then run below command to disable it for host OS:

```
rflist block 2
```

Then we can add the USB WiFi dongle to the VM.



We can run below command in ParaDrop VM to verify that the WiFi device has been detected:

```
iw dev
```

Sometimes, we have to repeat above steps to make sure the WiFi device can be used by the ParaDrop VM.

6.2 PC Engines APU2

6.2.1 Hardware requirements

- 1x system board (apu2c4)

- 1x case ([case1d2u](#))
- 1-2x miniPCIe Wi-Fi modules ([wle200nx](#) for 802.11n or [wle600vx](#) for 802.11ac)
- 2-4x pigtails ([pigsma](#))
- 2-4x antennas ([antsmadb](#))
- 1x power supply ([ac12vus2](#))
- 1x storage module ([sd4b](#)) or alternative (see below)

6.2.2 Storage Module

The APU can boot from an SD card or an m-SATA SSD. These instructions are written assuming you will use an SD card because they are easier to flash from another machine. However, we do frequently build Paradrop routers with SSDs to take advantage of the higher storage capacity and read/write speeds. The 4GB pSLC module listed above is known to be very reliable, but you may also prefer a larger SD card.

6.2.3 Preparing the SD card

1. Download the latest build of the Paradrop disk image. <https://paradrop.org/release/0.11/paradrop-amd64.img.xz>
2. Insert the SD card into the machine you used to download the image and find the device node for the card. This is often “/dev/sdb”, but please make sure, as the next command will overwrite the contents of whatever device you pass.
3. Copy the Paradrop image to the SD card.:

```
xzcat paradrop-amd64.img.xz | sudo dd of=<DEVICE> bs=32M status=progress; sync
```

4. Remove the SD card and proceed to assemble the router.

6.2.4 First Boot

If you know the IP address of the router, e.g. because you have access to the DHCP server upstream from the router, then you can skip this step and proceed with activating the router as described in the section [:doc:../manual/index_](#).

The first time you boot the Paradrop router, you can optionally connect a serial cable to complete the Ubuntu Core setup process. The default configuration is 9600 8N1. After the router boots, press Enter when prompted and follow the instructions on the console to configure Ubuntu Core. If you have an Ubuntu One account, you can enter the email address here. For consistency with the rest of the instructions, we recommend using the address *info@paradrop.io*. You will be able to manage your router and install chutes through paradrop.org either way.

Take note of the IP address displayed in the console. You will need this address for the next step, activating the router. For example, the message below indicates that the router has IP address 10.42.0.162.

```
Congratulations! This device is now registered to info@paradrop.io.

The next step is to log into the device via ssh:

ssh paradrop@10.42.0.162
```

6.3 Intel NUC

These instructions will help you install the ParaDrop daemon on the Intel NUC platform. At the end of this process, you will have a system ready for installing chutes.

We have specifically tested this process on the Skull Canyon (NUC6i7KYK) platform, which we recommend for high performance edge-computing needs.

6.3.1 Hardware and software requirements

- **Intel NUC Skull Canyon NUC6i7KYK**
 - The Intel NUC devices generally do not come with memory or storage pre-installed.
 - Memory: we recommend at least one 8 GB DDR4 SODIMM.
 - Storage: we have generally found one 16 GB SD card to be sufficient for our storage needs, but we recommend using one MX300 M.2 SSD card for the higher read and write speeds.
 - We recommend updating the BIOS on the NUC. Follow the instructions on [the Intel support site](#).
- 2 USB 2.0 or 3.0 flash drives (each 4 GB minimum)
- A monitor with an HDMI interface
- A network connection with Internet access
- An [Ubuntu Desktop 16.04.1 LTS](#) image.
- A [ParaDrop disk image](#).

6.3.2 Preparing for installation

1. Download the Ubuntu Desktop image and prepare a bootable USB flash drive.
2. Download the ParaDrop disk image and copy the file to the second flash drive.

6.3.3 Boot from the Live USB flash drive

1. Insert the Live USB Ubuntu Desktop flash drive in the NUC.
2. Start the NUC and push F10 to enter the boot menu.
3. Select the USB flash drive as a boot option.
4. Select “Try Ubuntu without installing”.

6.3.4 Flash ParaDrop

1. Once the system is ready, insert the second USB flash drive which contains the ParaDrop disk image.
2. Open a terminal and run the following command, where <disk label> is the name of the second USB flash drive. We recommend that you double-check that /dev/sda is the desired destination **before running dd**.

```
zcat /media/ubuntu/<disk label>/paradrop_router.img.gz | sudo dd of=/dev/sda_
↳bs=32M status=progress; sync
```

3. Reboot the system and remove all USB flash drives when prompted to do so.

6.3.5 First boot

1. At the Grub menu, press ‘e’ to edit the boot options.
2. Find the line that begins with “linux” and append the option “nomodeset”. It should look like “linux (loop)/kernel.img \$cmdline nomodeset”. Adding this option will temporarily fix a graphics issue that is known to occur with the Intel NUC.
3. Press F10 to continue booting.
4. Press Enter when prompted, and follow the instructions on the screen to configure Ubuntu Core. If you have an Ubuntu One account. By connecting your Ubuntu One account, you will be able to login via SSH with the key(s) attached to your account. Otherwise, if you do not have an Ubuntu One account or do not wish to use it, you may enter “info@paradrop.io” as your email address. You will still be able to manage your router and install chutes through paradrop.org either way.
5. Take note of the IP address displayed on the screen. You will need this address for the next step, activating the router. For example, the message below indicates that the router has IP address 10.42.0.162.

```
Congratulations! This device is now registered to info@paradrop.io.  
  
The next step is to log into the device via ssh:  
  
ssh paradrop@10.42.0.162  
...
```

Developing Applications

This section of the document is devoted to describing the edge computing services (chutes) that run on ParaDrop. There are two categories of ParaDrop applications - pure edge applications and cloud-edge hybrid applications. The pure edge applications have standalone chutes, which can be deployed in the ParaDrop routers. Cloud-edge hybrid applications have both a cloud component and an edge component. In this section, we will focus on the chute development, in other words, the edge component.

7.1 Introduction

ParaDrop is a software platform that enables services to run on Wi-Fi routers. We call these services *chutes* as in *parachutes*.

ParaDrop runs on top of [Ubuntu Core](#), a lightweight, transactionally updated operating system designed for deployments on embedded and IoT devices, cloud and more. It runs a new breed of secure, remotely upgradeable Linux app packages known as snaps. We support chute deployment through containerization powered by [Docker](#).

Minimally, a chute has a Dockerfile, which contains instructions for building and preparing the application to run on ParaDrop. A chute will usually also require scripts, binaries, configuration files, and other assets. For integration with the ParaDrop toolset, we highly recommend developing a chute as a [GitHub](#) project, but other organization methods are possible.

We will examine the [hello-world](#) chute as an example of a complete ParaDrop application.

7.1.1 Structure

Our hello-world chute is a git project with the following files:

```
chute/index.html
Dockerfile
README.md
```

The top-level contains a README and a special file called “Dockerfile”, which will be discussed below. As a convention, we place files that will be used by the running application in a subdirectory called “chute”. This is not necessary but helps keep the project organized. Valid alternatives include “src” or “app”.

7.1.2 Dockerfile

The Dockerfile contains instructions for building and preparing an application to run on ParaDrop. Here is a minimal Dockerfile for our hello-world chute:

```
FROM nginx
ADD chute/index.html /usr/share/nginx/html/index.html
```

FROM nginx

The FROM instruction specifies a base image for the chute. This could be a Linux distribution such as “ubuntu:14.04” or an standalone application such as “nginx”. The image name must match an image in the Docker public registry. We recommend choosing from the [official repositories](#). Here we use “nginx” for a light-weight web server.

ADD <source> <destination>

The ADD instruction copies a file or directory from the source repository to the chute filesystem. This is useful for installing scripts or other files required by the chute and are part of the source repository. The <source> path should be inside the repository, and the <destination> path should be an absolute path or a path inside the chute’s working directory. Here we install the index.html file from our source repository to the search directory used by nginx.

Other useful commands for building chutes are RUN and CMD. For a complete reference, please visit the [official Dockerfile reference](#).

Here is an alternative implementation of the hello-world Dockerfile that demonstrates some of the other useful instructions.

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y nginx
ADD chute/index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Here we use a RUN instruction to install nginx and a CMD instruction to set nginx as the command to run inside the chute container. Using “ubuntu:14.04” as the base image gives access to any packages that can be installed through apt-get.

7.1.3 Persistent Data

Each running chute has a persistent data storage that is not visible to other chutes. By default the persistent data directory is named “/data” inside the chute’s filesystem. Files stored in this directory will remain when upgrading or downgrading the chute and are only removed when uninstalling the chute.

7.1.4 System Information

The ParaDrop daemon share system information with chutes through a read-only directory named “/paradrop”. Chutes that are configured with a WiFi access point will find a file in this directory that lists wireless clients. In future versions there will also be information about Bluetooth and other wireless devices.

dnsmasq-wifi.leases

This file lists client devices that have connected to the chute's WiFi network and received a DHCP lease. This is a plain text file with one line for each device containing the following space-separated fields.

1. DHCP lease expiration time (seconds since Unix epoch).
2. MAC address.
3. IP address.
4. Host name, if known.
5. Client ID, if known; the format of this field varies between devices.

The following example shows two devices connected to the chute's WiFi network.

```
1480650200 00:11:22:33:44:55 192.168.128.130 android-ffeeddccbbaa9988 *
1480640500 00:22:44:66:88:aa 192.168.128.170 someones-iPod 01:00:22:44:66:88:aa
```

7.1.5 Chute-to-Host API

The Paradrop daemon exposes some functionality and configuration options to running chutes through an HTTP API. This aspect of Paradrop is under rapid development, and new features will be added with every release. The host API is available to chutes through the URL “<http://paradrop.io/api/v1>”. Paradrop automatically configures chutes to resolve “paradrop.io” to the ParaDrop device itself, so these requests go to the ParaDrop daemon running on the router and not to an outside server.

Authorization

In order to access the host API, chutes must pass a token with every request that proves the authenticity of the request. When chutes are installed on a ParaDrop router, they automatically receive a token through an environment variable named “PARADROP_API_TOKEN”. The chute should read this environment variable and pass the token as a Bearer token in an HTTP Authorization header. Here is an example in Python using the [Requests library](#):

```
import os
import requests

CHUTE_NAME = os.environ.get('PARADROP_CHUTE_NAME', 'chute')
API_TOKEN = os.environ.get('PARADROP_API_TOKEN', 'NA')

headers = { 'Authorization': 'Bearer ' + API_TOKEN }
url = 'http://paradrop.io/api/v1/chutes/{}/networks'.format(CHUTE_NAME)
res = requests.get(url, headers=headers)
print(res.json())
```

Please refer to *Host API Reference* for a complete listing of API functions.

7.2 Developing Light Chutes

Light chutes build and install the same way as normal chutes and can do many of the same things. However, they make use of prebuilt base images that are optimized for different programming languages.

Light chutes offer these advantages over normal chutes.

- **Safety:** Light chutes have stronger confinement properties, so you can feel safer installing a light chute written by a third party developer.
- **Fast installation:** Light chutes use a common base image that may already be cached on the router, so installation can be very fast.
- **Simplicity:** You do not need to learn how to write and debug a Dockerfile to develop a chute. Instead, you can use the package management tools you may already be using (e.g. `package.json` for npm and `requirements.txt` for pip).
- **Portability:** With ARM support coming soon for ParaDrop, your light chutes will most likely run on ARM with extra work on your part. This is not the case for normal chutes that use a custom Dockerfile.

We will look at the `node-hello-world` chute as an example of a light chute for ParaDrop.

7.2.1 Structure

Our hello-world chute is a git project with the following files:

```
README.md
index.js
package.json
paradrop.yaml
```

The project contains the typical files for a node.js project as well as a special file called “`paradrop.yaml`”.

7.2.2 `paradrop.yaml`

The `paradrop.yaml` file contains information that ParaDrop needs in order to run the chute. Here are the contents for the hello-world example:

```
version: 1
type: light
use: node
command: node index.js
```

All of these fields are required and very simple to use.

version: 1

This specifies the version of the `paradrop.yaml` schema in order to allow future changes without breaking existing chutes. You should specify version *1*.

type: light

This indicates that we are building a *light* chute.

use: node

This indicates that we are using the *node* base image for this chute. You should choose the base image appropriate for your project. Supported images are: *node* and *python2*.

command: node index.js

This line indicates the command for starting your application. You can either specified it this way, as a string with spaces between the parameters, or you can use a list of strings. The latter format would be particularly useful if your parameters include spaces. Here is an example:

```
command:
- node
- index.js
```

7.2.3 Persistent Data

Each running chute has a persistent data storage that is not visible to other chutes. By default the persistent data directory is named “/data” inside the chute’s filesystem. Files stored in this directory will remain when upgrading or downgrading the chute and are only removed when uninstalling the chute.

7.3 Getting Started with Go

This tutorial will teach you how to build a “Hello, World!” chute using Go.

7.3.1 Prerequisites

Make sure you have Go installed as well as ParaDrop pdtools (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

7.3.2 Set up

Make a new directory.

```
mkdir go-hello-world
cd go-hello-world
```

7.3.3 Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: go-hello-world
description: Hello World chute for ParaDrop using Go.
type: light
image: go
command: app
```

The end result should be a paradrop.yaml file similar to the following.

```
command: app
config: {}
description: Hello World chute for ParaDrop using Go.
name: go-hello-world
```

(continues on next page)

(continued from previous page)

```
type: light
use: go
version: 1
```

7.3.4 Develop the Application

Create a file name `main.go` with the following code.

```
package main

import (
    "fmt"
    "net/http"
)

func GetIndex(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World!\n")
}

func main() {
    fmt.Println("Listening on :8000")
    http.HandleFunc("/", GetIndex)
    http.ListenAndServe(":8000", nil)
}
```

Run the application locally with the following command.

```
go run main.go
```

Then load `http://localhost:8000/` in a web browser to see the result.

7.3.5 Wrap Up

The web server in this application listens on port 8000. We need to include that information in the `paradrop.yaml` file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 8000
```

7.4 Getting Started with Java

This tutorial will teach you how to build a “Hello, World!” chute using Java and Maven.

7.4.1 Prerequisites

Make sure you have Java 1.8+, Maven 3.0+, as well as ParaDrop `pdtools` (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

7.4.2 Set up

Use Maven to set up an empty project.

```
mvn archetype:generate -DgroupId=org.paradrop.app -DartifactId=java-hello-world -
↪DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
cd java-hello-world
```

7.4.3 Create a chute configuration

Use the pdtools interactive init command to create a paradrop.yaml file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: java-hello-world
description: Hello World chute for ParaDrop using Java.
type: light
image: maven
command: java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
```

The end result should be a paradrop.yaml file similar to the following.

```
command: java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
config: {}
description: Hello World chute for ParaDrop using Java.
name: java-hello-world
type: light
use: maven
version: 1
```

7.4.4 Develop the Application

Replace the automatically-generated application code in `src/main/java/org/paradrop/app/App.java` with the following code.

```
package org.paradrop.app;

import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Listening on :8000");
        HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
        server.createContext("/", new GetIndex());
        server.start();
    }
}
```

(continues on next page)

(continued from previous page)

```
}

static class GetIndex implements HttpHandler {
    @Override
    public void handle(HttpExchange t) throws IOException {
        String response = "Hello, World!";
        t.sendResponseHeaders(200, response.length());
        OutputStream os = t.getResponseBody();
        os.write(response.getBytes());
        os.close();
    }
}
```

Run the application locally with the following commands.

```
mvn package
java -cp target/java-hello-world-1.0-SNAPSHOT.jar org.paradrop.app.App
```

Then load `http://localhost:8000/` in a web browser to see the result.

7.4.5 Wrap Up

The web server in this application listens on port 8000. We need to include that information in the `paradrop.yaml` file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 8000
```

7.5 Getting Started with Node.js

This tutorial will teach you how to build a “Hello, World!” chute using Node.js and Express.

7.5.1 Prerequisites

Make sure you have Node.js (v6 or newer) installed as well as ParaDrop `pdtools` (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

7.5.2 Set up

Make a new directory.

```
mkdir node-hello-world
cd node-hello-world
```

7.5.3 Create a chute configuration

Use the `pdtools` interactive `init` command to create a `paradrop.yaml` file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of pdtools installed, the prompts may be slightly different.

```
name: node-hello-world
description: Hello World chute for ParaDrop using Node.js.
type: light
image: node
command: node index.js
```

The end result should be a paradrop.yaml file similar to the following.

```
command: node index.js
config: {}
description: Hello World chute for ParaDrop using Node.js.
name: node-hello-world
type: light
use: node
version: 1
```

The pdtools chute init command will also create a package.json file for you if one did not already exist, so there is no need to run npm init after running pdtools chute init.

7.5.4 Install Dependencies

Use the following command to install some dependencies. We will be using Express as a simple web server.

The --save option instructs npm to save the packages to the package.json file. When installing the chute, ParaDrop will read package.json to install the same versions of the packages that you used for development.:

```
npm install --save express@^4.16.1
```

7.5.5 Develop the Application

We indicated that index.js is the entrypoint for the application, so we will create a file named index.js and put our code there.

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello, World!')
})

app.listen(3000, function() {
  console.log('Listening on port 3000.')
})
```

Run the application locally with the following command.

```
node index.js
```

Then load `http://localhost:3000/` in a web browser to see the result.

7.5.6 Wrap Up

The web server in this application listens on port 3000. We need to include that information in the `paradrop.yaml` file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 3000
```

7.6 Getting Started with Python

This tutorial will teach you how to build a “Hello, World!” chute using Python and Flask.

7.6.1 Prerequisites

Make sure you have Python 2 installed as well as ParaDrop `pdtools` (v0.9.2 or newer).

```
pip install pdtools>=0.9.2
```

7.6.2 Set up

Make a new directory.

```
mkdir python-hello-world
cd python-hello-world
```

7.6.3 Create a chute configuration

Use the `pdtools` interactive `init` command to create a `paradrop.yaml` file for your chute.

```
python -m pdtools chute init
```

Use the following values as suggested responses to the prompts. If you have a different version of `pdtools` installed, the prompts may be slightly different.

```
name: python-hello-world
description: Hello World chute for ParaDrop using Python.
type: light
image: python2
command: python2 -u main.py
```

The end result should be a `paradrop.yaml` file similar to the following.

```
command: python2 -u main.py
config: {}
description: Hello World chute for ParaDrop using Python.
name: python-hello-world
type: light
use: python2
version: 1
```

7.6.4 Install Dependencies

We will use pip and virtualenv to manage dependencies for the project. First set up a virtual environment.

```
virtualenv venv
source venv/bin/activate
```

Use the following command to install some dependencies. We will be using Flask as a simple web server.

```
pip install Flask==0.12.2
```

Finally, save the version information to a file called `requirements.txt`. When installing the chute, ParaDrop will use this file to install the same versions of the packages that you used during development.

```
pip freeze >requirements.txt
```

7.6.5 Develop the Application

We indicated that `main.py` is the entrypoint for the application, so we will create a file named `main.py` and put our code there.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Run the application locally with the following command.

```
python main.py
```

Then load `http://localhost:5000/` in a web browser to see the result.

7.6.6 Wrap Up

The web server in this application listens on port 5000. We need to include that information in the `paradrop.yaml` file as well. Use the following command to alter the configuration file.

```
python -m pdtools chute set config.web.port 5000
```

7.7 Tutorial: Sticky Board

This tutorial will teach you how to build a fully-functional ParaDrop application from scratch. Through the tutorial, we will build a “Sticky Board”, a local board where visitors can post images for others to see. We will be using Node.js to build the application, so make sure you have that installed on your development machine.

7.7.1 Set Up

Make a new directory, and initialize a git repository:

```
mkdir sticky_board
cd sticky_board
git init
mkdir views
```

7.7.2 Setup Node.js Project

We will be using npm to manage Node.js packages. You can use the `npm init` command to get started or create a file called `package.json`. with the following contents:

```
{
  "name": "sticky_board",
  "version": "1.0.0",
  "description": "Post images for others to see.",
  "main": "index.js",
  "author": "ParaDrop Team"
}
```

7.7.3 Install Dependencies

Use the following command to install some dependencies that we will be using to build the application. We use express as a simple web server along with a plugin for accepting file uploads. We will also use Embedded JS (EJS) for simple templating, demonstrated later in this tutorial.

The `--save` option instructs npm to save the packages to the `package.json` file. ParaDrop will read `package.json` to install the same versions of the packages that you used for development.

```
npm install --save ejs@^2.5.6 express@^4.14.1 express-fileupload@^0.1.1
```

7.7.4 Hello World

Let's start with a minimal Hello World Express.js example. Create a file named `index.js` and add the following code:

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000, function() {
  console.log('Listening on port 3000.');
```

Run the app with the following command:

```
node index.js
```

Then load `http://localhost:3000/` in a web browser to see the result.

7.7.5 Image Uploads

Next, we will add an endpoint to receive image uploads.

```
var express = require('express');
var fileupload = require('express-fileupload');

var app = express();

// Use PARADROP_DATA_DIR when running on Paradrop and /tmp for testing.
var storage_dir = process.env.PARADROP_DATA_DIR || '/tmp';

app.use(fileupload());
app.use(express.static(storage_dir));
app.set('view engine', 'ejs');

app.post('/create', function(req, res) {
  var img = req.files.img;
  if (img) {
    img.mv(storage_dir + '/' + img.name);
  }

  res.redirect('/');
});

app.get('/', function (req, res) {
  res.render('home');
});

app.listen(3000, function() {
  console.log('Listening on port 3000.');
```

Create a new file in the views directory called home.ejs with the following contents:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>ParaDrop Sticky Board</title>
  </head>
  <body>
    <h1>ParaDrop Sticky Board</h1>
    <h2>Create a Note</h2>
    <p>Upload an image file to create a note for others to see.</p>
    <form action="/create" method="POST" enctype="multipart/form-data">
      <input type="file" name="img" />
      <input type="submit" value="Create" />
    </form>
  </body>
</html>
```

Right now it is just plain HTML. In the next section we will make use of templating to add images to the sticky board.

Run the app again and load `http://localhost:3000/`. Try using the form to upload an image. You should then be able to find your image by loading `http://localhost:3000/<filename>`.

7.7.6 Displaying Notes

The last thing the app needs to be able to do is display all of the notes that people have posted. First, add some logic to `index.js` to keep track of the most recent image uploads:

```
var express = require('express');
var fileupload = require('express-fileupload');

var app = express();

// Use PARADROP_DATA_DIR when running on Paradrop and /tmp for testing.
var storage_dir = process.env.PARADROP_DATA_DIR || '/tmp';

// Maximum number of notes to display.
var max_visible_notes = process.env.MAX_VISIBLE_NOTES || 16;

app.locals.notes = [];
for (var i = 0; i < max_visible_notes; i++) {
  if (i % 2 == 0) {
    addNote('http://pages.cs.wisc.edu/~hartung/paradrop/paradrop.png');
  } else {
    addNote('http://pages.cs.wisc.edu/~hartung/paradrop/paradrop_inverted.png');
  }
}

function addNote(img) {
  app.locals.notes.push({
    img: img,
  });

  if (app.locals.notes.length > max_visible_notes) {
    app.locals.notes = app.locals.notes.slice(-max_visible_notes);
  }
}

app.use(fileupload());
app.use(express.static(storage_dir));
app.set('view engine', 'ejs');

app.post('/create', function(req, res) {
  var img = req.files.img;
  if (img) {
    img.mv(storage_dir + '/' + img.name);
    addNote(img.name);
  }

  res.redirect('/');
});

app.get('/', function (req, res) {
  res.render('home');
});

app.listen(3000, function() {
  console.log('Listening on port 3000.');
```

The `paradrop.png` and `paradrop_inverted.png` are just used as fillers until people post other images. Feel free to use

different images.

Also, update `home.ejs`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>ParaDrop Sticky Board</title>

    <style>
      div.holder {
        float: left;
        min-width: 240px;
        width: 24%;
        padding: 5px 5px;
      }

      div.separator {
        clear: both;
      }
    </style>
  </head>
  <body>
    <h1>ParaDrop Sticky Board</h1>

    <div>
      <% for(var i = 0; i<notes.length; i++) {%>
        <div class="holder">
          </img>
        </div>
      <% } %>
    </div>

    <div class="separator"></div>

    <h2>Create a Note</h2>
    <p>Upload an image file to create a note for others to see.</p>
    <form action="/create" method="POST" enctype="multipart/form-data">
      <input type="file" name="img" />
      <input type="submit" value="Create" />
    </form>
  </body>
</html>
```

We use some Embedded JS code to loop over the array of notes stored in `app.locals.notes` and generate an `img` element for each one with the appropriate filename.

Now when you run the app and load `http://localhost:3000/` you should see the filler images. Try using the form to upload an image, and it should appear on the board.

7.7.7 Preparing the Chute

Create a file called `paradrop.yaml` with the following contents:

```
version: 1
type: light
```

(continues on next page)

(continued from previous page)

```
use: node
command: node index.js
```

This file tells ParaDrop a few things about how to run your code on a ParaDrop gateway.

Finally, add all of your new files to the git repository:

```
git add index.js package.json paradrop.yaml views/home.ejs
git commit -m "Created sticky board from tutorial"
```

Create a new repository on github.com and follow their instructions to push your code to github.

7.7.8 Registering the Chute with ParaDrop

Log on to paradrop.org and go to the Chute Store tab. Click “Create Chute” and give your chute a name and description. You may need to be creative with the name because the chute store requires unique names. Then click “Submit”.

Next, click “Create Version”. For this tutorial, there are only two important fields to fill out on this form. First, check the box to “enable web service” and enter the number 3000 because that is the port we chose in index.js. Second, select “Download from URL” for Project source and enter the github URL for your project. Then click “Submit”.

Congratulations! You have made a ParaDrop chute. If you have a ParaDrop router, you should now be able to install the chute on your router. If not, you can follow the Getting Started guide to set up a VM running ParaDrop.

Frequently Asked Questions

Please check here for issues or questions that commonly arise.

8.1 Issues with the hardware or operating system

8.1.1 Issue 1: Docker fails to start after a reboot

This can happen if either the *docker.pid* file or the *docker-containerd.pid* file was not properly cleaned up on system reboot, which causes the Docker daemon to conclude that it is already running.

To fix this, remove the pid file on the router and reboot.

```
sudo rm /var/snap/docker/current/run/docker.pid
sudo rm /var/snap/docker/current/run/docker/libcontainerd/docker-containerd.pid
sudo reboot
```

Occasionally Docker will crash and not restart properly even after a reboot. We find that disabling and re-enabling the service helps in such cases.

```
sudo snap disable docker
sudo snap enable docker
```

8.1.2 Issue 2: WiFi devices are not detected after a reboot

Occasionally, when routers start up the WiFi devices are not detected properly. When this happens the command `iw dev` will display nothing instead of the expected devices. This is usually remedied by rebooting. A global setting to reboot the router if WiFi devices are missing is available on the router settings page.

How to Contribute

This section focuses on the *ParaDrop Daemon*. This is the set of daemons and tools required to allow the Paradrop platform to function on virtual machines and real hardware. ParaDrop is an open source project. The source code of the ParaDrop daemon is available at [github](#). Issue reports and pull requests are welcomed.

9.1 ParaDrop daemon development

ParaDrop repository includes a set of tools to make development as easy as possible.

Currently this system takes the form of a bash script that automates installation and execution. This page outlines the steps required to manually install the dependencies, build the package and install the package into hardware/VMs.

We recommend using Ubuntu 16.04 LTS as the development environment for this version of ParaDrop because we use [snapcraft](#) to package and distribute the ParaDrop daemon.

You will only need to follow these instructions if you will be making changes to the ParaDrop daemon. Otherwise, you can use our pre-built ParaDrop snap or disk image from [ParaDrop release](#).

9.1.1 Building ParaDrop daemon

pdbuild.sh is the script we work with during the development. It provides following commands:

- `./pdbuild.sh setup` installs development dependencies.
- `./pdbuild.sh run` executes the ParaDrop daemon locally in the development machine. It is useful for debugging.
- `./pdbuild.sh build` builds the snap package. Check [snapcraft documentation](#) for detailed information about snap packages and snapcraft.
- `./pdbuild.sh image` builds the ubuntu core image that we can flash into SD card or SSD module of a ParaDrop router. It pre-installs the required snaps for us automatically, e.g. docker.

9.1.2 Installing ParaDrop into hardware/VMs

After the ParaDrop daemon snap is ready (paradrop-daemon-<version>-amd64.snap), we can install it on a ParaDrop router. Check [Hardware Support](#) for information about preparing a ParaDrop router.

Copy the paradrop snap to the router with ParaDrop image installed:

```
scp paradrop-daemon-<version>-amd64.snap paradrop@<router ip>:
```

Then we can log in to a ParaDrop router:

```
ssh paradrop@<router ip>
```

Install the dependent snaps in a ParaDrop router:

```
snap install docker
```

Install the newly created ParaDrop daemon snap package:

```
snap install --devmode paradrop-daemon-<version>-amd64.snap
```

9.1.3 Checking logs of ParaDrop daemon

After install the ParaDrop daemon, we can use ‘pdlog’ to check the log of ParaDrop daemon on the ParaDrop router:

```
paradrop-daemon.pdlog -f
```

9.1.4 Building ParaDrop tools

We have published the ParaDrop tools snap in the Ubuntu Snap Store. On the development machine, we can install it with below command:

```
snap install paradrop-tools
```

Get the manual of ParaDrop tools:

```
paradrop-tools.pdtools --help
```

More detailed information about ParaDrop tools can be find in [Developing Applications](#). The git repository of ParaDrop includes the source code of ParaDrop tools. Developers can build the latest version of ParaDrop tools by running below command in the folder ‘tools’:

```
snappy
```

9.2 Documentation and tests

Documentation is handled by [sphinx](#) and [readthedocs](#).

Testing is a joint effort between [nosetests](#), [travis-ci](#), and [coveralls](#).

9.2.1 Documentation

Sphinx reads files in `reStructuredText` and builds a set of HTML pages. Every time a new commit is pushed to github, readthedocs automatically updates documentation.

Additionally, sphinx knows all about python! The directives `automodule`, `autoclass`, `autofunction` and more instruct sphinx to inspect the code located in `paradrop/daemon/paradrop/` and build documentation from the docstrings within.

For example, the directive `.. automodule:: paradrop.backend` will build all the documentation for the given package. See [Docstring Conventions](#) for details.

All docstring documentation is rebuilt on every commit (unless there's a bug in the code.) Sphinx does not, however, know about structural changes in code! To alert sphinx of these changes, use the `autodoc` feature:

```
sphinx-apidoc -f -o docs/paradrop paradrop/daemon/paradrop/
```

This scans packages in the `paradrop/daemon/paradrop` directory and creates `.rst` files in `docs/paradrop`.

To create the documentation locally, run:

```
cd docs
make html
python -m SimpleHTTPServer 9999
```

Open your web browser of choice and point it to http://localhost:9999/_build/html/index.html.

9.2.2 Testing

As mentioned above, all testing is automatically run by travis-ci, a continuous integration service.

To manually run tests, install nosetest:

```
pip install nose
```

Install the required packages:

```
pip install -r docs/requirements.txt
```

Run all tests:

```
nosetests
```

How does nose detect tests? All tests live in the `tests/` directory. Nose adheres to a simple principle: anything marked with `test` in its name is most likely a test. When writing tests, make sure all functions begin with `test`.

Coverage analysis detects how much of the code is used by a test suite. If the result of the coverage is less than 100%, someone slacked. Install coveralls:

```
pip install coveralls
```

Run tests with coverage analysis:

```
nosetests --with-coverage --cover-package=paradrop
```


10.1 Host Configuration

The host configuration is a YAML file that resides on the ParaDrop device and controls many aspects of system functioning, particularly network and wireless device configuration. The host configuration may also appear in JSON format when manipulating it through the Local HTTP API or through the cloud controller. This page describes the structure and interpretation of values in the host configuration.

10.1.1 Host Configuration Object

Name	Type	Description	Validations
firewall	object	Firewall settings that apply to all network interfaces.	
firewall.defaults	object	Refer to: firewall defaults object.	
lan	object	Configuration for LAN interfaces (wired and wireless).	
lan.dhcp	object	Refer to: dhcp object	
lan.firewall	object	Firewall settings for the LAN interfaces.	
lan.firewall.defaults	object	Refer to: firewall defaults object.	
lan.firewall.forwarding	object	Settings for packet forwarding.	
lan.interfaces[]	array	List of wired interfaces to include in the LAN bridge, e.g. eth1.	

Continued on next page

Table 1 – continued from previous page

Name	Type	Description	Validations
lan.ipaddr	string	IP address to use on the LAN bridge.	
lan.netmask	string	Network mask for LAN.	
lan.proto	string	Method for setting interface IP address. ‘auto’ will choose a subnet that avoids conflict with the WAN interface.	• It must be equal to one of the elements in [“auto”, “static”]
system	object	Configure Paradrop system behaviors.	
system.autoUpdate	boolean	Enable automatically updating system software packages.	
system.chutePrefixSize	integer	The IP network size to assign to each chute.	• It must be lower than or equal to 32 • It must be greater than or equal to 0
system.chuteSubnetPool	string	The IP range available for chutes in CIDR notation or ‘auto’. ‘auto’ will choose a subnet that avoids conflict with the WAN interface.	
system.onMissingWiFi	string	Behavior if expected wireless devices are missing on boot.	• It must be equal to one of the elements in [“ignore”, “reboot”, “warn”]
telemetry	object	Configure telemetry function for collecting device measurements.	
telemetry.enabled	boolean	Enable sending device measurements to cloud controller.	
telemetry.interval	integer	Reporting interval (in seconds).	• It must be greater than or equal to 1
vlan-interfaces[]	array	Configure handling of VLAN tags on wired interfaces.	
wan	object	Configuration for WAN interface.	
wan.firewall	object	Firewall settings for the WAN interface.	
wan.firewall.defaults	object	Refer to firewall defaults object.	

Continued on next page

Table 1 – continued from previous page

Name	Type	Description	Validations
wan.interface	string	Name of interface to use for WAN.	
wan.proto	string	Method of acquiring interface IP address.	It must be equal to one of the elements in ["dhcp"]
wifi[]	array	List of physical Wi-Fi devices and their configuration.	
wifi-interfaces[]	array	List of virtual Wi-Fi interfaces and their configuration.	
zerotier	object	Configure ZeroTier service, which enables VPN-like functionality.	
zerotier.enabled	boolean	Enable the ZeroTier service.	
zerotier.networks[]	array	List of ZeroTier networks to join, using their string IDs.	Its elements must be unique

10.1.2 DHCP Object

Name	Type	Description	Validations
leasetime	string	Duration of client leases, e.g. 2h	
limit	integer	Size of address range beginning at start value.	It must be greater than or equal to 1
start	integer	Starting offset for address assignment.	It must be greater than or equal to 0

10.1.3 Firewall Defaults Object

Name	Type	Description	Validations
conntrack	object		
forward	string		It must be equal to one of the elements in ["ACCEPT", "REJECT", "DROP"]
input	string		It must be equal to one of the elements in ["ACCEPT", "REJECT", "DROP"]
masq	object		
masq_src[]	array	List of source addresses or subnets to which SNAT should be applied.	Its elements must be unique
output	string		It must be equal to one of the elements in ["ACCEPT", "REJECT", "DROP"]

10.1.4 Wi-Fi Device Object

Objects in the *wifi* array define physical device settings such as the channel and transmit power. These settings affect all interfaces in the “wifi-interfaces” array that use the corresponding device.

ParaDrop uses a deterministic system for identifying Wi-Fi devices, so that settings are applied to the same device on startup as long as there have been no hardware changes. ParaDrop numbers PCI and USB devices separately starting from zero, so a ParaDrop host with two PCI Wi-Fi cards and one USB card will have device IDs *pci-wifi-0*, *pci-wifi-1*, and *usb-wifi-0*.

The spectrum band is determined by the *hwmode* setting and the *channel* setting. They must be compatible. For 2.4 GHz channels (1-13), set *hwmode* to *11g*. For 5 GHz channels (36-165), set *hwmode* to *11a*.

Higher data rates and channel sizes (802.11n and 802.11ac) are configured with the *htmode* setting. For a 40 MHz channel width in 802.11n, set *htmode*=*HT40* or *htmode*=*HT40-*. Plus means add the next higher channel, and minus means add the lower channel. For example, setting *channel*=36 and *htmode*=*HT40+* results in using channels 36 and 40 as a 40 MHz channel.

If the hardware supports it, you can enable short guard interval for slightly higher data rates. There are separate settings for each channel width: *short_gi_20*, *short_gi_40*, and *short_gi_80*.

Name	Type	Description	Validations
channel	integer	Wi-Fi channel number.	• It must be lower than or equal to 165 • It must be greater than or equal to 1
htmode	string	Enable 802.11n or 802.11ac modes.	• It must be equal to one of the elements in [null, "HT20", "HT40+", "HT40-", "VHT20", "VHT40", "VHT80"]
hwmode	string	Basic operating mode (11b for old hardware, 11g for 2.4 GHz, 11a for 5 GHz).	• It must be equal to one of the elements in ["11b", "11g", "11a"]
id	string	Physical identifier, e.g. pci-wifi-1 or usb-wifi-0.	
rx_stbc	integer	Indicates support for receiving frames using STBC.	• It must be lower than or equal to 1 • It must be greater than or equal to 0
short_gi_20	boolean	Enable short guard interval (higher data rates) in 20 MHz channels, must be supported by device.	
short_gi_40	boolean	Enable short guard interval (higher data rates) in 40 MHz channel, must be supported by device.	
short_gi_80	boolean	Enable short guard interval (higher data rates) in 80 MHz channel, must be supported by device.	
tx_stbc	integer	Indicates support for transmitting frames using STBC.	• It must be lower than or equal to 1 • It must be greater than or equal to 0

10.1.5 Wi-Fi Interface Object

Objects in the *wifi-interfaces* array configure virtual interfaces. Each virtual interface has an underlying physical device, but there can be multiple interfaces per device up to a limit determined by the hardware. Virtual interfaces can be configured as APs or in other operating modes (with limited support).

The *encryption* setting can take a number of different values. The most common options are: “none” for an open access point, “psk2” for WPA2 Personal (PSK), and “wpa2” for WPA2 Enterprise. WPA2 Enterprise requires additional configuration to interact with an external RADIUS server.

Name	Type	Description	Validations
device	string	Physical device used by this interface, must match a device id in the wifi section.	
encryption	string	Type of wireless network security to use, e.g. none, psk2, wpa2 (Enterprise).	
mode	string	Operating mode for the interface.	• It must be equal to one of the elements in [“airshark”, “ap”, “managed”, “monitor”]
network	string	Network name the interface should be attached to, typically lan for ap mode interfaces.	
ssid	string	ESSID for ap and managed mode interfaces.	

10.2 Chute Configuration

The chute configuration is a YAML file (paradrop.yaml) that a chute developer creates to configure how resources from the host operating system should be allocated to the chute. The chute configuration may also appear in JSON format, particularly when manipulating it through the Local HTTP API or through the cloud API. This page describes the structure and interpretation of values in the chute configuration.

10.2.1 Chute Configuration Object

Name	Type	Description	Validations
environment	object	Environment variables that will be passed to the chute	
host_config	object	Options that affect the host network through Docker host config	
host_config.port_bindings	object	Bind host ports to ports inside the chute	
host_config.dns[]	array	List of DNS servers to be used by the chute	
net	object	Chute network configuration	
web	object	Declaration of web services hosted in the chute	
web.port	integer	Port number of listening web server in chute	• It must be lower than or equal to 65535 • It must be greater than or equal to 1

10.2.2 Chute Network Object

Chutes may have one or more network interfaces. All chutes are configured with a default *eth0* interface that provides WAN connectivity. Chutes may request additional network interfaces of various types by defining them in the *net* object. *net* is a dictionary, so each network object has a name of your choosing. The network name corresponds to the network name in certain Local API endpoints such as `/api/v1/chutes/(chute)/networks/(network)`.

WiFi AP Configuration

A WiFi AP interface is created by setting *type=wifi* and *mode=ap*. *ap* is the default mode for chute WiFi interfaces, so the latter option can be omitted.

Name	Type	Description	Validations
type	string (required)	Network type, must be wifi for an AP	It must match to regexp “wifi”
intfName	string (required)	Name of the network interface inside the chute	It must match to regexp “[a-z][a-z0-9]*”
mode	string	WiFi interface mode	It must match to regexp “ap”
dhcp	object	DHCP server configuration	
dhcp.leaseTime	string	Duration of client leases, e.g. 2h	
dhcp.limit	integer	Size of address range beginning at start value.	It must be greater than or equal to 1
dhcp.start	integer	Starting offset for address assignment.	It must be greater than or equal to 0
dhcp.relay	string	Address of a DHCP server if this server will be acting as a relay agent (experimental)	
ipv4_network	string	Request specific IP network in slash notation	
ssid	string	ESSID to broadcast	Its length must be less than or equal to 32
key	string	Password required from clients to connect	Its length must be greater than or equal to 8
options	object	Additional options to pass to the AP	
options.nasid	string	NAS ID for RADIUS	
options.acct_server	string	RADIUS accounting server	
options.acct_secret	string	RADIUS accounting secret	
options.acct_interval	integer	RADIUS accounting update interval (seconds)	It must be greater than or equal to 1
options.hidden	boolean	Disable broadcasting the ESSID in beacons	
options.isolate	boolean	Disable forwarding traffic between connected clients	
options.maxassoc	integer	Maximum number of associated clients	
dns[]	array	List of DNS servers to advertise to connected clients	

Monitor-mode Interface Configuration (Experimental)

A monitor-mode interface enables a chute to observe all detected WiFi traffic with RadioTap headers.

Name	Type	Description	Validations
type	string (required)	Network type, must be wifi for monitor mode	• It must match to regexp “wifi”
intfName	string (required)	Name of the network interface inside the chute	• It must match to regexp “[a-z][a-z0-9]*”
mode	string (required)	Wireless interface mode	• It must match to regexp “monitor”

VLAN Interface Configuration

A VLAN interface allows tagged traffic on the physical Ethernet ports of the device to be received by the chute. The interface must be configured with a VLAN ID. Incoming traffic with that VLAN tag will be untagged and forwarded to the chute interface. Likewise, traffic leaving the chute interface will be tagged and sent on one the physical ports.

Name	Type	Description	Validations
type	string (required)	Network type, must be vlan	It must match to regexp “wifi”
intfName	string (required)	Name of the network interface inside the chute	It must match to regexp “[a-z][a-z0-9]*”
vlan_id	integer (required)	VLAN tag for traffic to and from the chute	It must be greater than or equal to 1
dhcp	object	DHCP server configuration	
dhcp.leaseTime	string	Duration of client leases, e.g. 2h	
dhcp.limit	integer	Size of address range beginning at start value.	It must be greater than or equal to 1
dhcp.start	integer	Starting offset for address assignment.	It must be greater than or equal to 0
dhcp.relay	string	Address of a DHCP server if this server will be acting as a relay agent (experimental)	
ipv4_network	string	Request specific IP network in slash notation	
dns[]	array	List of DNS servers to advertise to connected clients	

10.2.3 Example

The following example chute configuration sets up a WiFi access point and a web server running on port 5000. First, we show the example in YAML format.

```
net:
  wifi:
    type: wifi
    intfName: wlan0
    dhcp:
      start: 3
      limit: 250
      lease: 1h
    ssid: Free WiFi
    options:
      isolate: true
      maxassoc: 100
  web:
    port: 5000
```

Here is the same example in JSON format.

```
{
  "net": {
    "wifi": {
      "type": "wifi",
      "intfName": "wlan0",
      "dhcp": {
        "start": 3,
        "limit": 250,
        "lease": "1h"
      },
      "ssid": "Free WiFi",
      "options": {
        "isolate": true,
        "maxassoc": 100
      }
    }
  },
  "web": {
    "port": 5000
  }
}
```

10.2.4 Experimental Features

ParaDrop is under heavy development. Features marked as *experimental* may be incomplete or buggy. Please contact us if you need help with any of these features.

10.3 Chute Management

Install and manage chutes on the host.

Endpoints for these functions can be found under `/api/v1/chutes`.

GET `/api/v1/chutes/`

List installed chutes.

Example request:

```
GET /api/v1/chutes/
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "environment": {},
    "name": "hello-world",
    "allocation": {
      "cpu_shares": 1024,
      "prioritize_traffic": false
    }
  },
  ...
]
```

(continues on next page)

(continued from previous page)

```

    "state": "running",
    "version": "x1511808778",
    "resources": null
  }
]

```

GET `/api/v1/chutes/(chute)/networks/network/stations/mac` Get detailed information about a connected station.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/stations/5c:59:48:7d:b9:e6
```

Example response:

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "rx_packets": "230",
  "tdls_peer": "no",
  "authenticated": "yes",
  "rx_bytes": "12511",
  "tx_bitrate": "1.0 MBit/s",
  "tx_retries": "0",
  "signal": "-45 [-49, -48] dBm",
  "authorized": "yes",
  "rx_bitrate": "65.0 MBit/s MCS 7",
  "mfp": "no",
  "tx_failed": "0",
  "inactive_time": "4688 ms",
  "mac_addr": "5c:59:48:7d:b9:e6",
  "tx_bytes": "34176",
  "wmm_wme": "yes",
  "preamble": "short",
  "tx_packets": "88",
  "signal_avg": "-44 [-48, -47] dBm"
}

```

GET `/api/v1/chutes/(chute)/networks/network/hostapd_status` Get low-level status information from the access point.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/hostapd_status
```

Example response:

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "olbc_ht": "0",
  "cac_time_left_seconds": "N/A",
  "num_sta_no_short_slot_time": "0",
  "olbc": "1",
  "num_sta_non_erp": "0",

```

(continues on next page)

(continued from previous page)

```

"ht_op_mode": "0x4",
"state": "ENABLED",
"num_sta_ht40_intolerant": "0",
"channel": "11",
"bssid[0]": "02:00:08:24:03:dd",
"ieee80211n": "1",
"cac_time_seconds": "0",
"num_sta[0]": "1",
"ieee80211ac": "0",
"phy": "phy0",
"num_sta_ht_no_gf": "1",
"freq": "2462",
"num_sta_ht_20_mhz": "1",
"num_sta_no_short_preamble": "0",
"secondary_channel": "0",
"ssid[0]": "Free WiFi",
"num_sta_no_ht": "0",
"bss[0]": "vwlان7e1b"
}

```

GET `/api/v1/chutes/ (chute) /networks/ network/stations` Get detailed information about connected wireless stations.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/stations
```

Example response:

```

HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "rx_packets": "230",
    "tdls_peer": "no",
    "authenticated": "yes",
    "rx_bytes": "12511",
    "tx_bitrate": "1.0 MBit/s",
    "tx_retries": "0",
    "signal": "-45 [-49, -48] dBm",
    "authorized": "yes",
    "rx_bitrate": "65.0 MBit/s MCS 7",
    "mfp": "no",
    "tx_failed": "0",
    "inactive_time": "4688 ms",
    "mac_addr": "5c:59:48:7d:b9:e6",
    "tx_bytes": "34176",
    "wmm_wme": "yes",
    "preamble": "short",
    "tx_packets": "88",
    "signal_avg": "-44 [-48, -47] dBm"
  }
]

```

GET `/api/v1/chutes/ (chute) /networks/ network/leases` Get current list of DHCP leases for chute network.

Returns a list of DHCP lease records with the following fields:

expires lease expiration time (seconds since Unix epoch)

mac_addr device MAC address

ip_addr device IP address

hostname name that the device reported

client_id optional identifier supplied by device

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/leases
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "client_id": "01:5c:59:48:7d:b9:e6",
    "expires": "1511816276",
    "ip_addr": "192.168.128.64",
    "mac_addr": "5c:59:48:7d:b9:e6",
    "hostname": "paradrops-iPod"
  }
]
```

GET `/api/v1/chutes/ (chute) /networks/ network/ssid` Get currently configured SSID for the chute network.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/ssid
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "ssid": "Free WiFi",
  "bssid": "02:00:08:24:03:dd"
}
```

PUT `/api/v1/chutes/ (chute) /networks/ network/ssid` Change the configured SSID for the chute network.

The change will not persist after a reboot. If a persistent change is desired, you should update the chute configuration instead.

Example request:

```
PUT /api/v1/chutes/captive-portal/networks/wifi/ssid
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
"ssid": "Best Free WiFi"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "OK"
}
```

GET `/api/v1/chutes/ (chute) /networks/`
network Get information about a network configured for the chute.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "interface": "wlan0",
  "type": "wifi",
  "name": "wifi"
}
```

GET `/api/v1/chutes/ (chute) /networks`
Get list of networks configured for the chute.

Example request:

```
GET /api/v1/chutes/captive-portal/networks
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "interface": "wlan0",
    "type": "wifi",
    "name": "wifi"
  }
]
```

GET `/api/v1/chutes/ (chute) /config`
Get current chute configuration.

Example request:

```
GET /api/v1/chutes/captive-portal/config
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "net": {
    "wifi": {
      "dhcp": {
        "lease": "1h",
        "limit": 250,
        "start": 3
      },
      "intfName": "wlan0",
      "options": {
        "isolate": True
      },
      "ssid": "Free WiFi",
      "type": "wifi"
    }
  }
}
```

PUT /api/v1/chutes/ (*chute*) /config

Update the chute configuration and restart to apply changes.

Example request:

```
PUT /api/v1/chutes/captive-portal/config
Content-Type: application/json

{
  "net": {
    "wifi": {
      "dhcp": {
        "lease": "1h",
        "limit": 250,
        "start": 3
      },
      "intfName": "wlan0",
      "options": {
        "isolate": True
      },
      "ssid": "Better Free WiFi",
      "type": "wifi"
    }
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "change_id": 1
}
```

GET /api/v1/chutes/ (*chute*) /cache

Get chute cache contents.

The chute cache is a key-value store used during chute installation. It can be useful for debugging the Paradrop platform.

GET `/api/v1/chutes/` (*chute*)
Get information about an installed chute.

Example request:

```
GET /api/v1/chutes/hello-world
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "environment": {},
  "name": "hello-world",
  "allocation": {
    "cpu_shares": 1024,
    "prioritize_traffic": false
  },
  "state": "running",
  "version": "x1511808778",
  "resources": null
}
```

10.4 Device Configuration

This module exposes device configuration.

Endpoints for these functions can be found under `/api/v1/config`.

POST `/api/v1/config/factoryReset`
Initiate the factory reset process.

PUT `/api/v1/config/hostconfig`
Replace the device's host configuration.

Example request:

```
PUT /api/v1/config/hostconfig
Content-Type: application/json

{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  change_id: 1
}
```

For a complete example, please see the Host Configuration section.

GET /api/v1/config/hostconfig

Get the device's current host configuration.

Example request:

```
GET /api/v1/config/hostconfig
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

For a complete example, please see the Host Configuration section.

GET /api/v1/config/new-config

Generate a new node configuration based on the hardware.

Example request:

```
GET /api/v1/config/new_config
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

For a complete example, please see the Host Configuration section.

POST /api/v1/config/provision

Provision the device with credentials from a cloud controller.

GET /api/v1/config/provision

Get the provision status of the device.

GET /api/v1/config/pdconf

Get configuration sections from pdconf.

This returns a list of configuration sections and whether they were successfully applied. This is intended for debugging purposes.

PUT /api/v1/config/pdconf

Trigger pdconf to reload UCI configuration files.

Trigger pdconf to reload UCI configuration files and return the status. This function is intended for low-level debugging of the paradrop pdconf module.

GET /api/v1/config/pdid

Get the device's current ParaDrop ID. This is the identifier assigned by the cloud controller.

Example request:

```
GET /api/v1/config/pdid
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  pdid: "5890e1e5ab7e317e6c6e049f"
}
```

POST /api/v1/config/sshKeys/ (user)

Manage list of authorized keys for SSH access.

GET /api/v1/config/sshKeys/ (user)

Manage list of authorized keys for SSH access.

10.5 Device Information

Provide information of the router, e.g. board version, CPU information, memory size, disk size.

Endpoints for these functions can be found under /api/v1/info.

GET /api/v1/info/environment

Get environment variables.

Returns a dictionary containing the environment variables passed to the Paradrop daemon. This is useful for development and debugging purposes (e.g. see how PATH is set on Paradrop when running in different contexts).

Example request:

```
GET /api/v1/info/environment
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "LANG": "C.UTF-8",
  "SNAP_REVISION": "x73",
  "SNAP_COMMON": "/var/snap/paradrop-daemon/common",
  "XDG_RUNTIME_DIR": "/run/user/0/snap.paradrop-daemon",
  "SNAP_USER_COMMON": "/root/snap/paradrop-daemon/common",
  "SNAP_LIBRARY_PATH": "/var/lib/snapd/lib/gl:/var/lib/snapd/void",
  "SNAP_NAME": "paradrop-daemon",
  "PWD": "/var/snap/paradrop-daemon/x73",
  "PATH": "/snap/paradrop-daemon/x73/usr/sbin:/snap/paradrop-daemon/x73/usr/bin:/
↪snap/paradrop-daemon/x73/sbin:/snap/paradrop-daemon/x73/bin:/usr/local/sbin:/
↪usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games",
  "SNAP": "/snap/paradrop-daemon/x73",
  "SNAP_DATA": "/var/snap/paradrop-daemon/x73",
  "SNAP_VERSION": "0.9.2",
  "SNAP_ARCH": "amd64",
  "SNAP_USER_DATA": "/root/snap/paradrop-daemon/x73",
  "TMPDIR": "/tmp",
  "HOME": "/root/snap/paradrop-daemon/x73",
  "SNAP_REEXEC": "",
  "LD_LIBRARY_PATH": "/var/lib/snapd/lib/gl:/var/lib/snapd/void:/snap/paradrop-
↪daemon/x73/usr/lib/x86_64-linux-gnu::/snap/paradrop-daemon/x73/lib:/snap/
↪paradrop-daemon/x73/usr/lib:/snap/paradrop-daemon/x73/lib/x86_64-linux-gnu:/
↪snap/paradrop-daemon/x73/usr/lib/x86_64-linux-gnu",
  "TMPDIR": "/tmp"
  ...
}
```

GET /api/v1/info/telemetry

Get a telemetry report.

This contains information about resource utilization by chute and system totals. This endpoint returns the same data that we periodically send to the controller if telemetry is enabled.

GET /api/v1/info/hardware

Get information about the hardware platform.

Example request:

```
GET /api/v1/info/hardware
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "wifi": [
    {
      "slot": "pci/0000:04:00.0",
      "vendorId": "0x168c",
      "macAddr": "04:f0:21:2f:b7:c1",
      "id": "pci-wifi-0",
      "deviceId": "0x003c"
    },
  ],
```

(continues on next page)

(continued from previous page)

```
{
  "slot": "pci/0000:06:00.0",
  "vendorId": "0x168c",
  "macAddr": "04:f0:21:0f:78:28",
  "id": "pci-wifi-1",
  "deviceId": "0x002a"
},
"memory": 2065195008,
"vendor": "PC Engines",
"board": "APU 1.0",
"cpu": "x86_64"
}
```

GET /api/v1/info/software

Get information about the operating system.

Returns a dictionary containing information the BIOS version, OS version, kernel version, Paradrop version, and system uptime.

Example request:

```
GET /api/v1/info/software
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "biosVersion": "SageBios_PCEngines_APU-45",
  "biosDate": "04/05/2014",
  "uptime": 15351,
  "kernelVersion": "Linux-4.4.0-101-generic",
  "pdVersion": "0.9.2",
  "biosVendor": "coreboot",
  "osVersion": "Ubuntu 4.4.0-101.124-generic 4.4.95"
}
```

GET /api/v1/info/features

Get features supported by the host.

This is a list of strings specifying features supported by the daemon.

Explanation of feature strings:

hostapd-control The daemon supports the hostapd control interface and provides a websocket channel for accessing it.

Example request:

```
GET /api/v1/info/features
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
[  
    "hostapd-control"  
]
```

CHAPTER 11

pdtools

Paradrop command line utility.

```
pdtools [OPTIONS] COMMAND [ARGS]...
```

11.1 chute

Utilities for developing chutes.

These commands all operate on a chute project in the current working directory. Remember, that all chutes must have a `paradrop.yaml` file in the top-level directory. You can create one interactively with the `initialize` command.

```
pdtools chute [OPTIONS] COMMAND [ARGS]...
```

11.1.1 add-wifi-ap

Add a WiFi AP to the chute configuration.

ESSID must be between 1 and 32 characters in length. Spaces are allowed if you enclose the argument in quotation marks.

```
pdtools chute add-wifi-ap [OPTIONS] ESSID
```

Options

--password <password>

Password for the network, must be at least 8 characters if specified.

--force

Overwrite an existing section in the configuration file.

Arguments

ESSID

Required argument

11.1.2 export-configuration

Export chute configuration in JSON or YAML format.

The configuration format used by the cloud API is slightly different from the paradrop.yaml file. This command can export a JSON object in a form suitable for installing the chute through the cloud API.

The config object will usually be used in an envelope like the following: {

```
  "updateClass": "CHUTE", "updateType": "update", "config": { config-object }
```

```
}
```

```
pdtools chute export-configuration [OPTIONS]
```

Options

-f, --format <format>

Format (json or yaml)

11.1.3 initialize

Interactively create a paradrop.yaml file.

```
pdtools chute initialize [OPTIONS]
```

11.1.4 set

Set a value in the paradrop.yaml file.

PATH must be a dot-separated path to a value in the paradrop.yaml file, such as "config.web.port". VALUE will be interpreted as a string, numeric, or boolean type as appropriate.

Changing values inside a list is not currently supported. For that you will need to edit the file directly.

Example: set config.web.port 80

```
pdtools chute set [OPTIONS] PATH VALUE
```

Arguments

PATH

Required argument

VALUE

Required argument

11.2 cloud

Access services provided by a cloud controller.

By default the cloud controller is assumed to be paradrop.org. This can be configured through the environment variable PDSERVER_URL.

```
pdtools cloud [OPTIONS] COMMAND [ARGS]...
```

11.2.1 claim-node

Take ownership of a node by using a claim token.

TOKEN is a hard-to-guess string that the previous owner would have configured when setting the node's status as orphaned.

```
pdtools cloud claim-node [OPTIONS] TOKEN
```

Arguments

TOKEN

Required argument

11.2.2 create-node

Create a new node to be tracked by the controller.

NAME must be unique among the nodes that you own and may only consist of lowercase letters, numbers, and hyphens. It must also begin with a letter.

```
pdtools cloud create-node [OPTIONS] NAME
```

Options

--orphaned, --not-orphaned

Allow another user to claim the node.

--claim <claim>

Claim token required to claim the node.

Arguments

NAME

Required argument

11.2.3 delete-node

Delete a node that is tracked by the controller.

NAME must be the name of a node that you own.

```
pdtools cloud delete-node [OPTIONS] NAME
```

Arguments

NAME

Required argument

11.2.4 describe-node

Get detailed information about an existing node.

NAME must be the name of a node that you own.

```
pdtools cloud describe-node [OPTIONS] NAME
```

Arguments

NAME

Required argument

11.2.5 edit-node-description

Interactively edit the node description and save.

NAME must be the name of a node that you own.

Open the text editor specified by the EDITOR environment variable with the current node description. If you save and exit, the changes will be applied to the node.

```
pdtools cloud edit-node-description [OPTIONS] NAME
```

Arguments

NAME

Required argument

11.2.6 group-add-node

Add a node to a group for other members to access.

GROUP must be the string ID of a group. NODE must be the name of a node that you control.

```
pdtools cloud group-add-node [OPTIONS] GROUP NODE
```

Arguments

GROUP

Required argument

NODE

Required argument

11.2.7 help

Show this message and exit.

```
pdtools cloud help [OPTIONS]
```

11.2.8 list-groups

List groups that you belong to.

```
pdtools cloud list-groups [OPTIONS]
```

11.2.9 list-nodes

List nodes that you own or have access to.

```
pdtools cloud list-nodes [OPTIONS]
```

11.2.10 login

Interactively login to your user account on the controller.

Authenticate with the controller using account credentials that you created either through the website or with the register command. Typically, the username will be your email address.

```
pdtools cloud login [OPTIONS]
```

11.2.11 logout

Log out and remove stored credentials.

```
pdtools cloud logout [OPTIONS]
```

11.2.12 register

Interactively create an account on the controller.

```
pdtools cloud register [OPTIONS]
```

11.2.13 rename-node

Change the name of a node.

NAME must be the name of a node that you control. NEW_NAME is the desired new name. It must adhere to the same naming rules as for the create-node command, namely, it must begin with a letter and consist of only lowercase letters, numbers, and hyphen.

```
pdtools cloud rename-node [OPTIONS] NAME NEW_NAME
```

Arguments

NAME

Required argument

NEW_NAME

Required argument

11.3 device

(deprecated) Sub-tree for configuring a device.

These commands are deprecated. Please use the equivalent commands under *pdtools node -help*.

```
pdtools device [OPTIONS] ADDRESS COMMAND [ARGS]...
```

Arguments

ADDRESS

Required argument

11.3.1 audio

Control device audio properties.

```
pdtools device audio [OPTIONS] COMMAND [ARGS]...
```

info

Get audio server information.

```
pdtools device audio info [OPTIONS]
```

load-module

Load a module.

```
pdtools device audio load-module [OPTIONS] NAME
```

Arguments

NAME

Required argument

modules

List loaded modules.

```
pdtools device audio modules [OPTIONS]
```

sink

Configure audio sink.

```
pdtools device audio sink [OPTIONS] SINK_NAME COMMAND [ARGS]...
```

Arguments

SINK_NAME

Required argument

volume

Set sink volume.

```
pdtools device audio sink volume [OPTIONS] [CHANNEL_VOLUME]...
```

Arguments

CHANNEL_VOLUME

Optional argument(s)

sinks

List audio sinks.

```
pdtools device audio sinks [OPTIONS]
```

source

Configure audio source.

```
pdtools device audio source [OPTIONS] SOURCE_NAME COMMAND [ARGS]...
```

Arguments

SOURCE_NAME

Required argument

volume

Set source volume.

```
pdtools device audio source volume [OPTIONS] [CHANNEL_VOLUME] ...
```

Arguments

CHANNEL_VOLUME

Optional argument(s)

sources

List audio sources.

```
pdtools device audio sources [OPTIONS]
```

11.3.2 changes

List changes in the working queue.

```
pdtools device changes [OPTIONS]
```

11.3.3 chute

Sub-tree for configuring a chute.

```
pdtools device chute [OPTIONS] CHUTE COMMAND [ARGS] ...
```

Arguments

CHUTE

Required argument

cache

Get details from the chute installation.

```
pdtools device chute cache [OPTIONS]
```

config

Get the chute's current configuration.

```
pdtools device chute config [OPTIONS]
```

delete

Uninstall the chute.

```
pdtools device chute delete [OPTIONS]
```

edit-environment

Interactively edit the chute environment vairables.

```
pdtools device chute edit-environment [OPTIONS]
```

info

Get information about the chute.

```
pdtools device chute info [OPTIONS]
```

logs

Watch log messages from a chute.

```
pdtools device chute logs [OPTIONS]
```

network

Sub-tree for accessing chute network.

```
pdtools device chute network [OPTIONS] NETWORK COMMAND [ARGS]...
```

Arguments

NETWORK

Required argument

station

Sub-tree for accessing network stations.

```
pdtools device chute network station [OPTIONS] STATION COMMAND [ARGS]...
```

Arguments

STATION

Required argument

delete

Kick a station off the network.

```
pdtools device chute network station delete [OPTIONS]
```

show

Show station information.

```
pdtools device chute network station show [OPTIONS]
```

stations

List stations connected to the network.

```
pdtools device chute network stations [OPTIONS]
```

networks

List the chute's networks.

```
pdtools device chute networks [OPTIONS]
```

reconfigure

Reconfigure the chute without rebuilding.

```
pdtools device chute reconfigure [OPTIONS]
```

restart

Restart the chute.

```
pdtools device chute restart [OPTIONS]
```

shell

Open a shell inside a chute.

This requires you to have enabled SSH access to the device and installed bash inside your chute.

```
pdtools device chute shell [OPTIONS]
```

start

Start the chute.

```
pdtools device chute start [OPTIONS]
```

stop

Stop the chute.

```
pdtools device chute stop [OPTIONS]
```

update

Update the chute from the working directory.

```
pdtools device chute update [OPTIONS]
```

11.3.4 chutes

View or create chutes on the router.

```
pdtools device chutes [OPTIONS] COMMAND [ARGS]...
```

create

Install a chute from the working directory.

```
pdtools device chutes create [OPTIONS]
```

list

List chutes installed on the router.

```
pdtools device chutes list [OPTIONS]
```

11.3.5 hostconfig

Sub-tree for the host configuration.

```
pdtools device hostconfig [OPTIONS] COMMAND [ARGS]...
```

change

Change one setting in the host configuration.

```
pdtools device hostconfig change [OPTIONS] OPTION VALUE
```

Arguments

OPTION

Required argument

VALUE

Required argument

edit

Interactively edit the host configuration.

```
pdtools device hostconfig edit [OPTIONS]
```

11.3.6 login

Log in using device-local credentials.

```
pdtools device login [OPTIONS]
```

11.3.7 logout

Log out by removing any stored access tokens.

```
pdtools device logout [OPTIONS]
```

11.3.8 password

Change the router admin password.

```
pdtools device password [OPTIONS]
```

11.3.9 pdconf

Access the pdconf subsystem.

pdconf manages low-level configuration of the Paradrop device. These commands are implemented for debugging purposes and are not intended for ordinary configuration purposes.

```
pdtools device pdconf [OPTIONS] COMMAND [ARGS]...
```

reload

Force pdconf to reload files.

```
pdtools device pdconf reload [OPTIONS]
```

show

Show status of pdconf subsystem.

```
pdtools device pdconf show [OPTIONS]
```

11.3.10 provision

Provision the router.

```
pdtools device provision [OPTIONS] ROUTER_ID ROUTER_PASSWORD
```

Options

--server <server>

--wamp <wamp>

Arguments

ROUTER_ID
Required argument

ROUTER_PASSWORD
Required argument

11.3.11 snapd

Access the snapd subsystem.

```
pdtools device snapd [OPTIONS] COMMAND [ARGS]...
```

connectall

Connect all interfaces.

```
pdtools device snapd connectall [OPTIONS]
```

createuser

Create user account.

```
pdtools device snapd createuser [OPTIONS] EMAIL
```

Arguments

EMAIL

Required argument

11.3.12 sshkeys

Sub-tree for accessing SSH authorized keys.

```
pdtools device sshkeys [OPTIONS] COMMAND [ARGS]...
```

Options

--user <user>

add

Add an authorized key from a file.

```
pdtools device sshkeys add [OPTIONS] PATH
```

Arguments

PATH

Required argument

list

List authorized keys.

```
pdtools device sshkeys list [OPTIONS]
```

11.3.13 watch

Stream messages for a change in progress.

```
pdtools device watch [OPTIONS] CHANGE_ID
```

Arguments

CHANGE_ID

Required argument

11.4 group

(deprecated) Manage a user group.

These commands are deprecated. Please use the equivalent commands under *pdtools cloud -help*.

```
pdtools group [OPTIONS] GROUP_ID COMMAND [ARGS]...
```

Arguments

GROUP_ID

Required argument

11.4.1 add-router

Add a router to the group.

```
pdtools group add-router [OPTIONS] ROUTER_ID
```

Arguments

ROUTER_ID

Required argument

11.5 help

Show this message and exit

```
pdtools help [OPTIONS]
```

11.6 list-groups

(deprecated) List user groups.

Please use *pdtools cloud list-groups*.

```
pdtools list-groups [OPTIONS]
```

11.7 node

Manage a Paradrop edge compute node.

These commands respect the following environment variables:

PDTOOLS_NODE_TARGET Default target node name or address.

```
pdtools node [OPTIONS] COMMAND [ARGS]...
```

Options

-t, --target <target>
Target node name or address

11.7.1 connect-snap-interfaces

Connect all interfaces for installed snaps.

```
pdtools node connect-snap-interfaces [OPTIONS]
```

11.7.2 create-user

Create local Linux user connected to Ubuntu store account.

EMAIL must be an email address which is registered as Ubuntu One account. The name of the local account that is created will depend on the email address used. If in doubt, use “info@paradrop.io”, which will result in a user named “paradrop” being created.

```
pdtools node create-user [OPTIONS] EMAIL
```

Arguments

EMAIL
Required argument

11.7.3 describe-audio

Display audio subsystem information.

Display information from the local PulseAudio server such as the default source and sink.

```
pdtools node describe-audio [OPTIONS]
```

11.7.4 describe-chute

Display information about a chute.

CHUTE must be the name of an installed chute.

```
pdtools node describe-chute [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.5 describe-chute-cache

Show internal details from a chute installation.

CHUTE must be the name of an installed chute.

This information is intended for Paradrop daemon developers for debugging purposes. The output is not expected to remain stable between Paradrop versions.

```
pdtools node describe-chute-cache [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.6 describe-chute-configuration

Display configuration of a chute.

CHUTE must be the name of an installed chute.

This information corresponds to the “config” section in a chute’s paradrop.yaml file.

```
pdtools node describe-chute-configuration [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.7 describe-chute-network-client

Display information about a network client.

CHUTE must be the name of an installed chute. NETWORK must be the name of one of the chute’s configured networks. Typically, this will be “wifi”. CLIENT identifies the network client, such as a MAC address.

```
pdtools node describe-chute-network-client [OPTIONS] CHUTE NETWORK CLIENT
```

Arguments

CHUTE

Required argument

NETWORK

Required argument

CLIENT

Required argument

11.7.8 describe-pdconf

Show status of the pdconf subsystem.

This information is intended for Paradrop daemon developers for debugging purposes.

```
pdtools node describe-pdconf [OPTIONS]
```

11.7.9 describe-provision

Show provisioning status of the node.

This shows whether the node is associated with a cloud controller, and if so, its identifier.

```
pdtools node describe-provision [OPTIONS]
```

11.7.10 edit-chute-configuration

Interactively edit the chute configuration and restart it.

CHUTE must be the name of an installed chute.

Open the text editor specified by the EDITOR environment variable with the current chute configuration. If you save and exit, the new configuration will be applied and the chute restarted.

```
pdtools node edit-chute-configuration [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.11 edit-chute-variables

Interactively edit a chute's environment variables and restart it.

CHUTE must be the name of an installed chute.

Open the text editor specified by the EDITOR environment variable with the current chute environment variables. If you save and exit, the new settings will be applied and the chute restarted.

```
pdtools node edit-chute-variables [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.12 edit-configuration

Interactively edit the node configuration and apply changes.

Open the text editor specified by the EDITOR environment variable with the current node configuration. If you save and exit, the new configuration will be applied to the node.

```
pdtools node edit-configuration [OPTIONS]
```

11.7.13 export-configuration

Display the node configuration in the desired format.

```
pdtools node export-configuration [OPTIONS]
```

Options

-f, --format <format>
Format (json or yaml)

11.7.14 generate-configuration

Generate a new node configuration based on detected hardware.

The new configuration is not automatically applied. Rather, you can save it to file and use the import-configuration command to apply it.

```
pdtools node generate-configuration [OPTIONS]
```

Options

-f, --format <format>
Format (json or yaml)

11.7.15 help

Show this message and exit.

```
pdtools node help [OPTIONS]
```

11.7.16 import-configuration

Import a node configuration from file and apply changes.

PATH must be a path to a node configuration file in YAML format.

```
pdtools node import-configuration [OPTIONS] PATH
```

Arguments

PATH

Required argument

11.7.17 import-ssh-key

Add an authorized key from a public key file.

PATH must be a path to a public key file, which corresponds to a private key that SSH can use for authentication. Typically, ssh-keygen will place the public key in “~/.ssh/id_rsa.pub”.

```
pdtools node import-ssh-key [OPTIONS] PATH
```

Options

-u, --user <user>
Local username

Arguments

PATH

Required argument

11.7.18 install-chute

Install a chute from the working directory.

Install the files in the current directory as a chute on the node. The directory must contain a paradrop.yaml file. The entire directory will be copied to the node for installation.

```
pdtools node install-chute [OPTIONS]
```

Options

-d, --directory <directory>
Directory containing chute files

11.7.19 list-audio-modules

List modules loaded by the audio subsystem.

```
pdtools node list-audio-modules [OPTIONS]
```

11.7.20 list-audio-sinks

List audio sinks.

```
pdtools node list-audio-sinks [OPTIONS]
```

11.7.21 list-audio-sources

List audio sources.

```
pdtools node list-audio-sources [OPTIONS]
```

11.7.22 list-changes

List queued or in progress changes.

```
pdtools node list-changes [OPTIONS]
```

11.7.23 list-chute-network-clients

List clients connected to the chute's network.

CHUTE must be the name of an installed chute. NETWORK must be the name of one of its configured networks. Typically, this will be called "wifi".

```
pdtools node list-chute-network-clients [OPTIONS] CHUTE NETWORK
```

Arguments

CHUTE

Required argument

NETWORK

Required argument

11.7.24 list-chute-networks

List networks configured by a chute.

CHUTE must be the name of an installed chute.

```
pdtools node list-chute-networks [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.25 list-chutes

List chutes installed on the node

```
pdtools node list-chutes [OPTIONS]
```

11.7.26 list-snap-interfaces

List interfaces for snaps installed on the node.

```
pdtools node list-snap-interfaces [OPTIONS]
```

11.7.27 list-ssh-keys

List keys authorized for SSH access to the node.

```
pdtools node list-ssh-keys [OPTIONS]
```

Options

-u, --user <user>

Local username

11.7.28 load-audio-module

Load a module into the audio subsystem.

MODULE must be the name of a PulseAudio module such as “module-loopback”.

```
pdtools node load-audio-module [OPTIONS] MODULE
```

Arguments

MODULE

Required argument

11.7.29 login

Interactively log in using the local admin password.

Authenticate with the node using the local username and password. Typically, the username will be “paradrop”. The password can be set with the set-password command.

```
pdtools node login [OPTIONS]
```

11.7.30 logout

Log out and remove stored credentials.

```
pdtools node logout [OPTIONS]
```

11.7.31 open-chute-shell

Open a shell inside the running chute.

CHUTE must be the name of a running chute.

This requires you to have enabled SSH access to the device and installed bash inside your chute.

Changes made to files inside the chute may not be persistent if the chute or the node is restarted. Only changes to files in the “/data” directory will be preserved.

```
pdtools node open-chute-shell [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.32 provision

Associate the node with a cloud controller.

ID and KEY are credentials that can be found when creating a node on the controller, either through the website or through *pdtools cloud create-node*. They may also be referred to as the Router ID and the Router Password.

```
pdtools node provision [OPTIONS] ID KEY
```

Options

-c, --controller <controller>
Cloud controller endpoint

-w, --wamp <wamp>
WAMP endpoint

Arguments

ID

Required argument

KEY

Required argument

11.7.33 reboot

Reboot the node.

```
pdtools node reboot [OPTIONS]
```

11.7.34 remove-chute

Remove a chute from the node.

CHUTE must be the name of an installed chute.

```
pdtools node remove-chute [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.35 remove-chute-network-client

Remove a connected client from the chute's network.

CHUTE must be the name of an installed chute. NETWORK must be the name of one of the chute's configured networks. Typically, this will be "wifi". CLIENT identifies the network client, such as a MAC address.

Only implemented for wireless clients, this effectively kicks the client off the network.

```
pdtools node remove-chute-network-client [OPTIONS] CHUTE NETWORK CLIENT
```

Arguments

CHUTE

Required argument

NETWORK

Required argument

CLIENT

Required argument

11.7.36 restart-chute

Restart a chute.

CHUTE must be the name of an installed chute.

```
pdtools node restart-chute [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.37 set-configuration

Change a node configuration value and apply.

PATH must be a dot-separated path to a value in the node configuration, such as “system.onMissingWiFi”. VALUE will be interpreted as a string, numeric, or boolean type as appropriate.

Changing values inside a list is not currently supported. Use edit-configuration instead.

```
pdtools node set-configuration [OPTIONS] PATH VALUE
```

Arguments

PATH

Required argument

VALUE

Required argument

11.7.38 set-password

Change the local admin password.

Set the password required by *pdtools node login* and the local web-based administration page.

```
pdtools node set-password [OPTIONS]
```

11.7.39 set-sink-volume

Configure audio sink volume.

SINK must be the name of a PulseAudio sink. VOLUME should be one (applied to all channels) or multiple (one for each channel) floating point values between 0 and 1.

```
pdtools node set-sink-volume [OPTIONS] SINK [VOLUME]...
```

Arguments

SINK

Required argument

VOLUME

Optional argument(s)

11.7.40 set-source-volume

Configure audio source volume.

SOURCE must be the name of a PulseAudio source. VOLUME should be one (applied to all channels) or multiple (one for each channel) floating point values between 0 and 1.

```
pdtools node set-source-volume [OPTIONS] SOURCE [VOLUME]...
```

Arguments

SOURCE

Required argument

VOLUME

Optional argument(s)

11.7.41 shutdown

Shut down the node.

```
pdtools node shutdown [OPTIONS]
```

11.7.42 start-chute

Start a stopped chute.

CHUTE must be the name of a stopped chute.

```
pdtools node start-chute [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.43 stop-chute

Stop a running chute.

CHUTE must be the name of a running chute.

```
pdtools node stop-chute [OPTIONS] CHUTE
```

Arguments

CHUTE

Required argument

11.7.44 trigger-pdconf

Trigger pdconf to reload configuration.

This function is intended for Paradrop daemon developers for debugging purposes. Generally, you should use edit-configuration and edit-chute-configuration for making configuration changes.

```
pdtools node trigger-pdconf [OPTIONS]
```

11.7.45 update-chute

Install a new version of the chute from the working directory.

Install the files in the current directory as a chute on the node. The directory must contain a paradrop.yaml file. The entire directory will be copied to the node for installation.

```
pdtools node update-chute [OPTIONS]
```

Options

-d, --directory <directory>
Directory containing chute files

11.7.46 watch-change-logs

Stream log messages from an in-progress change.

CHANGE_ID must be the ID of a queued or in-progress change as retrieved from the list-changes command.

```
pdtools node watch-change-logs [OPTIONS] CHANGE_ID
```

Arguments

CHANGE_ID
Required argument

11.7.47 watch-chute-logs

Stream log messages from a running chute.

CHUTE must be the name of a running chute.

```
pdtools node watch-chute-logs [OPTIONS] CHUTE
```

Arguments

CHUTE
Required argument

11.7.48 watch-logs

Stream log messages from the Paradrop daemon.

```
pdtools node watch-logs [OPTIONS]
```

11.8 routers

(deprecated) Access router information on the controller.

These commands are deprecated. Please use the equivalent commands under *pdtools cloud -help*.

```
pdtools routers [OPTIONS] COMMAND [ARGS]...
```

11.8.1 claim

Claim an existing router.

```
pdtools routers claim [OPTIONS] TOKEN
```

Arguments

TOKEN

Required argument

11.8.2 create

Create a new router.

```
pdtools routers create [OPTIONS] NAME
```

Options

--orphaned, --not-orphaned

--claim <claim>

Arguments

NAME

Required argument

11.8.3 delete

Delete a router.

```
pdtools routers delete [OPTIONS] ROUTER_ID
```

Arguments

ROUTER_ID

Required argument

11.8.4 list

List routers.

```
pdtools routers list [OPTIONS]
```

11.9 store

Publish and install from the public chute store.

By default the cloud controller is assumed to be paradrop.org. This can be configured through the environment variable `PDSERVER_URL`.

It is recommended that you log in with the cloud login command before using the store commands.

```
pdtools store [OPTIONS] COMMAND [ARGS]...
```

11.9.1 create-version

Push a new version of the chute to the store.

```
pdtools store create-version [OPTIONS]
```

11.9.2 describe-chute

Show detailed information about a chute in the store.

NAME must be the name of a chute in the store.

```
pdtools store describe-chute [OPTIONS] NAME
```

Arguments

NAME

Required argument

11.9.3 help

Show this message and exit.

```
pdtools store help [OPTIONS]
```

11.9.4 install-chute

Install a chute from the store.

CHUTE must be the name of a chute in the store. NODE must be the name of a node that you control.

```
pdtools store install-chute [OPTIONS] CHUTE NODE
```

Options

-v, --version <version>
Version of the chute to install.

Arguments

CHUTE
Required argument

NODE
Required argument

11.9.5 list-chutes

List chutes in the store that you own or have access to.

```
pdtools store list-chutes [OPTIONS]
```

11.9.6 list-versions

List versions of a chute in the store.

NAME must be the name of a chute in the store.

```
pdtools store list-versions [OPTIONS] NAME
```

Arguments

NAME
Required argument

11.9.7 register

Register a chute with the store.

The chute information including name will be taken from the paradrop.yaml file in the current working directory. If you receive an error, it may be that a chute with the same name is already registered.

```
pdtools store register [OPTIONS]
```

Options

--public, --not-public

List the chute publicly for other users to download.

11.10 wizard

Set up environment for development.

```
pdtools wizard [OPTIONS]
```


12.1 Subpackages

12.1.1 paradrop.airshark package

Submodules

paradrop.airshark.airshark module

```
class AirsharkManager
    Bases: object

    add_analyzer_observer (observer)
    add_spectrum_observer (observer)
    check_spectrum ()
    on_analyzer_message (message)
    on_interface_down (interface)
    on_interface_up (interface)
    read_raw_samples ()
    remove_analyzer_observer (observer)
    remove_spectrum_observer (observer)
    status ()
```

paradrop.airshark.analyzer module

```
class AnalyzerProcessProtocol (airshark_manager)
    Bases: twisted.internet.protocol.ProcessProtocol
```

```

childDataReceived(childFd, data)
connectionMade()
feedSpectrumData(data)
isRunning()
processEnded(status)
stop()

```

paradrop.airshark.scanner module

```

class Scanner(interface)
    Bases: object
    cmd_chanscan()
    cmd_disable()
    cmd_set_samplecount(count)
    cmd_set_short_repeat(short_repeat)
    debugfs_dir = None
    dev_to_phy(dev)
    freqlist = None
    get_debugfs_dir()
    interface = None
    process = None
    set_freqs(minf, maxf, spacing)
    spectrum_reader = None
    start()
    stop()

```

paradrop.airshark.spectrum_reader module

```

class SpectrumReader(path)
    Bases: object
    static decode()
        For information about the decoding of spectral samples see: https://wireless.wiki.kernel.org/en/users/drivers/ath9k/spectral\_scan https://github.com/erikarn/ath\_radar\_stuff/tree/master/lib and your ath9k implementation in e.g. /drivers/net/wireless/ath/ath9k/common-spectral.c
    flush()
    hdrsize = 3
    pktsize = 73
    read_samples()
    sc_wide = 0.3125

```

Module contents

12.1.2 paradrop.backend package

Submodules

paradrop.backend.airshark_api module

APIs for developers to check whether Airshark feature is available or not

class `AirsharkApi` (*airshark_manager*)

routes

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar _url_map: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar _endpoints: A C{dict} mapping endpoint names to handler functions.

status (*request*)

paradrop.backend.airshark_ws module

class `AirsharkAnalyzerFactory` (*airshark_manager, *args, **kwargs*)

Bases: autobahn.twisted.websocket.WebSocketServerFactory

buildProtocol (*addr*)

Create an instance of a subclass of Protocol.

The returned instance will handle input on an incoming server connection, and an attribute “factory” pointing to the creating factory.

Alternatively, C{None} may be returned to immediately close the new connection.

Override this method to alter how Protocol instances get created.

@param *addr*: an object implementing L{twisted.internet.interfaces.IAddress}

class `AirsharkAnalyzerProtocol` (*factory*)

Bases: autobahn.twisted.websocket.WebSocketServerProtocol

onClose (*wasClean, code, reason*)

Implements autobahn.websocket.interfaces.IWebSocketChannel.onClose()

onOpen ()

Implements autobahn.websocket.interfaces.IWebSocketChannel.onOpen()

on_analyzer_message (*message*)

class `AirsharkSpectrumFactory` (*airshark_manager, *args, **kwargs*)

Bases: autobahn.twisted.websocket.WebSocketServerFactory

buildProtocol (*addr*)

Create an instance of a subclass of Protocol.

The returned instance will handle input on an incoming server connection, and an attribute “factory” pointing to the creating factory.

Alternatively, C{None} may be returned to immediately close the new connection.

Override this method to alter how Protocol instances get created.

@param addr: an object implementing L{twisted.internet.interfaces.IAddress}

class AirsharkSpectrumProtocol (*factory*)

Bases: autobahn.twisted.websocket.WebSocketServerProtocol

onClose (*wasClean, code, reason*)

Implements autobahn.websocket.interfaces.IWebSocketChannel.onClose()

onOpen ()

Implements autobahn.websocket.interfaces.IWebSocketChannel.onOpen()

on_spectrum_data (*data*)

paradrop.backend.auth module

class AuthApi (*password_manager, token_manager*)

Bases: object

local_login (*request*)

Login using local authentication (username+password).

routes

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar **_url_map**: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar **_endpoints**: A C{dict} mapping endpoint names to handler functions.

check_auth (*password_manager, token_manager, auth_header*)

get_allowed_bearer ()

Return set of allowed bearer tokens.

get_username_password (*userpass*)

Please note: username and password can either be presented in plain text such as “admin:password” or base64 encoded such as “YWRtaW46GFzc3dvcmQ=”. Both forms should be returned from this function.

requires_auth (*func*)

Use as a decorator for API functions to require authorization.

This checks the Authorization HTTP header. It handles username and password as well as bearer tokens.

verify_password (*password_manager, userpass*)

paradrop.backend.chute_api module

Install and manage chutes on the host.

Endpoints for these functions can be found under /api/v1/chutes.

class ChuteApi (*update_manager*)

Bases: object

create_chute (*request*)

delete_chute (*request, chute*)

delete_station (*request, chute, network, mac*)

get_chute (*request, chute*)

Get information about an installed chute.

Example request:

```
GET /api/v1/chutes/hello-world
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "environment": {},
  "name": "hello-world",
  "allocation": {
    "cpu_shares": 1024,
    "prioritize_traffic": false
  },
  "state": "running",
  "version": "x1511808778",
  "resources": null
}
```

get_chute_cache (*request, chute*)

Get chute cache contents.

The chute cache is a key-value store used during chute installation. It can be useful for debugging the Paradrop platform.

get_chute_config (*request, chute*)

Get current chute configuration.

Example request:

```
GET /api/v1/chutes/captive-portal/config
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "net": {
    "wifi": {
      "dhcp": {
        "lease": "1h",
        "limit": 250,
        "start": 3
      },
      "intfName": "wlan0",
      "options": {
        "isolate": True
      },
      "ssid": "Free WiFi",
      "type": "wifi"
    }
  }
}
```

get_chutes (*request*)

List installed chutes.

Example request:

```
GET /api/v1/chutes/
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "environment": {},
    "name": "hello-world",
    "allocation": {
      "cpu_shares": 1024,
      "prioritize_traffic": false
    },
    "state": "running",
    "version": "x1511808778",
    "resources": null
  }
]
```

get_hostapd_status (*request, chute, network*)

Get low-level status information from the access point.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/hostapd_status
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "olbc_ht": "0",
  "cac_time_left_seconds": "N/A",
  "num_sta_no_short_slot_time": "0",
  "olbc": "1",
  "num_sta_non_erp": "0",
  "ht_op_mode": "0x4",
  "state": "ENABLED",
  "num_sta_ht40_intolerant": "0",
  "channel": "11",
  "bssid[0]": "02:00:08:24:03:dd",
  "ieee80211n": "1",
  "cac_time_seconds": "0",
  "num_sta[0]": "1",
  "ieee80211ac": "0",
  "phy": "phy0",
  "num_sta_ht_no_gf": "1",
  "freq": "2462",
  "num_sta_ht_20_mhz": "1",
  "num_sta_no_short_preamble": "0",
  "secondary_channel": "0",
  "ssid[0]": "Free WiFi",
  "num_sta_no_ht": "0",
  "bss[0]": "v wlan7e1b"
}
```

get_leases (*request, chute, network*)

Get current list of DHCP leases for chute network.

Returns a list of DHCP lease records with the following fields:

expires lease expiration time (seconds since Unix epoch)

mac_addr device MAC address

ip_addr device IP address

hostname name that the device reported

client_id optional identifier supplied by device

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/leases
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "client_id": "01:5c:59:48:7d:b9:e6",
    "expires": "1511816276",
    "ip_addr": "192.168.128.64",
    "mac_addr": "5c:59:48:7d:b9:e6",
    "hostname": "paradrops-iPod"
  }
]
```

get_network (*request, chute, network*)

Get information about a network configured for the chute.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "interface": "wlan0",
  "type": "wifi",
  "name": "wifi"
}
```

get_networks (*request, chute*)

Get list of networks configured for the chute.

Example request:

```
GET /api/v1/chutes/captive-portal/networks
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "interface": "wlan0",
    "type": "wifi",
    "name": "wifi"
  }
]
```

get_ssid (*request, chute, network*)

Get currently configured SSID for the chute network.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/ssid
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "ssid": "Free WiFi",
  "bssid": "02:00:08:24:03:dd"
}
```

get_station (*request, chute, network, mac*)

Get detailed information about a connected station.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/stations/5c:59:48:7d:b9:e6
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "rx_packets": "230",
  "tdls_peer": "no",
  "authenticated": "yes",
  "rx_bytes": "12511",
  "tx_bitrate": "1.0 MBit/s",
  "tx_retries": "0",
  "signal": "-45 [-49, -48] dBm",
  "authorized": "yes",
  "rx_bitrate": "65.0 MBit/s MCS 7",
  "mfp": "no",
  "tx_failed": "0",
  "inactive_time": "4688 ms",
  "mac_addr": "5c:59:48:7d:b9:e6",
  "tx_bytes": "34176",
  "wmm_wme": "yes",
  "preamble": "short",
  "tx_packets": "88",
```

(continues on next page)

(continued from previous page)

```

    "signal_avg": "-44 [-48, -47] dBm"
  }

```

get_stations (*request, chute, network*)

Get detailed information about connected wireless stations.

Example request:

```
GET /api/v1/chutes/captive-portal/networks/wifi/stations
```

Example response:

```

HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "rx_packets": "230",
    "tdls_peer": "no",
    "authenticated": "yes",
    "rx_bytes": "12511",
    "tx_bitrate": "1.0 MBit/s",
    "tx_retries": "0",
    "signal": "-45 [-49, -48] dBm",
    "authorized": "yes",
    "rx_bitrate": "65.0 MBit/s MCS 7",
    "mfp": "no",
    "tx_failed": "0",
    "inactive_time": "4688 ms",
    "mac_addr": "5c:59:48:7d:b9:e6",
    "tx_bytes": "34176",
    "wmm_wme": "yes",
    "preamble": "short",
    "tx_packets": "88",
    "signal_avg": "-44 [-48, -47] dBm"
  }
]

```

hostapd_control (*request, chute, network*)**restart_chute** (*request, chute*)**routes**

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar **_url_map**: A C{werkzeug.routing.Map} object which will be used for routing resolution.@ivar **_endpoints**: A C{dict} mapping endpoint names to handler functions.**set_chute_config** (*request, chute*)

Update the chute configuration and restart to apply changes.

Example request:

```

PUT /api/v1/chutes/captive-portal/config
Content-Type: application/json

{
  "net": {

```

(continues on next page)

(continued from previous page)

```
    "wifi": {
      "dhcp": {
        "lease": "1h",
        "limit": 250,
        "start": 3
      },
      "intfName": "wlan0",
      "options": {
        "isolate": True
      },
      "ssid": "Better Free WiFi",
      "type": "wifi"
    }
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "change_id": 1
}
```

set_ssid (*request, chute, network*)

Change the configured SSID for the chute network.

The change will not persist after a reboot. If a persistent change is desired, you should update the chute configuration instead.

Example request:

```
PUT /api/v1/chutes/captive-portal/networks/wifi/ssid
Content-Type: application/json

{
  "ssid": "Best Free WiFi"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "OK"
}
```

start_chute (*request, chute*)**stop_chute** (*request, chute*)**update_chute** (*request, chute*)

```
class ChuteCacheEncoder (skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, encoding='utf-8', default=None)
Bases: json.encoder.JSONEncoder
```

JSON encoder for chute cache dictionary.

The chute cache can contain arbitrary objects, some of which may not be JSON-serializable. This encoder returns handles unserializable objects by returning the *repr* string.

default (*o*)

Implement this method in a subclass such that it returns a serializable object for *o*, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

```
class UpdateEncoder(skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True,
                    sort_keys=False, indent=None, separators=None, encoding='utf-8',
                    default=None)
```

Bases: `json.encoder.JSONEncoder`

default (*o*)

Implement this method in a subclass such that it returns a serializable object for *o*, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

extract_tarred_chute (*data*)

tarfile_is_safe (*tar*)

Check the names of files in the archive for safety.

Returns True if all paths are relative and safe or False if any of the paths are absolute (leading slash) or try to access parent directories (leading `..`).

paradrop.backend.config_api module

This module exposes device configuration.

Endpoints for these functions can be found under `/api/v1/config`.

class ConfigApi (*update_manager, update_fetcher*)

Bases: `object`

Configuration API.

This class handles HTTP API calls related to router configuration.

factory_reset (***kwargs*)

Initiate the factory reset process.

get_hostconfig (*request*)

Get the device's current host configuration.

Example request:

```
GET /api/v1/config/hostconfig
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

For a complete example, please see the Host Configuration section.

get_pdid (*request*)

Get the device's current ParaDrop ID. This is the identifier assigned by the cloud controller.

Example request:

```
GET /api/v1/config/pdid
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  pdid: "5890e1e5ab7e317e6c6e049f"
}
```

get_provision (*request*)

Get the provision status of the device.

new_config (*request*)

Generate a new node configuration based on the hardware.

Example request:

```
GET /api/v1/config/new_config
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

For a complete example, please see the Host Configuration section.

pdconf (*request*)

Get configuration sections from pdconf.

This returns a list of configuration sections and whether they were successfully applied. This is intended for debugging purposes.

pdconf_reload (*request*)

Trigger pdconf to reload UCI configuration files.

Trigger pdconf to reload UCI configuration files and return the status. This function is intended for low-level debugging of the paradrop pdconf module.

provision (*request*)

Provision the device with credentials from a cloud controller.

routes

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar **_url_map**: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar **_endpoints**: A C{dict} mapping endpoint names to handler functions.

sshKeys (*request*, *user*)

Manage list of authorized keys for SSH access.

start_update (*request*)

update_hostconfig (*request*)

Replace the device's host configuration.

Example request:

```
PUT /api/v1/config/hostconfig
Content-Type: application/json

{
  "firewall": {
    "defaults": {
      "forward": "ACCEPT",
      "input": "ACCEPT",
      "output": "ACCEPT"
    }
  },
  ...
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  change_id: 1
}
```

For a complete example, please see the Host Configuration section.

paradrop.backend.cors module

Write the CROSS-ORIGIN RESOURCE SHARING headers required Reference: <http://msoulier.wordpress.com/2010/06/05/cross-origin-requests-in-twisted/>

config_cors (*request*)

paradrop.backend.http_server module

The HTTP server to serve local portal and provide RESTful APIs

class **HttpServer** (*update_manager, update_fetcher, airshark_manager, portal_dir=None*)

Bases: **object**

airshark_analyzer (*request, *args, **kwargs*)

airshark_spectrum (*request, *args, **kwargs*)

api_airshark (*request, *args, **kwargs*)

api_audio (*request*)

api_auth (*request*)

api_changes (*request, *args, **kwargs*)

api_chute (*request, *args, **kwargs*)

api_configuration (*request, *args, **kwargs*)

api_information (*request, *args, **kwargs*)

api_password (*request, *args, **kwargs*)

app

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar _url_map: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar _endpoints: A C{dict} mapping endpoint names to handler functions.

change_stream (*request, *args, **kwargs*)

chute_logs (*request, *args, **kwargs*)

home (*request, *args, **kwargs*)

logs (*request, *args, **kwargs*)

paradrop_logs (*request, *args, **kwargs*)

snapt (*request, *args, **kwargs*)

status (*request, *args, **kwargs*)

annotate_routes (*router, prefix*)

Annotate klein routes for compatibility with autoflask generator.

setup_http_server (*http_server, host, port*)

paradrop.backend.information_api module

Provide information of the router, e.g. board version, CPU information, memory size, disk size.

Endpoints for these functions can be found under `/api/v1/info`.

class InformationApi

get_environment (*request*)

Get environment variables.

Returns a dictionary containing the environment variables passed to the Paradrop daemon. This is useful for development and debugging purposes (e.g. see how PATH is set on Paradrop when running in different contexts).

Example request:

```
GET /api/v1/info/environment
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "LANG": "C.UTF-8",
  "SNAP_REVISION": "x73",
  "SNAP_COMMON": "/var/snap/paradrop-daemon/common",
  "XDG_RUNTIME_DIR": "/run/user/0/snap.paradrop-daemon",
  "SNAP_USER_COMMON": "/root/snap/paradrop-daemon/common",
  "SNAP_LIBRARY_PATH": "/var/lib/snapd/lib/gl:/var/lib/snapd/void",
  "SNAP_NAME": "paradrop-daemon",
  "PWD": "/var/snap/paradrop-daemon/x73",
  "PATH": "/snap/paradrop-daemon/x73/usr/sbin:/snap/paradrop-daemon/x73/usr/
↪bin:/snap/paradrop-daemon/x73/sbin:/snap/paradrop-daemon/x73/bin:/usr/local/
↪sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
↪games",
  "SNAP": "/snap/paradrop-daemon/x73",
  "SNAP_DATA": "/var/snap/paradrop-daemon/x73",
  "SNAP_VERSION": "0.9.2",
  "SNAP_ARCH": "amd64",
  "SNAP_USER_DATA": "/root/snap/paradrop-daemon/x73",
  "TMPDIR": "/tmp",
  "HOME": "/root/snap/paradrop-daemon/x73",
  "SNAP_REEXEC": "",
  "LD_LIBRARY_PATH": "/var/lib/snapd/lib/gl:/var/lib/snapd/void:/snap/
↪paradrop-daemon/x73/usr/lib/x86_64-linux-gnu::/snap/paradrop-daemon/x73/
↪lib:/snap/paradrop-daemon/x73/usr/lib:/snap/paradrop-daemon/x73/lib/x86_64-
↪linux-gnu:/snap/paradrop-daemon/x73/usr/lib/x86_64-linux-gnu",
  "TMPDIR": "/tmp"
  ...
}
```

get_features (*request*)

Get features supported by the host.

This is a list of strings specifying features supported by the daemon.

Explanation of feature strings:

hostapd-control The daemon supports the hostapd control interface and provides a websocket channel for accessing it.

Example request:

```
GET /api/v1/info/features
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  "hostapd-control"
]
```

get_telemetry (*request*)

Get a telemetry report.

This contains information about resource utilization by chute and system totals. This endpoint returns the same data that we periodically send to the controller if telemetry is enabled.

hardware_info (*request*)

Get information about the hardware platform.

Example request:

```
GET /api/v1/info/hardware
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "wifi": [
    {
      "slot": "pci/0000:04:00.0",
      "vendorId": "0x168c",
      "macAddr": "04:f0:21:2f:b7:c1",
      "id": "pci-wifi-0",
      "deviceId": "0x003c"
    },
    {
      "slot": "pci/0000:06:00.0",
      "vendorId": "0x168c",
      "macAddr": "04:f0:21:0f:78:28",
      "id": "pci-wifi-1",
      "deviceId": "0x002a"
    }
  ],
  "memory": 2065195008,
  "vendor": "PC Engines",
}
```

(continues on next page)

(continued from previous page)

```
"board": "APU 1.0",
"cpu": "x86_64"
}
```

routes

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar _url_map: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar _endpoints: A C{dict} mapping endpoint names to handler functions.

software_info(request)

Get information about the operating system.

Returns a dictionary containing information the BIOS version, OS version, kernel version, Paradrop version, and system uptime.

Example request:

```
GET /api/v1/info/software
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "biosVersion": "SageBios_PCEngines_APU-45",
  "biosDate": "04/05/2014",
  "uptime": 15351,
  "kernelVersion": "Linux-4.4.0-101-generic",
  "pdVersion": "0.9.2",
  "biosVendor": "coreboot",
  "osVersion": "Ubuntu 4.4.0-101.124-generic 4.4.95"
}
```

paradrop.backend.log_sockjs module

class LogSockJSFactory(chutename)

Bases: twisted.internet.protocol.Factory

buildProtocol(addr)

class LogSockJSProtocol(factory)

Bases: twisted.internet.protocol.Protocol

check_log()

connectionLost(reason)

connectionMade()

paradrop.backend.password_api module

class PasswordApi(password_manager)

Bases: object

For now, we only support set/reset password for the default user: 'paradrop'

`change (request)`

`clear (request)`

routes

L{Klein} is an object which is responsible for maintaining the routing configuration of our application.

@ivar _url_map: A C{werkzeug.routing.Map} object which will be used for routing resolution.

@ivar _endpoints: A C{dict} mapping endpoint names to handler functions.

paradrop.backend.password_manager module

class PasswordManager

Bases: `object`

DEFAULT_PASSWORD = ''

DEFAULT_USER_NAME = 'paradrop'

add_user (user_name, password)

change_password (user_name, newPassword)

remove_user (user_name)

reset ()

verify_password (user_name, password)

paradrop.backend.snapd_resource module

class SnapdResource

Bases: `twisted.web.resource.Resource`

Expose the snapd API by forwarding requests.

<https://github.com/snapcore/snapd/wiki/REST-API>

do_snapd_request (request)

Forward the API request to snapd.

isLeaf = True

render (request)

Fulfill requests by forwarding them to snapd.

We use a synchronous implementation of HTTP over Unix sockets, so we do the request in a worker thread and have it call request.finish.

paradrop.backend.status_sockjs module

class StatusSockJSFactory (system_status)

Bases: `twisted.internet.protocol.Factory`

buildProtocol (addr)

class StatusSockJSProtocol (factory)

Bases: `twisted.internet.protocol.Protocol`

connectionLost (reason)

```

connectionMade ()
dataReceived (data)

```

Module contents

12.1.3 paradrop.base package

Submodules

paradrop.base.cxbr module

Wamp utility methods.

```

class BaseClientFactory (factory, *args, **kwargs)
    Bases: autobahn.twisted.websocket.WampWebSocketClientFactory, twisted.
           internet.protocol.ReconnectingClientFactory

    clientConnectionFailed (connector, reason)
        Called when a connection has failed to connect.

        It may be useful to call connector.connect() - this will reconnect.

        @type reason: L{twisted.python.failure.Failure}

    clientConnectionLost (connector, reason)
        Called when an established connection is lost.

        It may be useful to call connector.connect() - this will reconnect.

        @type reason: L{twisted.python.failure.Failure}

    initialDelay = 1
    maxDelay = 60

class BaseSession (config=None)
    Bases: autobahn.twisted.wamp.ApplicationSession

    Temporary base class for crossbar implementation

    call (procedure, *args, **kwargs)
        Implements autobahn.wamp.interfaces.ICaller.call()

    leave ()
        Implements autobahn.wamp.interfaces.ISession.leave()

    onJoin (**kwargs)
        Implements autobahn.wamp.interfaces.ISession.onJoin()

    publish (topic, *args, **kwargs)
        Implements autobahn.wamp.interfaces.IPublisher.publish()

    register (endpoint, procedure=None, options=None)
        Implements autobahn.wamp.interfaces.ICallee.register()

    classmethod start (address, pdid, realm='paradrop', start_reactor=False, debug=False, ex-
                      tra=None, reconnect=True)
        Creates a new instance of this session and attaches it to the router at the given address and realm.

        reconnect: The session will attempt to reconnect on connection failure and continue trying indefi-
            nitely.

```

```
stockCall (procedure, *args, **kwargs)
stockPublish (topic, *args, **kwargs)
stockRegister (endpoint, procedure=None, options=None)
stockSubscribe (handler, topic=None, options=None)
subscribe (handler, topic=None, options=None)
    Implements autobahn.wamp.interfaces.ISubscriber.subscribe()
class BaseSessionFactory (config, deferred=None)
    Bases: autobahn.twisted.wamp.ApplicationSessionFactory
```

paradrop.base.exceptions module

Exceptions and their subclasses

TODO: Distill these down and make a heirarchy.

```
exception AuthenticationError
    Bases: paradrop.base.exceptions.PdServerException
exception ChuteNotFound
    Bases: paradrop.base.exceptions.ParadropException
exception ChuteNotRunning
    Bases: paradrop.base.exceptions.ParadropException
exception DeviceNotFoundExpection
    Bases: paradrop.base.exceptions.ParadropException
exception InterlException
    Bases: exceptions.Exception
exception InvalidCredentials
    Bases: paradrop.base.exceptions.PdServerException
exception ModelNotFound
    Bases: paradrop.base.exceptions.PdServerException
exception ParadropException
    Bases: exceptions.Exception
exception PdServerException
    Bases: exceptions.Exception
exception PdidError
    Bases: paradrop.base.exceptions.PdServerException
exception PdidExclusionError
    Bases: paradrop.base.exceptions.PdServerException
```

paradrop.base.nexus module

Stateful, singleton, paradrop daemon command center. See docstring for NexusBase class for information on settings.

SETTINGS QUICK REFERENCE: # assuming the following import from paradrop.base import nexus

```
nexus.core.info.version
nexus.core.info.pdid
```

class AttrWrapper

Bases: `object`

Simple attr interceptor to make accessing settings simple.

Stores values in an internal dict called contents.

Does not allow modification once `_lock()` is called. Respect it.

Once you've filled it up with the appropriate initial values, set `onChange` to assign

setOnChange (*func*)

class NexusBase (*stealStdio=True, printToConsole=True*)

Bases: `object`

Resolving these values to their final forms: 1 - module imported, initial values assigned(as written below) 2 - class is instantiated, passed settings to replace values 3 - instance chooses appropriate values based on current state(production or local)

Each category has its own method for initialization here (see: `resolveNetwork`, `resolvePaths`)

PDID = None

VERSION = 1

connect (***kwargs*)

Takes the given session class and attempts to connect to the crossbar fabric.

If an existing session is connected, it is cleanly closed.

getKey (*name*)

Returns the given key or None

onInfoChange (*key, value*)

Called when an internal setting is changed. Trigger a save automatically.

onStart ()

onStop ()

provision (*pdid, pdserver='https://paradrop.org', wampRouter='ws://paradrop.org:9086/ws'*)

provisioned ()

Checks if this[whatever] appears to be provisioned or not

save ()

Ehh. Ideally this should happen asynchronously.

saveKey (*key, name*)

Save the key with the given name. Overwrites by default

createDefaultInfo (*path*)

loadYaml (*path*)

Return dict from YAML found at path

resolveInfo (*nexus, path*)

Given a path to the config file, load its contents and assign it to the config file as appropriate.

validateInfo (*contents*)

Error checking on the read YAML file. This is a temporary method.

:

param contents: the read - in yaml to check

```

:
    type contents: dict.

:
    returns: True if valid, else false

```

writeYaml (*contents*, *path*)
Overwrites content with YAML representation at given path

paradrop.base.output module

Output mapping, capture, storage, and display.

Some of the methods and choice here may seem strange – they are meant to keep this file in

class BaseOutput (*logType*)

Bases: *object*

Base output type class.

This class and its subclasses are registered with an attribute on the global ‘out’ function and is responsible for formatting the given output stream and returning it as a “log structure” (which is a dict.)

For example: out.info(“Text”, anObject)

requires a custom object to figure out what to do with anObject where the default case will simply parse the string with an appropriate color.

Objects are required to output a dict that minimally contains the keys message and type.

formatOutput (*logDict*)

Convert a logdict into a custom formatted, human readable version suitable for printing to console.

class ExceptionOutput (*logType*)

Bases: *paradrop.base.output.BaseOutput*

Handle vanilla exceptions passed directly to us using out.exception

class Level

Bases: *enum.Enum*

ERR = 6

FATAL = 8

HEADER = 1

INFO = 3

PERF = 4

SECURITY = 7

USAGE = 9

VERBOSE = 2

WARN = 5

class Output (***kwargs*)

Class that allows stdout/stderr trickery. By default the paradrop object will contain an @out variable (defined below) and it will contain 2 members of “err” and “fatal”.

Each attribute of this class should be a function which points to a class that inherits IOutput(). We call these functions “output streams”.

The way this Output class is setup is that you pass it a series of kwargs like (stuff=OutputClass()). Then at any point in your program you can call “paradrop.out.stuff(‘This is a string

’”).

This way we can easily support different levels of verbosity without the need to use some kind of bitmask or anything else. On-the-fly output creation is no longer supported due to the metadata and special processing added. It is still possible, but not implemented.

This is done by the `__getattr__` function below, basically in `__init__` we set any attributes you pass as args, and anything else not defined gets sent to `__getattr__` so that it doesn’t error out.

endLogging()

Ask the printing thread to flush and end, then return.

getLogsSince(target, purge=False)

Reads all logs and returns their contents. The current log file is not touched. Removes old log files if ‘purge’ is set (though this is a topic for debate...)

The server will be most interested in this call, but it needs to register for new logs first, else there’s a good chance to see duplicates.

NOTE: don’t open all log files, check to open only the ones that might be relevant. This is certainly a bug and can cause memory issues.

Parameters

- **target** (*float*.) – seconds since the GMT epoch. Method returns logs that have times-tamps later than this.
- **purge** (*bool*.) – deletes the old log files (except today’s) if set

Returns a list of dictionaries containing log information. Not ordered.

handlePrint(logDict)

All printing objects return their messages. These messages are routed to this method for handling.

Send the messages to the printer. Optionally display the messages. Decorate the print messages with metadata.

Parameters **logDict** – a dictionary representing this log item. Must contain keys

message and type. :type logDict: dict.

logToConsole(newStatus)

messageToString(message)

Converts message dicts to a format suitable for printing based on the conversion rules laid out in in that class’s implementation.

Parameters **message** (*dict*.) – the dict to convert to string

Returns str

startLogging(filePath=None, stealStdio=False, printToConsole=True)

Begin logging. The output class is ready to go out of the box, but in order to prevent mere imports from stealing stdio or console logging to vanish these must be manually turned on.

Parameters

- **filePath** (*str*) – if provided, begin logging to the given directory. If not provided, do not write out logs.
- **stealStdio** (*bool*.) – choose to intercept stdio (including vanilla print statements) or allow it to passthrough
- **printToConsole** (*bool*.) – output the results of all logging to the console. This is primarily a performance consideration when running in production

stealStdio (*newStatus*)

class OutputRedirect (*output, contentAppearedCallback, logType*)

Bases: `object`

Intercepts passed output object (either stdout and stderr), calling the provided callback method when input appears.

Retains the original mappings so writing can still happen. Performs no formatting.

flush ()

trueWrite (*contents*)

Someone really does want to output

write (*contents*)

Intercept output to the assigned target and callback with it. The true output is returned with the callback so the delegate can differentiate between captured outputs in the case when two redirecters are active.

class PrintLogThread (*path, queue, name*)

Bases: `threading.Thread`

All file printing access from one thread.

Receives information when its placed on the passed queue. Called from one location: `Output.handlePrint`.

Does not close the file: this happens in `Output.endLogging`. This simplifies the operation of this class, since it only has to concern itself with the queue.

The path must exist before `DailyLog` runs for the first time.

run ()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

class TwistedException (*logType*)

Bases: `paradrop.base.output.BaseOutput`

class TwistedOutput (*logType*)

Bases: `paradrop.base.output.BaseOutput`

blacklist = ['Starting factory', 'Stopping factory', 'Log opened']

parseLogPrefix (*tb*)

Takes a traceback returned by 'extract_tb' and returns the package, module, and line number

silentLogPrefix (*stepsUp*)

`logPrefix v2`– gets caller information silently (without caller intervention) The single parameter reflects how far up the stack to go to find the caller and depends how deep the direct caller to this method is wrt to the target caller

NOTE: Some calls cannot be silently prefixed (getting into the twisted code is a great example)

Parameters `stepsUp` – the number of steps to move up the stack for the caller

paradrop.base.pdutils module

`lib.utils.output`. Helper for formatting output from Paradrop.

class `Timer` (*key*=", *verbose*=True)

Bases: `object`

A timer object for simple benchmarking.

Usage:

with `Timer(key='Name of this test')` **as** `t`: `do.someCode(thatTakes=aWhile)`

Once the code finishes executing the time is output.

check (*pkt*, *pktType*, *keyMatches*=None, ***valMatches*)

This function takes an object that was expected to come from a packet (after it has been JSONized) and compares it against the arg requirements so you don't have to have 10 if() statements to look for keys in a dict, etc..

Args: `@pkt` : object to look at `@pktType` : object type expected (dict, list, etc..) `@keyMatches` : a list of minimum keys found in parent level of dict, expected to be an array `@valMatches` : a dict of key:value pairs expected to be found in the parent level of dict

the value can be data (like 5) OR a type (like this value must be a @list@).

Returns: None if everything matches, otherwise it returns a string as to why it failed.

convertUnicode (*elem*)

Converts all unicode strings back into UTF-8 (str) so everything works. Call this function like:

```
json.loads(s, object_hook=convertUnicode)
```

class `dict2obj` (*aDict*=None, ***kwargs*)

Bases: `object`

explode (*pkt*, **args*)

This function takes a dict object and explodes it into the tuple requested.

It returns None for any value it doesn't find.

The only error it throws is if args is not defined.

Example: `pkt = { 'a':0, 'b':1 } 0, 1, None = pdcomm.explode(pkt, 'a', 'b', 'c')`

json2str (*j*, *safe*=')

Properly converts and encodes all data related to the JSON object into a string format that can be transmitted through a network and stored properly in a database. Arguments:

`@j` : json to be converted `@safe` : optional, string of chars to pass to `urlencodeMe` that are declared safe (don't encode)

jsonPretty (*j*)

Returns a string of a JSON object in 'pretty print' format fully indented, and sorted.

stimestr (*x*=None)

str2json (*s*)

timedur (*x*)

Print consistent string format of seconds passed. Example: 300 = '5 mins' Example: 86400 = '1 day' Example: 86705 = '1 day, 5 mins, 5 sec'

timeflt ()

timeint ()

timestr (*x=None*)

urlDecodeMe (*elem*)

Converts any values that would cause JSON parsing to fail into URL percent encoding equivalents. This function can be used for any valid JSON type including str, dict, list. Returns:

Same element properly decoded.

urlEncodeMe (*elem, safe=' '*)

Converts any values that would cause JSON parsing to fail into URL percent encoding equivalents. This function can be used for any valid JSON type including str, dict, list. Returns:

Same element properly encoded.

paradrop.base.settings module

This file contains any settings required by ANY and ALL modules of the paradrop system. They are defaulted to some particular value and can be called by any module in the paradrop system with the following code:

```
from paradrop import settings print(settings.STUFF)
```

These settings can be overridden by a KYE:VALUE array

If settings need to be changed, they should be done so by the initialization code (such as pdfcd, pdfc_config, etc. . .)

This is done by calling the following function: settings.updateSettings(settings_array)

loadSettings (*mode='local', slist=[]*)

Take a list of key:value pairs, and replace any setting defined. Also search through the settings module and see if any matching environment variables exist to replace as well.

Parameters **slist** (*array.*) – the list of key:val settings

Returns None

parseValue (*key*)

Attempts to parse the key value, so if the string is 'False' it will parse a boolean false.

Parameters **key** (*string*) – the key to parse

Returns the parsed key.

updatePaths (*configHomeDir, runtimeHomeDir='/var/run/paradrop'*)

Module contents

12.1.4 paradrop.conf.d package

Submodules

paradrop.conf.d.base module

class **ConfigObject** (*name=None*)

Bases: `object`

PRIO_CONFIG_IFACE = 30

PRIO_CONFIG_QDISC = 45

PRIO_CREATE_IFACE = 20

PRIO_CREATE_QDISC = 40

PRIO_CREATE_VLAN = 25

PRIO_IPTABLES_RULE = 37

PRIO_IPTABLES_TOP = 35

PRIO_IPTABLES_ZONE = 36

PRIO_START_DAEMON = 60

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

classmethod build (*manager, source, name, options, comment*)

Build a config object instance from the UCI section.

Arguments: source – file containing this configuration section name – name of the configuration section

If None, a unique name will be generated.

options – dictionary of options loaded from the section comment – comment string or None

copy ()

Make a copy of the config object.

The copy will receive the same name and option values.

dump ()

Return full configuration section as a string.

findByType (*allConfigs, module, typename, where={}*)

Look up sections by type (generator).

where: filter the returned results by checking option values.

classmethod getModule ()

Get the module name (e.g. “dhcp”, “wireless”) for a ConfigObject class.

getName ()

Return section name.

Subclasses that do not have names (anonymous sections) should override this to return some other unique identifier such as an interface name.

getTypeAndName ()

Return tuple (section module, section type, section name).

lookup (*allConfigs, sectionModule, sectionType, sectionName, addDependent=True*)

Look up a section by type and name.

If addDependent is True (default), the current object will be added as a dependent of the found section.

Will raise an exception if the section is not found.

maskable = True

nextId = 0

options = []

optionsMatch (*other*)

Test equality of config sections by comparing option values.

static prioritizeConfigs (*reverse=False*)

Assign priorities to config objects based on the dependency graph.

Priority zero is assigned to all configs with no dependencies.

`priority(config1) > priority(config2)` means `config1` should be applied later than `config2`, and `config1` should be reverted earlier than `config2`. For configs with the same priority value, it is presumed that order does not matter.

If `reverse` is `True`, the priorities are made negative so that traversing in increasing order gives the proper order for reverting.

Returns a list of tuples (`priority`, `config`). This format is suitable for `heapq`.

removeFromParents ()

Remove this section from being tracked by its parents.

Call this before discarding a configuration section so that later on, if the parent is updated, it doesn't try to update non-existent children.

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (`priority`, `Command`) tuples.

setup ()

Finish object initialization.

This is called after the config object is initialized with all of its options values filled in. Override to do some preparation work before we start generating commands.

typename = `None`

updateApply (*new, allConfigs*)

Return a list of commands to update to new configuration.

Implementing this is optional for subclasses. The default behavior is to call `apply`.

Returns a list of (`priority`, `Command`) tuples.

updateRevert (*new, allConfigs*)

Return a list of commands to (partially) revert the configuration.

The implementation can be selective about what it reverts (e.g. do not delete an interface if we are only updating its IP address). The default behavior is to call `revert`.

Returns a list of (`priority`, `Command`) tuples.

class ConfigOption (*name, type=<type 'str'>, required=False, default=None*)

Bases: `object`

default

name

required

type

interpretBoolean (*s*)

Interpret string as a boolean value.

"0" and "False" are interpreted as `False`. All other strings result in `True`. Technically, only "0" and "1" values should be seen in UCI files. However, because of some string conversions in Python, we may encounter "False".

paradrop.conf.d.client module

reload (*path*)

Reload file(s) specified by path.

This function blocks until the request completes. On completion it returns a status string, which is a JSON list of loaded configuration sections with a 'success' field. For critical errors it will return None.

reloadAll ()

Reload all files from the system configuration directory.

This function blocks until the request completes. On completion it returns a status string, which is a JSON list of loaded configuration sections with a 'success' field. For critical errors it will return None.

systemStatus ()

Return system status string from pdconf.

waitSystemUp ()

Wait for the configuration daemon to finish its first load.

This function blocks until the request completes. On completion it returns a status string, which is a JSON list of loaded configuration sections with a 'success' field. For critical errors it will return None.

paradrop.conf.d.command module

class Command (*command, parent=None, ignoreFailure=False*)

Bases: `object`

execute ()

success ()

Returns True if the command was successfully executed.

class CommandList

Bases: `list`

append (*priority, command*)

L.append(object) – append object to end

commands ()

Iterate over commands in order by priority.

Commands are first sorted by assigned priority. Within each priority level, the order in which they were added is maintained.

class ErrorCommand (*error, parent=None*)

Bases: `paradrop.conf.d.command.Command`

Special command object that indicates an error occurred.

execute ()

success ()

Returns True if the command was successfully executed.

class KillCommand (*pid, parent=None*)

Bases: `paradrop.conf.d.command.Command`

Special command object for killing a process

execute ()

getPid ()

kill (*pid*, *kill_signal*=4, *timeout*=8)

Kill a child process and wait with timeout.

1. Send a SIGTERM signal to the process.
2. Wait up to *kill_signal* seconds for the process to exit.
3. If process is still running, send a SIGKILL signal.
4. Wait up to *timeout* seconds (cumulative with *kill_signal*) for the process to exit.

Returns True if the process exited before *timeout* seconds elapsed.

paradrop.conf.dhcp module

class ConfigDhcp (*name*=None)

Bases: *paradrop.conf.d.base.ConfigObject*

options = [ConfigOption(name='interface', type=<type 'str'>, required=True, default=None),

typename = 'dhcp'

class ConfigDnsmasq (*name*=None)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

options = [ConfigOption(name='authoritative', type=<type 'bool'>, required=False, default=False),

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

typename = 'dnsmasq'

class ConfigDomain (*name*=None)

Bases: *paradrop.conf.d.base.ConfigObject*

getName ()

Return section name.

Subclasses that do not have names (anonymous sections) should override this to return some other unique identifier such as an interface name.

options = [ConfigOption(name='name', type=<type 'str'>, required=False, default=None),

typename = 'domain'

paradrop.conf.firewall module

class ConfigDefaults (*name*=None)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

getName ()

Return section name.

Subclasses that do not have names (anonymous sections) should override this to return some other unique identifier such as an interface name.

get_iptables ()

Get the list of iptables commands to use (iptables / ip6tables).

options = [ConfigOption(name='input', type=<type 'str'>, required=False, default='ACCEPT')

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

typename = 'defaults'

updateApply (*new*, *allConfigs*)

Return a list of commands to update to new configuration.

Implementing this is optional for subclasses. The default behavior is to call apply.

Returns a list of (priority, Command) tuples.

updateRevert (*new*, *allConfigs*)

Return a list of commands to (partially) revert the configuration.

The implementation can be selective about what it reverts (e.g. do not delete an interface if we are only updating its IP address). The default behavior is to call revert.

Returns a list of (priority, Command) tuples.

class ConfigForwarding (*name=None*)

Bases: [paradrop.conf.d.base.ConfigObject](#)

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

options = [ConfigOption(name='src', type=<type 'str'>, required=True, default=None), C

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

typename = 'forwarding'

class ConfigRedirect (*name=None*)

Bases: [paradrop.conf.d.base.ConfigObject](#)

ANY_PROTO = set(['none', None, 'any'])

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

options = [ConfigOption(name='src', type=<type 'str'>, required=False, default=None),

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

typename = 'redirect'

class ConfigRule (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

get_iptables ()

Get the list of iptables commands to use (iptables / ip6tables).

options = [ConfigOption(name='name', type=<type 'str'>, required=False, default=None),

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

typename = 'rule'

class ConfigZone (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

get_iptables ()

Get the list of iptables commands to use (iptables / ip6tables).

options = [ConfigOption(name='name', type=<type 'str'>, required=True, default=None),

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

setup ()

Finish object initialization.

This is called after the config object is initialized with all of its options values filled in. Override to do some preparation work before we start generating commands.

```
typename = 'zone'
```

paradrop.conf.d.main module

This module listens for messages and triggers reloading of configuration files. This module is the service side of the implementation. If you want to issue reload commands to the service, see the client.py file instead.

listen (*configManager*)

run_thread (*execute=True*)

Start pdconfd service as a thread.

This function schedules pdconfd to run as a thread and returns immediately.

paradrop.conf.d.manager module

class ConfigManager (*writeDir, execCommands=True*)

Bases: `object`

changingSet (*files*)

Return the sections from the current configuration that may have changed.

This checks which sections from the current configuration came from files in the given file list. These are sections that may be changed or removed when we reload the files.

execute (*commands*)

Execute commands.

Takes a CommandList object.

findMatchingConfig (*config, byName=False*)

Check the current config for an identical section.

Returns the matching object or None.

getPreviousCommands ()

Get the most recent command list.

loadConfig (*search=None, execute=True*)

Load configuration files and apply changes to the system.

We process the configuration files in sections. Each section corresponds to an interface, firewall rule, DHCP server instance, etc. Each time we reload configuration files after the initial time, we check for changes against the current configuration. Here is the decision tree for handling differences in the newly loaded configuration vs. the existing configuration:

Section exists in current config (by type and name)?

- No -> Add section, apply changes, and stop.
- Yes -> Continue.

Section is identical to the one in the current config (by option values)?

- **No -> Revert current section, mark any affected dependents,** add new section, apply changes, and stop.
- Yes -> Continue.

Section has not changed but one of its dependencies has?

- No -> Stop.
- Yes -> **Revert current section, mark any affected dependents**, add new section, apply changes, and stop.

readConfig (*files*)

Load configuration files and return configuration objects.

This method only loads the configuration files without making any changes to the system and returns configuration objects as a generator.

statusString ()

Return a JSON string representing status of the system.

The format will be a list of dictionaries. Each dictionary corresponds to a configuration block and contains at the following fields.

type: interface, wifi-device, etc. name: name of the section (may be autogenerated for some configs)
comment: comment from the configuration file or None success: True if all setup commands succeeded

unload (*execute=True*)

waitSystemUp ()

Wait for the first load to complete and return system status string.

findConfigFiles (*search=None*)

Look for and return a list of configuration files.

The behavior depends on whether the search argument is a file, a directory, or None.

If search is None, return a list of files in the system config directory. If search is a file name (not a path), look for it in the working directory first, and the system directory second. If search is a full path to a file, and it exists, then return that file. If search is a directory, return the files in that directory.

paradrop.conf.network module

class ConfigInterface (*name=None*)

Bases: *paradrop.conf.base.ConfigObject*

DEV_PLUS_VID = *<_sre.SRE_Pattern object>*

addToBridge (*ifname*)

Generate commands to add ifname to bridge.

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

maskable = **False**

options = [**ConfigOption**(name='proto', type=<type 'str'>, required=True, default=None),

removeFromBridge (*ifname*)

Generate commands to add ifname to bridge.

revert (*allConfigs*)

Return a list of commands to revert this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

setup ()

Finish object initialization.

This is called after the config object is initialized will all of its options values filled in. Override to do some preparation work before we start generating commands.

typename = 'interface'

updateApply (*new*, *allConfigs*)

Return a list of commands to update to new configuration.

Implementing this is optional for subclasses. The default behavior is to call apply.

Returns a list of (priority, Command) tuples.

updateRevert (*new*, *allConfigs*)

Return a list of commands to (partially) revert the configuration.

The implementation can be selective about what it reverts (e.g. do not delete an interface if we are only updating its IP address). The default behavior is to call revert.

Returns a list of (priority, Command) tuples.

paradrop.conf.d.qos module

class ConfigClass (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

options = [ConfigOption(name='packetize', type=<type 'int'>, required=False, default=)

typename = 'class'

class ConfigClassgroup (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

get_class_id (*class_name*)

Get ID for a traffic class in this group.

Returns None if the class is not a member of the group.

options = [ConfigOption(name='classes', type=<type 'str'>, required=True, default=None)

setup ()

Finish object initialization.

This is called after the config object is initialized will all of its options values filled in. Override to do some preparation work before we start generating commands.

typename = 'classgroup'

class ConfigClassify (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)

Return a list of commands to apply this configuration.

Most subclasses will need to implement this function.

Returns a list of (priority, Command) tuples.

make_iptables_cmd (*action*, *ifname*, *class_id*)

options = [ConfigOption(name='target', type=<type 'str'>, required=True, default=None)

revert (*allConfigs*)
 Return a list of commands to revert this configuration.
 Most subclasses will need to implement this function.
 Returns a list of (priority, Command) tuples.

typename = 'classify'

class ConfigInterface (*name=None*)
 Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)
 Return a list of commands to apply this configuration.
 Most subclasses will need to implement this function.
 Returns a list of (priority, Command) tuples.

options = [ConfigOption(name='enabled', type=<type 'bool'>, required=True, default=None)

revert (*allConfigs*)
 Return a list of commands to revert this configuration.
 Most subclasses will need to implement this function.
 Returns a list of (priority, Command) tuples.

typename = 'interface'

compute_hfsc_params (*classes, capacity*)

paradrop.conf.d.wireless module

class ConfGenerator

Bases: *object*

writeHeader (*output*)

writeOptions (*options, output, title=None*)

class ConfigWifiDevice (*name=None*)

Bases: *paradrop.conf.d.base.ConfigObject*

apply (*allConfigs*)
 Return a list of commands to apply this configuration.
 Most subclasses will need to implement this function.
 Returns a list of (priority, Command) tuples.

detectPrimaryInterface ()

Find the primary network interface associated with this Wi-Fi device.

By primary we mean the first interface (e.g. wlan0 or wlan1) that exists at system startup before any *interface add* commands. We will use the primary interface first, and create additional virtual interfaces after that.

That seems overly complicated, but it is required in cases where the Wi-Fi device does not support virtual interfaces.

Returns interface name or None.

nextInterfaceName ()

Get the next available interface name.

```

options = [ConfigOption(name='type', type=<type 'str'>, required=True, default=None),
releaseInterfaceName (ifname)
    Mark an interface name as no longer used.

revert (allConfigs)
    Return a list of commands to revert this configuration.

    Most subclasses will need to implement this function.

    Returns a list of (priority, Command) tuples.

setup ()
    Finish object initialization.

    This is called after the config object is initialized will all of its options values filled in. Override to do some
    preparation work before we start generating commands.

typename = 'wifi-device'
class ConfigWifiIface (name=None)
    Bases: paradrop.conf.d.base.ConfigObject

    apply (allConfigs)
        Return a list of commands to apply this configuration.

        Most subclasses will need to implement this function.

        Returns a list of (priority, Command) tuples.

    getIfname (device, interface)
        Returns the name to be used by this WiFi interface, e.g. as seen by ifconfig.

        This comes from the "ifname" option if it is set. Otherwise, we use the interface name of the associated
        network.

    getName ()
        Return a unique and consistent identifier for the section.

        If ifname is set, then that is a good choice for the name because interface names need to be unique on the
        system.

        If ifname is not set, then we use the combined string device:network. The assumption is that no one will
        put multiple APs on the same device and same network, or if they do, (e.g. multiple APs on the br-lan
        bridge), then they will configure the ifname to be unique.

    getRandomMAC ()
        Generate a random MAC address.

        Returns a string "02:xx:xx:xx:xx:xx". The first byte is 02, which indicates a locally administered address.

    makeHostapdConf (wifiDevice, interface)

    makeWpaSupplicantConf (wifiDevice, interface)

    options = [ConfigOption(name='device', type=<type 'str'>, required=True, default=None)

    revert (allConfigs)
        Return a list of commands to revert this configuration.

        Most subclasses will need to implement this function.

        Returns a list of (priority, Command) tuples.

    typename = 'wifi-iface'

```

updateApply (*new, allConfigs*)

Return a list of commands to update to new configuration.

Implementing this is optional for subclasses. The default behavior is to call apply.

Returns a list of (priority, Command) tuples.

updateRevert (*new, allConfigs*)

Return a list of commands to (partially) revert the configuration.

The implementation can be selective about what it reverts (e.g. do not delete an interface if we are only updating its IP address). The default behavior is to call revert.

Returns a list of (priority, Command) tuples.

class HostapdConfGenerator (*wifiIface, wifiDevice, interface*)

Bases: *paradrop.conf.d.wireless.ConfGenerator*

generate (*path*)

get11acOptions ()

get11nOptions ()

get11rOptions ()

Get options related to 802.11r (fast BSS transition).

getMainOptions ()

getRadiusOptions ()

getSecurityOptions ()

readMode (*device*)

Determine HT/VHT mode if applicable.

writeHeader (*output*)

class WpaSupplicantConfGenerator (*wifiIface, wifiDevice, interface*)

Bases: *paradrop.conf.d.wireless.ConfGenerator*

generate (*path*)

getMainOptions ()

writeHeader (*output*)

getPhyFromMAC (*mac*)

getPhyMACAddress (*phy*)

get_cipher_list (*encryption_mode*)

Get list of ciphers from encryption mode.

Example: `get_cipher_list("psk2+tkip+aes")` -> ["TKIP", "CCMP"]

isHexString (*data*)

Test if a string contains only hex digits.

Module contents

12.1.5 paradrop.core package

Subpackages

paradrop.core.agent package

Submodules

paradrop.core.agent.http module

class CurlRequestDriver

Bases: `paradrop.core.agent.http.HTTPRequestDriver`

code_pattern = `<_sre.SRE_Pattern object>`

curl = `<MagicMock name='mock.Curl()' id='140696307865488'>`

header_pattern = `<_sre.SRE_Pattern object>`

lock = `<twisted.internet.defer.DeferredLock object>`

receive (*ignore*)

Receive response from curl and convert it to a response object.

receiveHeaders (*header_line*)

request (*method, url, body=None*)

class HTTPRequestDriver

Bases: `object`

request (*method, url, body*)

setHeader (*key, value*)

class HTTPResponse (*data=None*)

Bases: `object`

class JSONReceiver (*response, finished*)

Bases: `twisted.internet.protocol.Protocol`

JSON Receiver

A JSONReceiver object can be used with the twisted HTTP client to receive data from a request and provide it to a callback function when complete.

Example (response came from an HTTP request): `finished = Deferred() response.deliverBody(JSONReceiver(finished)) finished.addCallback(func_that_takes_result)`

Some error conditions will result in the callback firing with a result of None. The receiver needs to check for this. This seems to occur on 403 errors where the server does not return any data, but twisted just passes us a ResponseDone object the same type as a normal result.

connectionLost (*reason*)

internal: handles connection close events.

dataReceived (*data*)

internal: handles incoming data.

```
class PDServerRequest (path, driver=<class 'paradrop.core.agent.http.TwistedRequestDriver'>, setAuthHeader=True)
```

Bases: `object`

Make an HTTP request to pdserver.

The API is assumed to use application/json for sending and receiving data. Authentication is automatically handled here if the router is provisioned.

We handle missing, invalid, or expired tokens by making the request and detecting a 401 (Unauthorized) response. We request a new token and retry the failed request. We do this at most once and return failure if the second attempt returns anything other than 200 (OK).

PDServerRequest objects are not reusable; create a new one for each request.

URL String Substitutions: router_id -> router id

Example: /routers/{router_id}/states -> /routers/halo06/states

get (***query*)

classmethod **getServerInfo** ()

Return the information needed to send API messages to the server.

This can be used by an external program (e.g. pdinstall).

patch (**ops*)

Expects a list of operations in jsonpatch format (<http://jsonpatch.com/>).

An example operation would be: {'op': 'replace', 'path': '/completed', 'value': True}

post (***data*)

put (***data*)

receiveResponse (*response*)

Intercept the response object, and if it's a 401 authenticate and retry.

request ()

classmethod **resetToken** ()

Reset the auth token, to be called if the router's identity has changed.

token = None

```
class PDServerResponse (response, data=None)
```

Bases: `object`

A PDServerResponse object contains the results of a request to pdserver.

This wraps twisted.web.client.Response (cannot be subclassed) and exposes the same variables in addition to a 'data' variables. The 'data' variable, if not None, is the parsed object from the response body.

```
class TwistedRequestDriver
```

Bases: `paradrop.core.agent.http.HTTPRequestDriver`

pool = <twisted.web.client.HTTPConnectionPool object>

receive (*response*)

Receive response from twisted web client and convert it to a PDServerResponse object.

request (*method, url, body=None*)

sem = <twisted.internet.defer.DeferredSemaphore object>

urlEncodeParams (*data*)

Return data URL-encoded.

This function specifically handles None and boolean values to convert them to JSON-friendly strings (e.g. None -> 'null').

paradrop.core.agent.reporting module

class ReportSender (*model='states', max_retries=None*)

Bases: `object`

increaseDelay ()

send (*report*)

class StateReport

Bases: `object`

toJSON ()

class StateReportBuilder

Bases: `object`

prepare ()

class TelemetryReportBuilder

Bases: `object`

prepare ()

sendStateReport ()

sendTelemetryReport ()

paradrop.core.agent.wamp_session module

The WAMP session of the paradrop daemon

class WampSession (**args, **kwargs*)

Bases: `paradrop.base.cxbr.BaseSession`

onChallenge (*challenge*)

Implements autobahn.wamp.interfaces.ISession.onChallenge ()

onConnect ()

Implements autobahn.wamp.interfaces.ISession.onConnect ()

onDisconnect ()

Implements autobahn.wamp.interfaces.ISession.onDisconnect ()

onJoin (***kwargs*)

Implements autobahn.wamp.interfaces.ISession.onJoin ()

onLeave (*details*)

Implements autobahn.wamp.interfaces.ISession.onLeave ()

classmethod set_update_fetcher (*update_fetcher*)

update (*pdid, data*)

update_fetcher = None

`updatesPending (**kwargs)`

Module contents

paradrop.core.chute package

Submodules

paradrop.core.chute.chute module

class `Chute` (*descriptor*, *strip=None*)

Bases: `object`

Wrapper class for Chute objects.

`CONFIG_FIELDS = set(['environment', 'web', 'net', 'host_config'])`

`STATE_DISABLED = 'disabled'`

`STATE_FROZEN = 'frozen'`

`STATE_INVALID = 'invalid'`

`STATE_RUNNING = 'running'`

`STATE_STOPPED = 'stopped'`

appendCache (*key*, *val*)

Finds the key they requested and appends the val into it, this function assumes the cache object is of list type, if the key hasn't been defined yet then it will set it to an empty list.

delCache (*key*)

Delete the key:val from the `_cache` dict object.

dumpCache ()

Return a string of the contents of this chute's cache. In case of catastrophic failure dump all cache content so we can debug.

getCache (*key*)

Get the val out of the `_cache` dict object, or None if it doesn't exist.

getCacheContents ()

Return the cache dictionary.

getConfiguration ()

Get the chute's configuration object.

getHostConfig ()

Get the chute's `host_config` options for Docker.

Returns an empty dictionary if there is no `host_config` setting.

getWebPort ()

Get the port configured for the chute's web server.

Returns port (int) or None if no port is configured.

isRunning ()

isValid ()

Return True only if the Chute object we have has all the proper things defined to be in a valid state.

setCache (*key, val*)

Set the key:val into the `_cache` dict object to carry around.

paradrop.core.chute.chute_storage module

class ChuteStorage (*filename=None, save_timer=0*)

Bases: `paradrop.lib.utils.pd_storage.PDStorage`

ChuteStorage class.

This class holds onto the list of Chutes on this AP.

It implements the PDStorage class which allows us to save the chuteList to disk transparently

attrSaveable ()

Returns True if we should save the ChuteList, otherwise False.

chuteList = {}

clearChuteStorage ()

deleteChute (*ch*)

Deletes a chute from the chute storage. Can be sent the chute object, or the chute name.

getAttr ()

Get our attr (as class variable for all to see)

getChute (*name*)

Returns a reference to a chute we have in our cache, or None.

getChuteList ()

Return a list of the names of the chutes we know of.

saveChute (*ch*)

Saves the chute provided in our internal chuteList. Also since we just received a new chute to hold onto we should save our ChuteList to disk.

setAttr (*attr*)

Save our attr however we want (as class variable for all to see)

paradrop.core.chute.restart module

Contains the functions required to restart chutes properly on power cycle of device. Checks with pdconfd to make sure it was able to properly bring up all interfaces before starting chutes.

reloadChutes ()

This function is called to restart any chutes that were running prior to the system being restarted. It waits for pdconfd to come up and report whether or not it failed to bring up any of the interfaces that existed before the power cycle. If pdconfd indicates something failed we then force a stop update in order to bring down all interfaces associated with that chute and mark it with a warning. If the stop fails we mark the chute with a warning manually and change its state to stopped and save to storage this isn't great as it could mean our system still has interfaces up associated with that chute. If pdconfd doesn't report failure we attempt to start the chute and if this fails we trust the abort process to restore the system to a consistent state and we manually mark the chute as stopped and add a warning to it. :param None :returns: (list) A list of update dicts to be used to create updateObjects that should be run before accepting new updates.

updateStatus (*update*)

This function is a callback for the updates we do upon restarting the system. It checks whether or not the update completed successfully and if not it changes the state of the chute to stopped and adds a warning. :param update:

The update object containing information about the chute that was created and whether it was successful or not.
:type update: obj :returns: None

Module contents

paradrop.core.config package

Submodules

paradrop.core.config.airshark module

class **AirsharkInterfaceManager**

Bases: `object`

add_observer (*observer*)

interface_available ()

remove_observer (*observer*)

reset_interface ()

set_interface (*interface*)

configure (*update*)

paradrop.core.config.configservice module

configservice module: This module is responsible for “poking” the proper host OS services to change the host OS config. This would include things like changing the networking, DHCP server settings, wifi, etc..

reloadAll (*update*)

paradrop.core.config.devices module

Detect physical devices that can be used by chutes.

This module detects physical devices (for now just network interfaces) that can be used by chutes. This includes WAN interfaces for Internet connectivity and WiFi interfaces which can host APs.

It also makes sure certain entries exist in the system UCI files for these devices, for example “wifi-device” sections. These are shared between chutes, so they only need to be added when missing.

class **SysReader** (*phy*)

Bases: `object`

PCI_BUS_ID = `<_sre.SRE_Pattern object>`

USB_BUS_ID = `<_sre.SRE_Pattern object>`

getDeviceId (*default='????'*)

Return the device ID for the device.

This is a four-digit hexadecimal number. For example, our Qualcomm 802.11n chips have device ID 002a.

getSlotName (*default='????'*)

Return the PCI/USB slot name for the device.

Example: “pci/0000:04:00.0” or “usb/1-1:1.0”

getVendorId (*default='????'*)

Return the vendor ID for the device.

This is a four-digit hexadecimal number. For example, our Qualcomm 802.11n chips have vendor ID 168c.

read_uevent ()

Read the device uevent file and return the contents as a dictionary.

class UCIBuilder

Bases: `object`

UCIBuilder helps aggregate UCI configuration sections for writing to files.

FILES = ['dhcp', 'network', 'firewall', 'wireless', 'qos']

add (*file_, type_, options, name=None*)

Add a new configuration section.

getSections (*file_*)

Get sections associated with a single file.

Returns: list of tuples, [(config, options)]

write ()

Write all of the configuration sections to files.

checkSystemDevices (*update*)

Check whether expected devices are present.

This may reboot the machine if devices are missing and the host config is set to do that.

detectSystemDevices ()

Detect devices on the system.

The result is three lists stored in a dictionary. The three lists are indexed by ‘wan’, ‘wifi’, and ‘lan’. Other devices may be supported by adding additional lists.

Within each list, a device is represented by a dictionary. For all devices, the ‘name’ and ‘mac’ fields are defined. For WiFi devices, the ‘phy’ is defined in addition. Later, we may fill in more device information (e.g. what channels a WiFi card supports).

flushWirelessInterfaces (*phy*)

Remove all virtual interfaces associated with a wireless device.

This should be used before giving a chute exclusive access to a device (e.g. monitor mode), so that it does not inherit unexpected interfaces.

getMACAddress (*ifname*)

getPhyMACAddress (*phy*)

getSystemDevices (*update*)

Detect devices on the system.

Store device information in cache key “networkDevices” as well as “networkDevicesByName”.

getWirelessPhyName (*ifname*)

handleMissingWiFi (*hostConfig*)

Take appropriate action in response to missing WiFi devices.

Depending on the host configuration, we may either emit a warning or reboot the system.

isVirtual (*ifname*)

Test if an interface is a virtual one.

FIXME: This just tests for the presence of certain strings in the interface name, so it is not very robust.

isWAN (*ifname*)

Test if an interface is a WAN interface.

isWireless (*ifname*)

Test if an interface is a wireless device.

listSystemDevices ()

Detect devices on the system.

The result is a single list of dictionaries, each containing information about a network device.

listWiFiDevices ()

readHostconfigVlan (*vlanInterfaces*, *builder*)

readHostconfigWifi (*wifi*, *networkDevices*, *builder*)

readHostconfigWifiInterfaces (*wifiInterfaces*, *networkDevices*, *builder*)

readSysFile (*path*)

resetWirelessDevice (*phy*, *primary_interface*)

Reset a wireless device's interfaces to clean state.

This will rename, delete, or add an interface as necessary to make sure only the primary interface exists, e.g. "wlan0" for a wireless device, e.g. phy0.

resolveWirelessDevRef (*name*, *networkDevices*)

Resolve a WiFi device reference (wlan0, phy0, 00:11:22:33:44:55, etc.) to the name of the device section as used by pdconf (wifiXXXXXXXXXXXX).

Unambiguous naming is preferred going forward (either wifiXX or the MAC address), but to maintain backward compatibility, we attempt to resolve either wlanX or phyX to the MAC address of the device that currently uses that name.

select_brlan_address (*hostConfig*)

Select IP address and netmask to use for LAN bridge.

Behavior depends on the proto field, which can either be 'auto' or 'static'. When proto is set to 'auto', we check the WAN interface address and choose either 10.0.0.0 or 192.168.0.1 to avoid conflict. Otherwise, when proto is set to 'static', we use the specified address.

setConfig (*chuteName*, *sections*, *filepath*)

setSystemDevices (*update*)

Initialize system configuration files.

This section should only be run for host configuration updates.

Creates basic sections that all chutes require such as the "wan" interface.

paradrop.core.config.dhcp module

getVirtDHCPSettings (*update*)

Looks at the runtime rules the developer defined to see if they want a dhcp server. If so it generates the data and stores it into the chute cache key:virtDHCPSettings.

setVirtDHCPSettings (*update*)

Takes a list of tuples (config, opts) and saves it to the dhcp config file.

paradrop.core.config.dockerconfig module

dockerconfig module: This module contains all of the knowledge of how to take internal pdfcd representation of configurations of chutes and translate them into specifically what docker needs to function properly, whether that be in the form of dockerfiles or the HostConfig JSON object known at init time of the chute.

abortCreateVolumeDirs (*update*)

createVolumeDirs (*update*)

generateToken (*bits=128*)

getVirtPreamble (*update*)

paradrop.core.config.firewall module

findMatchingInterface (*iface_name, interfaces*)

Search an interface list for one matching a given name.

iface_name can contain shell-style wildcards (* and ?).

getDeveloperFirewallRules (*update*)

Generate other firewall rules requested by the developer such as redirects. The object returned is a list of tuples (config, options).

getOSFirewallRules (*update*)

There is a set of default things that must exist just for the chute to function at all, generate those here.

Stored in key: osFirewallRules

setOSFirewallRules (*update*)

Takes a list of tuples (config, opts) and saves it to the firewall config file.

paradrop.core.config.haproxy module

This module is responsible for configuration haproxy.

generateConfigSections ()

reconfigureProxy (*update*)

writeConfigFile (*output*)

paradrop.core.config.hostconfig module

The host configuration controls system settings of the host OS.

This module operates as follows:

1. The first time, we try to detect all devices and auto-generate a reasonable configuration, which we store to a persistent file.
2. (TODO) We present the configuration to the owner sometime around provisioning or first chute creation and allow him to change settings.

3. (TODO) We have some kind of update operation that can manipulate settings.

generateHostConfig (*devices*)

Scan for devices on the machine and generate a working configuration.

getHostConfig (*update*)

Load host configuration.

Read device information from networkDevices. Store host configuration in hostConfig.

load (*path=None*)

Load host configuration.

Tries to load host configuration from persistent file. If that does not work, it will try to automatically generate a working configuration.

Returns a host config object on success or None on failure.

prepareHostConfig (*devices=None, hostConfigPath=None, write=True*)

Load an existing host configuration or generate one.

Tries to load host configuration from persistent file. If that does not work, it will try to automatically generate a working configuration.

write: if True and host config was automatically generated, then write the new host config to a file.

revertHostConfig (*update*)

Restore host configuration from before update.

Uses oldHostConfig cache entry.

save (*config, path=None*)

Save host configuration.

May raise exception if unable to write the configuration file.

setAutoUpdate (*enable*)

setHostConfig (*update*)

Write host configuration to persistent storage.

Read host configuration from hostConfig.

paradrop.core.config.network module

abortNetworkConfig (*update*)

Release resources claimed by chute network configuration.

chooseExternalIntf (*update, iface*)

chooseSubnet (*update, cfg, iface*)

fulfillDeviceRequest (*update, cfg, devices*)

Find a physical device that matches the requested device type.

Raises an exception if one cannot be found.

getExtraOptions (*cfg*)

Get dictionary of extra wifi-iface options that we are not interpreting but just passing on to pdconf.

getInterfaceAddress (*update, name, cfg, iface*)

Dynamically select IP address for the chute interface.

This function will use a subnet from the chute subnet pool and assign IP addresses to the external (in host) and internal (in chute) interfaces.

The addresses are stored in the iface object.

getInterfaceDict (*chute*)

Return interfaces from a chute as a dict with interface names as the keys. Returns an empty dict if chute is None or it had no interfaces.

getL3BridgeConfig (*update*)

Creates configuration sections for layer 3 bridging.

getNetworkConfig (*update*)

For the Chute provided, return the dict object of a 100% filled out configuration set of network configuration. This would include determining what the IP addresses, interfaces names, etc. . .

Store configuration in networkInterfaces cache entry.

getNetworkConfigLan (*update, name, cfg, iface*)

getNetworkConfigVlan (*update, name, cfg, iface*)

getNetworkConfigWifi (*update, name, cfg, iface*)

getOSNetworkConfig (*update*)

Takes the network interface obj created by NetworkManager.getNetworkConfiguration and returns a properly formatted object to be passed to the UCIconfig class. The object returned is a list of tuples (config, options).

getWifiKeySettings (*cfg, iface*)

Read encryption settings from cfg and transfer them to iface.

get_current_phy_conf (*update, device_id*)

Lookup current configuration for a network device.

This includes information such as the Wi-Fi channel.

Returns a dictionary, which may be empty if no configuration was found.

reclaimNetworkResources (*chute*)

Reclaim network resources for a previously running chute.

This function only applies to the special case in which pd starts up and loads a list of chutes that were running. This function marks their IP addresses and interface names as taken so that new chutes will not use the same values.

satisfies_requirements (*obj, requirements*)

Checks that an object satisfies given requirements.

Every key-value pair in the requirements object must be present in the target object for it to be considered satisfied.

Returns True/False.

select_chute_subnet_pool (*host_config*)

Select IP subnet to use as pool for chutes.

Behavior depends on whether a static subnet is configured or auto configuration is requested. If the chuteSubnetPool option is set to 'auto', then we check the WAN interface address and choose either 10.128.0.0/9 or 192.168.128.0/17 to avoid conflict. Otherwise, we used the specified subnet.

setL3BridgeConfig (*update*)

Apply configuration for layer 3 bridging.

setOSNetworkConfig (*update*)

Takes a list of tuples (config, opts) and saves it to the network config file.

paradrop.core.config.osconfig module

osconfig module: This module is in charge of changing configuration files for pdfcd on the host OS. This relates to things like network, dhcp, wifi, firewall changes. Pdfcd should be able to make simple abstracted calls into this module so that if we need to change what type of OS config we need to support only this module would change.

applyWaitOnlineFix (*update*)

Configure systemd-networkd to wait for WAN interface only.

This fixes the long delay on system startup while it waits for all network interfaces to be connected.

revertConfig (*update, theType*)

Basically the UCI system saves a backup of the original config file, if we need to revert changes at all, we can just tell our UCI module to revert back using that backup copy.

paradrop.core.config.power module

reboot (*update*)

shutdown (*update*)

paradrop.core.config.reservations module

Module for checking resource reservations by chutes.

One idea motivating this design is to reduce the amount of state in memory for resource reservations. We have the chute list, which contains information about what devices the chute is using. If we also maintain a separate list of devices used by chutes, we need to keep them synchronized. This becomes messy when a chute fails to install or uninstall correctly. The `getDeviceReservations` function iterates over the chute list and returns an up-to-date view of device usage. This can be called as needed.

class DeviceReservations

Bases: `object`

add (*chute, dtype, mode=None*)

count (*dtype=None, mode=None*)

Return the number of reservations matching the given criteria.

None is used as a wildcard, so if no arguments are passed, the count returned is the total number of reservations.

class InterfaceReservationSet

Bases: `object`

add (*interface*)

class SubnetReservationSet

Bases: `object`

add (*subnet*)

getDeviceReservations (*exclude=None*)

Produce a dictionary mapping device names to DeviceReservations objects that describe the current usage of the device.

The returned type is a defaultdict, so there is no need to check if a key exists before accessing it.

exclude: name of chute whose device reservations should be excluded

getInterfaceReservations (*exclude=None*)

Get current set of interface reservations.

Returns an instance of InterfaceReservationSet.

exclude: name of chute whose interfaces should be excluded

getReservations (*update*)

Get device and resource reservations claimed by other users.

getSubnetReservations (*exclude=None*)

Get current set of subnet reservations.

Returns an instance of SubnetReservationSet.

exclude: name of chute whose reservations should be excluded

paradrop.core.config.resource module

computeResourceAllocation (*chutes*)

getResourceAllocation (*update*)

paradrop.core.config.services module

Configure optional additional services such as telemetry.

configure_telemetry (*update*)

paradrop.core.config.snap module

updateSnap (*update*)

paradrop.core.config.state module

removeAllChutes (*update*)

revertChute (*update*)

saveChute (*update*)

paradrop.core.config.uciutils module

restoreConfigFile (*chute, configname*)

Restore a system config file from backup.

This can only be used during a chute update operation to revert changes that were made during that update operation.

configname: name of configuration file (“network”, “wireless”, etc.)

setConfig (*chute, old, cacheKeys, filepath*)

Helper function used to modify config file of each various setting in /etc/config/ Returns:

True: if it modified a file False: if it did NOT cause any modifications

Raises exception if an error occurs.

paradrop.core.config.wifi module

getOSWirelessConfig (*update*)

Read settings from networkInterfaces for wireless interfaces. Store wireless configuration settings in osWirelessConfig.

setOSWirelessConfig (*update*)

Write settings from osWirelessConfig out to UCI files.

paradrop.core.config.zerotier module

configure (*update*)

getAddress ()

Return the zerotier address for this device or None if unavailable.

get_auth_token ()

Return the zerotier auth token for accessing its API.

get_networks ()

Get list of active ZeroTier networks.

manage_network (*nwid*, *action*='join')

Join or leave a ZeroTier network.

nwid: ZeroTier network ID, e.g. "e5cd7a9e1c8a5e83" action: either "join" or "leave"

Module contents

paradrop.core.container package

Submodules

paradrop.core.container.chutecontainer module

class ChuteContainer (*name*, *docker_url*='unix://var/run/docker.sock')

Bases: `object`

Class for accessing information about a chute's container.

getID ()

Look up the container ID as used by Docker.

getIP ()

Look up the IP address assigned to the container.

getPID ()

Look up the PID of the container, if running.

getPortConfiguration (*port*, *protocol*='tcp')

Look up network port configuration. This tells us if a port in the host is bound to a port inside the container.

Returns a list, typically with zero or one elements.

Example:

```
[{ "HostIp": "0.0.0.0", "HostPort": "32768"
}]
```

getStatus ()

Return the status of the container (running, exited, paused).

Returns “missing” if the chute does not exist.

inspect ()

Return the full container status from Docker.

isRunning ()

Check if container is running.

Returns True/False; returns False if the container does not exist.

paradrop.core.container.dockerapi module

Functions associated with deploying and cleaning up docker containers.

buildImage (update)

Build the Docker image and monitor progress.

build_host_config (chute)

Build the host_config dict for a docker container based on the passed in update.

Parameters **chute** (*obj*) – The chute object containing information about the chute.

Returns (dict) The host_config dict which docker needs in order to create the container.

call_in_netns (chute, env, command, onerror='raise', pid=None)

Call command within a chute’s namespace.

command: should be a list of strings. onerror: should be “raise” or “ignore”

call_retry (cmd, env, delay=3, tries=3)

cleanup_net_interfaces (chute)

Cleanup special interfaces when bringing down a container.

This applies to monitor mode interfaces, which need to be renamed before they come back to the host network, e.g. “mon0” inside the container should be renamed to the appropriate “wlanX” before the container exits.

getBridgeGateway ()

Look up the gateway IP address for the docker bridge network.

This is the docker0 IP address; it is the IP address of the host from the chute’s perspective.

getImageName (chute)

getPortList (chute)

Get a list of ports to expose in the format expected by create_container.

Uses the port binding dictionary from the chute host_config section. The keys are expected to be integers or strings in one of the following formats: “port” or “port/protocol”.

Example: port_bindings = {

 “1111/udp”: 1111, “2222”: 2222

} getPortList returns [(1111, ‘udp’), (2222, ‘tcp’)]

prepare_environment (chute)

Prepare environment variables for a chute container.

prepare_port_bindings (*chute*)

removeAllContainers (*update*)

Remove all containers on the system. This should only be used as part of a factory reset mechanism.

Returns None

removeChute (*update*)

Remove a docker container and the image it was built on based on the passed in update.

Parameters **update** (*obj*) – The update object containing information about the chute.

Returns None

removeNewContainer (*update*)

Remove the newly started container during abort sequence.

removeNewImage (*update*)

Remove the newly built image during abort sequence.

removeOldContainer (*update*)

Remove the docker container for the old version of a chute.

Parameters **update** (*obj*) – The update object containing information about the chute.

Returns None

removeOldImage (*update*)

Remove the image for the old version of the chute.

restartChute (*update*)

Start a docker container based on the passed in update.

Parameters **update** (*obj*) – The update object containing information about the chute.

Returns None

revertResourceAllocation (*update*)

setResourceAllocation (*update*)

setup_net_interfaces (*chute*)

Link interfaces in the host to the internal interfaces in the Docker container.

The commands are based on the pipework script (<https://github.com/jpetazzo/pipework>).

Parameters **chute** – The chute object containing information about the chute.

Returns None

startChute (*update*)

Create a docker container based on the passed in update.

startOldContainer (*update*)

Create a docker container using the old version of the image.

stopChute (*update*)

Stop a docker container based on the passed in update.

Parameters **update** (*obj*) – The update object containing information about the chute.

Returns None

writeDockerConfig ()

Write options to Docker configuration.

Mainly, we want to tell Docker not to start containers automatically on system boot.

paradrop.core.container.dockerfile module

This module generates a Dockerfile for use with light chutes.

```

class Dockerfile (config)
    Bases: object

    getBytesIO ()
        Geterate a Dockerfile and return as a BytesIO object.

    getString ()
        Generate a Dockerfile as a multi-line string.

    isValid ()
        Check if configuration is valid.

        Returns a tuple (True/False, None or str).

    requiredFields = ['use', 'command']

    writeFile (path)
        Generate Dockerfile and write to a file.

```

paradrop.core.container.downloader module

This module downloads a package from a given URL using one of potentially many different methods. We currently support the github web API and simple HTTP(S). The github method is more developed and returns meta data about the project (the commit hash and message), but support for other methods, e.g. download a tar file that was uploaded to a web server, are not precluded.

Private downloads are supported with the HTTP Authorization header. For github, we need to use the github API to request a token to access the owner's private repository. That part is not implemented here.

```

class Downloader (url, user=None, secret=None, repo_owner=None, repo_name=None)
    Bases: object

    download ()

    extract ()

    fetch ()
        Download the project.

        Returns the full path to the temporary directory containing the project and a dictionary containing meta data.

    meta ()

class GithubDownloader (url, checkout='master', **kwargs)
    Bases: paradrop.core.container.downloader.Downloader

    download ()

    meta ()
        Return repository meta data as a dictionary.

class WebDownloader (url, user=None, secret=None, repo_owner=None, repo_name=None)
    Bases: paradrop.core.container.downloader.Downloader

    download ()

    meta ()
        Return repository meta data as a dictionary.

```

download (*url*, *user=None*, *secret=None*, ***kwargs*)

Return an appropriate Downloader for the given URL.

This should be used in a “with ... as ...” statement to perform cleanup on all exit cases.

Example: with `downloader(“https://github.com/...”) as dl`:

```
path, meta = dl.fetch() # do some work on the repo here
```

paradrop.core.container.log_provider module

Provides messages from container logs (STDOUT and STDERR).

class `LogProvider` (*chutename*)

Bases: `object`

attach ()

Start listening for log messages.

Log messages in the queue will appear like the following: {

```
    ‘timestamp’: ‘2017-01-30T15:46:23.009397536Z’, ‘message’: ‘Something happened’
```

```
}
```

detach ()

Stop listening for log messages.

After this is called, no additional messages will be added to the queue.

get_logs ()

monitor_logs (*chute_name*, *queue*, *tail=200*)

Iterate over log messages from a container and add them to the queue for consumption. This function will block and wait for new messages from the container. Use the queue to interface with async code.

tail: number of lines to retrieve from log history; the string “all” is also valid, but highly discouraged for performance reasons.

Module contents

paradrop.core.plan package

Submodules

paradrop.core.plan.executionplan module

This module contains the methods required to generate and act upon execution plans.

An execution plan is a set of operations that must be performed to update a Chute from some old state into the new state provided by the API server.

All plans that are generated are function pointers, as in no actual operations are performed during the generation process.

abortPlans (*update*)

This function should be called if one of the Plan objects throws an Exception. It takes the PlanMap argument and calls the getNextAbort function just like executePlans does with todoPlans. This dynamically generates an abort plan list based on what plans were originally executed. Returns:

True in error : This is really bad False otherwise : we were able to restore system state back to before the executePlans function was called

aggregatePlans (*update*)

Takes the PlanMap provided which can be a combination of changes for multiple different chutes and it puts things into a sane order and removes duplicates where possible.

This keeps things like reloading networking from happening twice if 2 chutes make changes.

Returns: A new PlanMap that should be executed

executePlans (*update*)

Primary function that actually executes all the functions that were added to plans by all the exc modules. This function can heavily modify the OS/files/etc.. so the return value is very important. Returns:

True in error : abortPlans function should be called False otherwise : everything is OK

generatePlans (*update*)

For an update object provided this function references the updateModuleList which lets all exc modules determine if they need to add functions to change the state of the system when new chutes are added to the OS.

Returns: True in error, as in we should stop with this update plan

paradrop.core.plan.hostconfig module

This module generates update plans for a host configuration operation. It is separate from the modules that generate plans for chute operations because we only need to do a subset of the operations.

generatePlans (*update*)

paradrop.core.plan.name module

generatePlans (*update*)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

paradrop.core.plan.plangraph module

class Plan (*func, *args*)

Helper class to hold onto the actual plan data associated with each plan

class PlanMap (*name*)

This class helps build a dependency graph required to determine what steps are required to update a Chute from a previous version of its configuration.

addMap (*other*)

Takes another PlanMap object and appends whatever the plans are into this plans object.

addPlans (*priority, todoPlan, abortPlan=[]*)

Adds new Plan objects into the list of plans for this PlanMap.

Arguments: @priority : The priority number (1 is done first, 99 done last - see PRIORITYFLAGS section at top of this file) @todoPlan : A tuple of (function, (args)), this is the function that completes the actual task requested

the args can either be a single variable, a tuple of variables, or None.

@abortPlan [A tuple of (function, (args)) or a list of tuple or None.] This is what should be called if a plan somewhere in the chain fails and we need to undo the work we did here - this function is only called if a higher priority function fails (ie we were called, then something later on fails that would cause us to undo everything we did to setup/change the Chute).

getNextAbort ()

Like an iterator function, it returns each element in the list of abort plans in order.

Returns: (function, args) : Each todo is returned just how the user first added it None : None is returned when there are no more todo's

getNextTodo ()

Like an iterator function, it returns each element in the list of plans in order.

Returns: (function, args) : Each todo is returned just how the user first added it None : None is returned when there are no more todo's

registerSkip (func)

Register this function as one to skip execution on, if provided it shouldn't return the (func, args) tuple as a result from the getNextTodo function.

sort ()

Sorts the plans based on priority.

paradrop.core.plan.resource module

generatePlans (update)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

paradrop.core.plan.router module

This module generates update plans for router operations such as factory reset.

generatePlans (update)

paradrop.core.plan.runtime module

generatePlans (update)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

paradrop.core.plan.snap module

This module generates update plans for a snap operation.

generatePlans (update)

paradrop.core.plan.state module

generatePlans (*update*)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

paradrop.core.plan.struct module

generatePlans (*update*)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

paradrop.core.plan.traffic module

generatePlans (*update*)

This function looks at a diff of the current Chute (in @chuteStor) and the @newChute, then adds Plan() calls to make the Chute match the @newChute.

Returns: True: abort the plan generation process

Module contents

paradrop.core.system package

Submodules

paradrop.core.system.system_info module

Get system information

getDMI ()

Read hardware information from DMI.

This function attempts to read from known files in /sys/class/dmi/id/. If any are missing or an error occurs, those fields will be omitted from the result.

Returns: a dictionary with fields such as bios_version and product_serial.

getOSVersion ()

Return a string identifying the host OS.

getPackageVersion (*name*)

Get a python package version.

Returns: a string or None

paradrop.core.system.system_status module

Get system running status including CPU load, memory usage, network traffic.

class `SystemStatus`

Bases: `object`

`INCLUDED_PARTITIONS = set(['/writable', '/'])`

`classmethod getNetworkInfo()`

`classmethod getProcessInfo(pid)`

`getStatus(max_age=0.8)`

Get current system status.

max_age: maximum tolerable age of cached status information. Set to None to force a refresh regardless of cache age.

Returns a dictionary with fields 'cpu_load', 'mem', 'disk', and 'network'.

`classmethod getSystemInfo()`

`refreshCpuLoad()`

`refreshDiskInfo()`

`refreshMemoryInfo()`

`refreshNetworkTraffic()`

Module contents

paradrop.core.update package

Submodules

paradrop.core.update.update_fetcher module

Fetch new updates from the pdserver and apply the updates

class `UpdateFetcher(update_manager)`

Bases: `object`

`pull_update(**kwargs)`

Start updates by polling the server for the latest updates.

This is the only method that needs to be called from outside. The rest are triggered asynchronously.

Call chain: pull_update -> _updates_received -> _update_complete

_auto: Set to True when called by the scheduled LoopingCall.

`start_long_poll(**kwargs)`

`start_polling(**kwargs)`

paradrop.core.update.update_manager module

class `UpdateManager` (*reactor*)

This class is in charge of making the configuration changes required on the chutes. It utilizes the ChuteStorage class to hold onto the chute data.

Use @updateChutes to make the configuration changes on the AP. This function is thread-safe, this class will only call one update set at a time. All others are held in a queue until the last update is complete.

add_update (***update*)

MUTEX: updateLock Take the list of Chutes and push the list into a queue object, this object will then call the real update function in another thread so the function that called us is not blocked.

We take a callable responseFunction to call, when we are done with this update we should call it.

assign_change_id ()

Get a unique change ID for an update.

This should be used to set the change_id field in an update object.

clear_update_list ()

MUTEX: updateLock Clears all updates from list (new array).

find_change (*change_id*)

Search active and queued changes for the requested change.

Returns an Update object or None.

paradrop.core.update.update_object module

This holds onto the UpdateObject class. It allows us to easily abstract away different update types and provide a uniform way to interpret the results through a set of basic actionable functions.

class `UpdateChute` (*obj*)

Bases: `paradrop.core.update.update_object.UpdateObject`

Updates specifically tailored to chute actions like create, delete, etc...

`updateModuleList = [<module 'paradrop.core.plan.name' from '/home/docs/checkouts/readt...`

class `UpdateObject` (*obj*)

Bases: `object`

The base UpdateObject class, covers a few basic methods but otherwise all the intelligence exists in the inherited classes.

All update information passed by the API server is contained as variables of this class such as update.updateType, update.updateClass, etc...

By default, the following variables should be utilized: responses : an array of messages any module can choose to append warnings or errors to

failure [the module that chose to fail this update can set a string message to return] : to the user in the failure variable. It should be very clear as to why the : failure occurred, but if the user wants more information they may find it : in the responses variable which may contain debug information, etc...

add_message_observer (*observer*)

complete (***kwargs*)

Signal to the API server that any action we need to perform is complete and the API server can finish its connection with the client that initiated the API request.

execute()

The function that actually walks through the main process required to create the chute. It follows the executeplan module through the paces of:

1. Generate the plans for each plan module
2. Prioritize the plans
3. Execute the plans

If at any point we fail then this function will directly take care of completing the update process with an error state and will close the API connection.

progress(message)

remove_message_observer(observer)

started()

This function should be called when the updated object is dequeued and execution is about to begin.

Sends a notification to the pdserver if this is a tracked update.

updateModuleList = []

class UpdateRouter(obj)

Bases: *paradrop.core.update.update_object.UpdateObject*

Updates specifically tailored to router configuration.

updateModuleList = [<module 'paradrop.core.plan.hostconfig' from '/home/docs/checkouts/readt

class UpdateSnap(obj)

Bases: *paradrop.core.update.update_object.UpdateObject*

Updates specifically tailored to installing snaps.

updateModuleList = [<module 'paradrop.core.plan.snap' from '/home/docs/checkouts/readt

parse(obj)

Determines the update type and returns the proper class.

Module contents

Module contents

12.1.6 paradrop.lib package

Subpackages

paradrop.lib.misc package

Submodules

paradrop.lib.misc.pdinstall module

sendCommand(command, data)

Send a command to the pdinstall service.

Commands: install - Install snaps from a file path or http(s) URL.

Required data fields: sources - List with at least one snap file path or URL. The snaps are installed in order until one succeeds or all fail.

Returns True/False for success. Currently, we cannot check whether the call succeeded, only whether it was delivered. A return value of False means we could not deliver the command to pdinstall.

paradrop.lib.misc.procmon module

The ProcessMonitor class ensures that a service is running and that its pid file is consistent.

This addresses an issue we have had with Docker on Ubuntu Snappy, where its pid file sometimes persists and prevents the service from starting.

class ProcessMonitor (*service, cmdstring=None, pidfile=None, action='restart'*)

Bases: `object`

allowedActions = `set(['reboot', 'restart'])`

check ()

Check that the service is running and consistent with pid file(s).

Returns True or False.

ensureReady (*delay=5, tries=3*)

Look through checking and restarting the service until it is ready or the maximum number of tries has been reached.

delay: time delay (seconds) between retries. tries: maximum number of restart-wait-check cycles.

restart ()

Restart the service.

paradrop.lib.misc.resopt module

Resource optimization functions.

allocate (*reservations, total=1.0*)

Allocate resources among slices with specified and unspecified reservations.

Returns a new list of values with the following properties: - Every value is $\geq$ the corresponding input value. - The result sums to *total*.

Examples: `allocate([0.25, None, None]) -> [0.5, 0.25, 0.25]` `allocate([0.4, None, None]) -> [0.6, 0.2, 0.2]` `allocate([0.2, 0.2, 0.2]) -> [0.33, 0.33, 0.33]` `allocate([None, None, None]) -> [0.33, 0.33, 0.33]` `allocate([0.5, 0.5, 0.5]) -> ERROR`

paradrop.lib.misc.snapd module

class SnapdClient (*logging=True, wait_async=False*)

Bases: `object`

Client for interacting with the snapd API to manage installed snaps.

connect (*plug_snap='paradrop-daemon', plug=None, slot_snap='core', slot=None*)

Connect an interface.

get_change (*change_id*)

Get the current status of a change.

installSnap (*snapName*)

Install a snap from the store.

listSnaps ()

Get a list of installed snaps.

updateSnap (*snapName, data*)

Post an update to a snap.

Valid actions are: install, refresh, remove, revert, enable, disable.

Example: updateSnap("paradrop-daemon", {"action": "refresh"})

paradrop.lib.misc.ssh_keys module

addAuthorizedKey (*key, user='paradrop'*)

getAuthorizedKeys (*user='paradrop'*)

writeAuthorizedKeys (*keys, user='paradrop'*)

Module contents

paradrop.lib.utils package

Submodules

paradrop.lib.utils.addresses module

checkPhyExists (*radioid*)

Check if this chute exists at all, a directory /sys/class/ieee80211/phyX must exist.

getGatewayIntf (*ch*)

Looks at the key:networkInterfaces for the chute and determines what the gateway should be including the IP address and the internal interface name.

Returns: A tuple (gatewayIP, gatewayInterface) None if networkInterfaces doesn't exist or there is an error

getInternalIntfList (*ch*)

Takes a chute object and uses the key:networkInterfaces to return a list of the internal network interfaces that will exist in the chute (e.g., eth0, eth1, ...)

Returns: A list of interface names None if networkInterfaces doesn't exist or there is an error

getSubnet (*ipaddr, netmask*)

getWANIntf (*ch*)

Looks at the key:networkInterfaces for the chute and finds the WAN interface.

Returns: The dict from networkInterfaces None

incIpaddr (*ipaddr, inc=1*)

Takes a quad dot format IP address string and adds the @inc value to it by converting it to a number.

Returns: Incremented quad dot IP string or None if error

isIpAvailable (*ipaddr, chuteStor, name*)

Make sure this IP address is available.

Checks the IP addresses of all zones on all other chutes, makes sure subnets are not the same.

isIpValid (*ipaddr*)

Return True if Valid, otherwise False.

isStaticIpAvailable (*ipaddr, chuteStor, name*)

Make sure this static IP address is available.

Checks the IP addresses of all zones on all other chutes, makes sure not equal.

isWifiSSIDAvailable (*ssid, chuteStor, name*)

Make sure this SSID is available.

maxIpaddr (*ipaddr, netmask*)

Takes a quad dot format IP address string and makes it the largest valid value still in the same subnet.

Returns: Max quad dot IP string or None if error

paradrop.lib.utils.datastruct module

Utilities for reading from data structures.

getValue (*struct, path, default=None*)

Read a value from the data structure.

Arguments: struct can comprise one or more levels of dicts and lists. path should be a string using dots to separate levels. default will be returned if the path cannot be traced.

Example: `getValue({'a': [1, 2, 3]}, "a.1")` -> 2 `getValue({'a': [1, 2, 3]}, "a.3")` -> None

paradrop.lib.utils.pd_storage module

class PDStorage (*filename, saveTimer*)

Bases: `object`

ParaDropStorage class.

This class is designed to be implemented by other classes. Its purpose is to make whatever data is considered important persistent to disk.

The implementer can override functions in order to implement this class: `getAttr()` : Get the attr we need to save to disk `setAttr()` : Set the attr we got from disk `importAttr()`: Takes a payload and returns the properly formatted data `exportAttr()`: Takes the data and returns a payload `attrSaveable()`: Returns True if we should save this attr

attrSaveable ()

THIS SHOULD BE OVERRIDEN BY THE IMPLEMENTER.

exportAttr (*data*)

By default do nothing, but expect that this function could be overwritten

importAttr (*pyld*)

By default do nothing, but expect that this function could be overwritten

loadFromDisk ()

Attempts to load the data from disk. Returns True if success, False otherwise.

saveToDisk ()

Saves the data to disk.

paradrop.lib.utils.pdos module

basename (*x*)

copy (*a*, *b*)

copytree (*a*, *b*)
shutil's copytree is dumb so use distutils.

exists (*p*)

fixpath (*p*)
This function is required because if we need to pass a path to something like tarfile, we cannot overwrite the function to fix the path, so we need to expose it somehow.

getFileType (*f*)

getMountCmd ()

isMount (*mnt*)
This function checks if @mnt is actually mounted.

isdir (*a*)

isfile (*a*)

ismount (*p*)

listdir (*p*)

mkdir (*p*)

move (*a*, *b*)

open (*p*, *mode*)

oscall (*cmd*, *get=False*)
This function performs a OS subprocess call. All output is thrown away unless an error has occurred or if @get is True Arguments:

@cmd: the string command to run [get] : True means return (stdout, stderr)

Returns: None if not @get and no error (stdout, retcode, stderr) if @get or yes error

readFile (*filename*, *array=True*, *delimiter='\n'*)

Reads in a file, the contents is NOT expected to be binary. Arguments:

@filename: absolute path to file @array : optional: return as array if true, return as string if False

@delimiter: optional: if returning as a string, this str specifies what to use to join the lines

Returns: A list of strings, separated by newlines None: if the file doesn't exist

remove (*path*, *suppressNotFound=False*)

symlink (*a*, *b*)

unlink (*p*)

write (*filename*, *data*, *mode='w'*)
Writes out a config file to the specified location.

writeFile (*filename*, *line*, *mode='a'*)
Adds the following cfg (either str or list(str)) to this Chute's current config file (just stored locally, not written to file.

paradrop.lib.utils.pdosq module

Quiet pdos module. Implements utility OS operations without relying on the output module. Therefore, this module can be used by output without circular dependency.

makedirs (*p*)

Recursive directory creation (like `mkdir -p`). Returns True if the path is successfully created, False if it existed already, and raises an `OSError` on other error conditions.

paradrop.lib.utils.uci module

class UCISConfig (*filepath*)

Wrapper around the UCI configuration files. These files are found under `/etc/config/`, and are used by *OpenWrt* to keep track of configuration for modules typically found in `/etc/init.d/`

The modules of interest and with current support are:

- firewall
- network
- wireless
- qos
- This class should work with any UCI module but ALL others are UNTESTED!

New configuration settings can be added to the UCI file via `addConfig()`.

Each UCI config file is expected to contain the following syntax:

```
config keyA [valueA] option key1 value1 ... list key2 value1 list key2 value2 ... list key3 value1
list key3 value2
```

Based on the UCI file above, the config syntax would look like the following: `config` is a list of tuples, containing 2 dict objects in each tuple:

- **tuple[0] describes the first line (config keyA [valueA])** {`'type': keyA`, `'name': valueA`}
The value parameter is optional and if missing, then the `'name'` key is also missing (rather than set to `None`).
- **tuple[1] describes the options associated with the settings (both 'option' and 'list' lines)**
{`'key1': 'value1', ...`}

If a list is present, it looks like the following:

```
{ ..., 'key2': [value1, value2, ...], 'key3': [value1, value2, ...]
}
```

So for the example above, the full config definition would look like: `C = {'type': 'keyA', 'name': 'valueA'}` `O = {'key1': 'value1', 'key2': ['value1', 'value2'], 'key3': ['value1', 'value2']}` `config = [(C, O)]`

addConfig (*config, options*)

Adds the tuple to our config.

addConfigs (*configs*)

Adds a list of tuples to our config

backup (*backupToken*)

Puts a backup of this config to the location specified in @backupPath.

delConfig (*config, options*)

Finds a match to the config input and removes it from the internal config data structure.

delConfigs (*configs*)

Adds a list of tuples to our config

existsConfig (*config, options*)

Tests if the (config, options) is in the current config file.

getChuteConfigs (*internalid*)

getConfig (*config*)

Returns a list of call configs with the given title

getConfigIgnoreComments (*config*)

Returns a list of call configs with the given title. Comments are ignored.

readConfig ()

Reads in the config file.

restore (*backupToken, saveBackup=True*)

Replaces real file (at /etc/config/) with backup copy from /tmp/-@backupToken location.

Arguments: backupToken: The backup token appended at the end of the backup path saveBackup : A flag to keep a backup copy or delete it (default is keep backup)

save (*backupToken='paradrop', internalid=None*)

Saves out the file in the proper format.

Arguments:

[backupPath] [Save a backup copy of the UCI file to the path provided.] Should be a token name like 'backup', it gets appended with a hyphen.

chuteConfigsMatch (*chutePre, chutePost*)

Takes two lists of objects, and returns whether or not they are identical.

getLineParts (*line*)

Split the UCI line into its whitespace-separated parts.

Returns a list of strings, with apostrophes removed.

getSystemConfigDir ()

getSystemPath (*filename*)

Get the path to the system configuration file.

This function also attempts to create the configuration directory if it does not exist.

Typical filenames: network, wireless, qos, firewall, dhcp, etc.

isMatch (*a, b*)

isMatchIgnoreComments (*a, b*)

singleConfigMatches (*a, b*)

stringify (*a*)

Recursively convert all primitives in a data structure to strings.

stringifyOptionValue (*value*)

Convert option value from in-memory representation to a suitable string.

In particular, boolean values are converted to ‘0’ or ‘1’.

paradrop.lib.utils.uhttp module

```
class UHTTPConnection(path)
    Bases: httpplib.HTTPConnection

    Subclass of Python library HTTPConnection that uses a unix-domain socket.

    Source: http://7bits.nl/blog/posts/http-on-unix-sockets-with-python

    connect ()
```

Module contents

Module contents

12.2 Submodules

12.3 paradrop.main module

Core module. Contains the entry point into Paradrop and establishes all other modules. Does not implement any behavior itself.

```
class Nexus(update_fetcher)
    Bases: paradrop.base.nexus.NexusBase

    onStart (**kwargs)

    onStop ()

main ()
```

12.4 paradrop.plan_demo module

This module is here purely to help with understanding the rather complex execution plan in Paradrop. Simply run it (python -m paradrop.plan_demo), and it will walk through all of the functions that make up a chute creation operation.

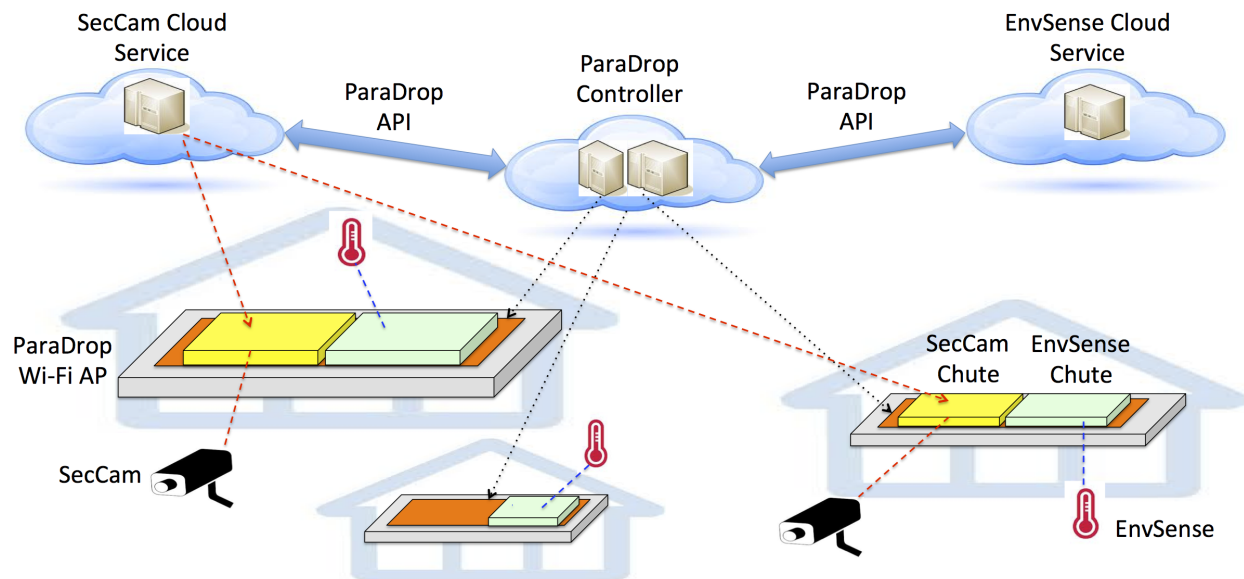
```
loadPriorityMap ()
    Make a map of priority values back to their names for reference.

    These are defined as constant integer values in paradrop.backend.exc.plagraph. For example, for priority 9 (STRUCT_GET_SYSTEM_DEVICES), the dictionary produced by this function would contain the entry 9: "STRUCT_GET_SYSTEM_DEVICES".
```

12.5 Module contents

ParaDrop - Enabling Edge Computing at the Extreme Edge

ParaDrop is an open source edge computing platform developed by the [WiNGS Lab](#) at the University of Wisconsin-Madison. We built the ParaDrop platform with WiFi routers, so that we can “paradrop” services from the cloud to the extreme wireless edge - just one hop from user’s mobile devices, data sources, and actuators of IoT applications. The name “ParaDrop” comes from the ability to “drop” supplies and resources (“services”) into the network edge.



The above figure gives a high level overview of ParaDrop, including the ParaDrop platform and two example applications. With the ParaDrop API, third-party applications can deploy services into the network edge - the WiFi routers. More information about the design and evolution of ParaDrop can be found in the [paper](#).

CHAPTER 14

Getting Started

Please visit the [Quick Start](#) page for a quick introduction about how to use ParaDrop.

CHAPTER 15

Where to go from here?

We have document about ParaDrop application development found under *Developing Applications*. If you are interested in working on the development of the ParaDrop platform (our github code) then check out: *How to Contribute*.

/api

GET /api/v1/chutes/, [53](#)
GET /api/v1/chutes/(chute), [59](#)
GET /api/v1/chutes/(chute)/cache, [58](#)
GET /api/v1/chutes/(chute)/config, [57](#)
GET /api/v1/chutes/(chute)/networks, [57](#)
GET /api/v1/chutes/(chute)/networks/(network),
[57](#)
GET /api/v1/chutes/(chute)/networks/(network)/hostapd_status,
[54](#)
GET /api/v1/chutes/(chute)/networks/(network)/leases,
[55](#)
GET /api/v1/chutes/(chute)/networks/(network)/ssid,
[56](#)
GET /api/v1/chutes/(chute)/networks/(network)/stations,
[55](#)
GET /api/v1/chutes/(chute)/networks/(network)/stations/(mac),
[54](#)
GET /api/v1/config/hostconfig, [60](#)
GET /api/v1/config/new-config, [60](#)
GET /api/v1/config/pdconf, [61](#)
GET /api/v1/config/pdid, [61](#)
GET /api/v1/config/provision, [61](#)
GET /api/v1/config/sshKeys/(user), [61](#)
GET /api/v1/info/environment, [61](#)
GET /api/v1/info/features, [63](#)
GET /api/v1/info/hardware, [62](#)
GET /api/v1/info/software, [63](#)
GET /api/v1/info/telemetry, [62](#)
POST /api/v1/config/factoryReset, [59](#)
POST /api/v1/config/provision, [60](#)
POST /api/v1/config/sshKeys/(user), [61](#)
PUT /api/v1/chutes/(chute)/config, [58](#)
PUT /api/v1/chutes/(chute)/networks/(network)/ssid,
[56](#)
PUT /api/v1/config/hostconfig, [59](#)
PUT /api/v1/config/pdconf, [61](#)

p

paradrop, 165
paradrop.airshark, 99
paradrop.airshark.airshark, 97
paradrop.airshark.analyzer, 97
paradrop.airshark.scanner, 98
paradrop.airshark.spectrum_reader, 98
paradrop.backend, 115
paradrop.backend.airshark_api, 99
paradrop.backend.airshark_ws, 99
paradrop.backend.auth, 100
paradrop.backend.chute_api, 100
paradrop.backend.config_api, 107
paradrop.backend.cors, 110
paradrop.backend.http_server, 110
paradrop.backend.information_api, 111
paradrop.backend.log_sockjs, 113
paradrop.backend.password_api, 113
paradrop.backend.password_manager, 114
paradrop.backend.snapd_resource, 114
paradrop.backend.status_sockjs, 114
paradrop.base, 122
paradrop.base.cxbr, 115
paradrop.base.exceptions, 116
paradrop.base.nexus, 116
paradrop.base.output, 118
paradrop.base.pdutils, 121
paradrop.base.settings, 122
paradrop.conf, 135
paradrop.conf.base, 122
paradrop.conf.client, 125
paradrop.conf.command, 125
paradrop.conf.dhcp, 126
paradrop.conf.firewall, 126
paradrop.conf.main, 129
paradrop.conf.manager, 129
paradrop.conf.network, 130
paradrop.conf.qos, 131
paradrop.conf.wireless, 132
paradrop.core, 158
paradrop.core.agent, 138
paradrop.core.agent.http, 135
paradrop.core.agent.reporting, 137
paradrop.core.agent.wamp_session, 137
paradrop.core.chute, 140
paradrop.core.chute.chute, 138
paradrop.core.chute.chute_storage, 139
paradrop.core.chute.restart, 139
paradrop.core.config, 148
paradrop.core.config.airshark, 140
paradrop.core.config.configservice, 140
paradrop.core.config.devices, 140
paradrop.core.config.dhcp, 142
paradrop.core.config.dockerconfig, 143
paradrop.core.config.firewall, 143
paradrop.core.config.haproxy, 143
paradrop.core.config.hostconfig, 143
paradrop.core.config.network, 144
paradrop.core.config.osconfig, 146
paradrop.core.config.power, 146
paradrop.core.config.reservations, 146
paradrop.core.config.resource, 147
paradrop.core.config.services, 147
paradrop.core.config.snap, 147
paradrop.core.config.state, 147
paradrop.core.config.uciutils, 147
paradrop.core.config.wifi, 148
paradrop.core.config.zerotier, 148
paradrop.core.container, 152
paradrop.core.container.chutecontainer, 148
paradrop.core.container.dockerapi, 149
paradrop.core.container.dockerfile, 151
paradrop.core.container.downloader, 151
paradrop.core.container.log_provider, 152
paradrop.core.plan, 155
paradrop.core.plan.executionplan, 152
paradrop.core.plan.hostconfig, 153

- paradrop.core.plan.name, [153](#)
- paradrop.core.plan.plangraph, [153](#)
- paradrop.core.plan.resource, [154](#)
- paradrop.core.plan.router, [154](#)
- paradrop.core.plan.runtime, [154](#)
- paradrop.core.plan.snap, [154](#)
- paradrop.core.plan.state, [155](#)
- paradrop.core.plan.struct, [155](#)
- paradrop.core.plan.traffic, [155](#)
- paradrop.core.system, [156](#)
- paradrop.core.system.system_info, [155](#)
- paradrop.core.system.system_status, [156](#)
- paradrop.core.update, [158](#)
- paradrop.core.update.update_fetcher, [156](#)
- paradrop.core.update.update_manager, [157](#)
- paradrop.core.update.update_object, [157](#)
- paradrop.lib, [165](#)
- paradrop.lib.misc, [160](#)
- paradrop.lib.misc.pdinstall, [158](#)
- paradrop.lib.misc.procmon, [159](#)
- paradrop.lib.misc.resopt, [159](#)
- paradrop.lib.misc.snapd, [159](#)
- paradrop.lib.misc.ssh_keys, [160](#)
- paradrop.lib.utils, [165](#)
- paradrop.lib.utils.addresses, [160](#)
- paradrop.lib.utils.datastruct, [161](#)
- paradrop.lib.utils.pd_storage, [161](#)
- paradrop.lib.utils.pdos, [162](#)
- paradrop.lib.utils.pdosq, [163](#)
- paradrop.lib.utils.uci, [163](#)
- paradrop.lib.utils.uhttp, [165](#)
- paradrop.main, [165](#)
- paradrop.plan_demo, [165](#)

Symbols

-claim <claim>
 pdtools-cloud-create-node command line option, 67
 pdtools-routers-create command line option, 92
 -force
 pdtools-chute-add-wifi-ap command line option, 65
 -orphaned, -not-orphaned
 pdtools-cloud-create-node command line option, 67
 pdtools-routers-create command line option, 92
 -password <password>
 pdtools-chute-add-wifi-ap command line option, 65
 -public, -not-public
 pdtools-store-register command line option, 95
 -server <server>
 pdtools-device-provision command line option, 77
 -user <user>
 pdtools-device-sshkeys command line option, 78
 -wamp <wamp>
 pdtools-device-provision command line option, 77
 -c, -controller <controller>
 pdtools-node-provision command line option, 87
 -d, -directory <directory>
 pdtools-node-install-chute command line option, 84
 pdtools-node-update-chute command line option, 91
 -f, -format <format>
 pdtools-chute-export-configuration command line option, 66
 pdtools-node-export-configuration command line option, 83
 pdtools-node-generate-configuration command line option, 83
 -t, -target <target>
 pdtools-node command line option, 80
 -u, -user <user>
 pdtools-node-import-ssh-key command line option, 84
 pdtools-node-list-ssh-keys command line option, 86
 -v, -version <version>
 pdtools-store-install-chute command line option, 94

-w, -wamp <wamp>
 pdtools-node-provision command line option, 87

A

abortCreateVolumeDirs() (in module
 paradrop.core.config.dockerconfig), 143
 abortNetworkConfig() (in module
 paradrop.core.config.network), 144
 abortPlans() (in module
 paradrop.core.plan.executionplan), 152
 add() (DeviceReservations method), 146
 add() (InterfaceReservationSet method), 146
 add() (SubnetReservationSet method), 146
 add() (UCIBuilder method), 141
 add_analyzer_observer() (AirsharkManager method), 97
 add_message_observer() (UpdateObject method), 157
 add_observer() (AirsharkInterfaceManager method), 140
 add_spectrum_observer() (AirsharkManager method), 97
 add_update() (UpdateManager method), 157
 add_user() (PasswordManager method), 114
 addAuthorizedKey() (in module
 paradrop.lib.misc.ssh_keys), 160
 addConfig() (UCIConfig method), 163
 addConfigs() (UCIConfig method), 163
 addMap() (PlanMap method), 153
 addPlans() (PlanMap method), 153
 ADDRESS
 pdtools-device command line option, 70
 addToBridge() (ConfigInterface method), 130
 aggregatePlans() (in module
 paradrop.core.plan.executionplan), 153
 airshark_analyzer() (HttpServer method), 110
 airshark_spectrum() (HttpServer method), 110
 AirsharkAnalyzerFactory (class in
 paradrop.backend.airshark_ws), 99
 AirsharkAnalyzerProtocol (class in
 paradrop.backend.airshark_ws), 99
 AirsharkApi (class in paradrop.backend.airshark_api), 99
 AirsharkInterfaceManager (class in
 paradrop.core.config.airshark), 140

AirsharkManager (class in `paradrop.airshark.airshark`), 97
 AirsharkSpectrumFactory (class in `paradrop.backend.airshark_ws`), 99
 AirsharkSpectrumProtocol (class in `paradrop.backend.airshark_ws`), 100
 allocate() (in module `paradrop.lib.misc.resopt`), 159
 allowedActions (ProcessMonitor attribute), 159
 AnalyzerProcessProtocol (class in `paradrop.airshark.analyzer`), 97
 annotate_routes() (in module `paradrop.backend.http_server`), 110
 ANY_PROTO (ConfigRedirect attribute), 127
 api_airshark() (HttpServer method), 110
 api_audio() (HttpServer method), 110
 api_auth() (HttpServer method), 110
 api_changes() (HttpServer method), 110
 api_chute() (HttpServer method), 110
 api_configuration() (HttpServer method), 110
 api_information() (HttpServer method), 110
 api_password() (HttpServer method), 110
 app (HttpServer attribute), 110
 append() (CommandList method), 125
 appendCache() (Chute method), 138
 apply() (ConfigClassify method), 131
 apply() (ConfigDefaults method), 126
 apply() (ConfigDnsmasq method), 126
 apply() (ConfigForwarding method), 127
 apply() (ConfigInterface method), 130, 132
 apply() (ConfigObject method), 123
 apply() (ConfigRedirect method), 127
 apply() (ConfigRule method), 128
 apply() (ConfigWifiDevice method), 132
 apply() (ConfigWifiIface method), 133
 apply() (ConfigZone method), 128
 applyWaitOnlineFix() (in module `paradrop.core.config.osconfig`), 146
 assign_change_id() (UpdateManager method), 157
 attach() (LogProvider method), 152
 attrSaveable() (ChuteStorage method), 139
 attrSaveable() (PDStorage method), 161
 AttrWrapper (class in `paradrop.base.nexus`), 116
 AuthApi (class in `paradrop.backend.auth`), 100
 AuthenticationError, 116

B

backup() (UCIConfig method), 163
 BaseClientFactory (class in `paradrop.base.cxbr`), 115
 basename() (in module `paradrop.lib.utils.pdos`), 162
 BaseOutput (class in `paradrop.base.output`), 118
 BaseSession (class in `paradrop.base.cxbr`), 115
 BaseSessionFactory (class in `paradrop.base.cxbr`), 116
 blacklist (TwistedOutput attribute), 120
 build() (`paradrop.conf.d.base.ConfigObject` class method), 123

build_host_config() (in module `paradrop.core.container.dockerapi`), 149
 buildImage() (in module `paradrop.core.container.dockerapi`), 149
 buildProtocol() (AirsharkAnalyzerFactory method), 99
 buildProtocol() (AirsharkSpectrumFactory method), 99
 buildProtocol() (LogSockJSFactory method), 113
 buildProtocol() (StatusSockJSFactory method), 114

C

call() (BaseSession method), 115
 call_in_netns() (in module `paradrop.core.container.dockerapi`), 149
 call_retry() (in module `paradrop.core.container.dockerapi`), 149
 change() (PasswordApi method), 113
 CHANGE_ID
 pdtools-device-watch command line option, 79
 pdtools-node-watch-change-logs command line option, 91
 change_password() (PasswordManager method), 114
 change_stream() (HttpServer method), 110
 changingSet() (ConfigManager method), 129
 CHANNEL_VOLUME
 pdtools-device-audio-sink-volume command line option, 71
 pdtools-device-audio-source-volume command line option, 72
 check() (in module `paradrop.base.pdutils`), 121
 check() (ProcessMonitor method), 159
 check_auth() (in module `paradrop.backend.auth`), 100
 check_log() (LogSockJSProtocol method), 113
 check_spectrum() (AirsharkManager method), 97
 checkPhyExists() (in module `paradrop.lib.utils.addresses`), 160
 checkSystemDevices() (in module `paradrop.core.config.devices`), 141
 childDataReceived() (AnalyzerProcessProtocol method), 97
 chooseExternalIntf() (in module `paradrop.core.config.network`), 144
 chooseSubnet() (in module `paradrop.core.config.network`), 144

CHUTE

pdtools-device-chute command line option, 72
 pdtools-node-describe-chute command line option, 81
 pdtools-node-describe-chute-cache command line option, 81
 pdtools-node-describe-chute-configuration command line option, 81
 pdtools-node-describe-chute-network-client command line option, 82

- pdtools-node-edit-chute-configuration command line option, 82
- pdtools-node-edit-chute-variables command line option, 83
- pdtools-node-list-chute-network-clients command line option, 85
- pdtools-node-list-chute-networks command line option, 86
- pdtools-node-open-chute-shell command line option, 87
- pdtools-node-remove-chute command line option, 88
- pdtools-node-remove-chute-network-client command line option, 88
- pdtools-node-restart-chute command line option, 89
- pdtools-node-start-chute command line option, 90
- pdtools-node-stop-chute command line option, 90
- pdtools-node-watch-chute-logs command line option, 91
- pdtools-store-install-chute command line option, 94
- Chute (class in paradrop.core.chute.chute), 138
- chute_logs() (HttpServer method), 110
- ChuteApi (class in paradrop.backend.chute_api), 100
- ChuteCacheEncoder (class in paradrop.backend.chute_api), 106
- chuteConfigsMatch() (in module paradrop.lib.utils.uci), 164
- ChuteContainer (class in paradrop.core.container.chutecontainer), 148
- chuteList (ChuteStorage attribute), 139
- ChuteNotFound, 116
- ChuteNotRunning, 116
- ChuteStorage (class in paradrop.core.chute.chute_storage), 139
- cleanup_net_interfaces() (in module paradrop.core.container.dockerapi), 149
- clear() (PasswordApi method), 114
- clear_update_list() (UpdateManager method), 157
- clearChuteStorage() (ChuteStorage method), 139
- CLIENT
 - pdtools-node-describe-chute-network-client command line option, 82
 - pdtools-node-remove-chute-network-client command line option, 88
- clientConnectionFailed() (BaseClientFactory method), 115
- clientConnectionLost() (BaseClientFactory method), 115
- cmd_chanscan() (Scanner method), 98
- cmd_disable() (Scanner method), 98
- cmd_set_samplecount() (Scanner method), 98
- cmd_set_short_repeat() (Scanner method), 98
- code_pattern (CurlRequestDriver attribute), 135
- Command (class in paradrop.conf.command), 125
- CommandList (class in paradrop.conf.command), 125
- commands() (CommandList method), 125
- complete() (UpdateObject method), 157
- compute_hfsc_params() (in module paradrop.conf.qos), 132
- computeResourceAllocation() (in module paradrop.core.config.resource), 147
- ConfigGenerator (class in paradrop.conf.wireless), 132
- config_cors() (in module paradrop.backend.cors), 110
- CONFIG_FIELDS (Chute attribute), 138
- ConfigApi (class in paradrop.backend.config_api), 107
- ConfigClass (class in paradrop.conf.qos), 131
- ConfigClassgroup (class in paradrop.conf.qos), 131
- ConfigClassify (class in paradrop.conf.qos), 131
- ConfigDefaults (class in paradrop.conf.firewall), 126
- ConfigDhcp (class in paradrop.conf.dhcp), 126
- ConfigDnsmasq (class in paradrop.conf.dhcp), 126
- ConfigDomain (class in paradrop.conf.dhcp), 126
- ConfigForwarding (class in paradrop.conf.firewall), 127
- ConfigInterface (class in paradrop.conf.network), 130
- ConfigInterface (class in paradrop.conf.qos), 132
- ConfigManager (class in paradrop.conf.manager), 129
- ConfigObject (class in paradrop.conf.base), 122
- ConfigOption (class in paradrop.conf.base), 124
- ConfigRedirect (class in paradrop.conf.firewall), 127
- ConfigRule (class in paradrop.conf.firewall), 128
- configure() (in module paradrop.core.config.airshark), 140
- configure() (in module paradrop.core.config.zerotier), 148
- configure_telemetry() (in module paradrop.core.config.services), 147
- ConfigWifiDevice (class in paradrop.conf.wireless), 132
- ConfigWifiIface (class in paradrop.conf.wireless), 133
- ConfigZone (class in paradrop.conf.firewall), 128
- connect() (NexusBase method), 117
- connect() (SnapdClient method), 159
- connect() (UHTTPConnection method), 165
- connectionLost() (JSONReceiver method), 135
- connectionLost() (LogSockJSProtocol method), 113
- connectionLost() (StatusSockJSProtocol method), 114
- connectionMade() (AnalyzerProcessProtocol method), 98
- connectionMade() (LogSockJSProtocol method), 113
- connectionMade() (StatusSockJSProtocol method), 114
- convertUnicode() (in module paradrop.base.pdutils), 121
- copy() (ConfigObject method), 123
- copy() (in module paradrop.lib.utils.pdos), 162
- copytree() (in module paradrop.lib.utils.pdos), 162
- count() (DeviceReservations method), 146
- create_chute() (ChuteApi method), 100
- createDefaultInfo() (in module paradrop.base.nexus), 117
- createVolumeDirs() (in module paradrop.core.config.dockerconfig), 143
- curl (CurlRequestDriver attribute), 135

CurlRequestDriver (class in paradrop.core.agent.http), 135

D

dataReceived() (JSONReceiver method), 135
 dataReceived() (StatusSockJSProtocol method), 115
 debugfs_dir (Scanner attribute), 98
 decode() (SpectrumReader static method), 98
 default (ConfigOption attribute), 124
 default() (ChuteCacheEncoder method), 107
 default() (UpdateEncoder method), 107
 DEFAULT_PASSWORD (PasswordManager attribute), 114
 DEFAULT_USER_NAME (PasswordManager attribute), 114
 delCache() (Chute method), 138
 delConfig() (UCIConfig method), 164
 delConfigs() (UCIConfig method), 164
 delete_chute() (ChuteApi method), 100
 delete_station() (ChuteApi method), 100
 deleteChute() (ChuteStorage method), 139
 detach() (LogProvider method), 152
 detectPrimaryInterface() (ConfigWifiDevice method), 132
 detectSystemDevices() (in module paradrop.core.config.devices), 141
 DEV_PLUS_VID (ConfigInterface attribute), 130
 dev_to_phy() (Scanner method), 98
 DeviceNotFoundException, 116
 DeviceReservations (class in paradrop.core.config.reservations), 146
 dict2obj (class in paradrop.base.pdutils), 121
 do_snapt_request() (SnaptResource method), 114
 Dockerfile (class in paradrop.core.container.dockerfile), 151
 download() (Downloader method), 151
 download() (GithubDownloader method), 151
 download() (WebDownloader method), 151
 Downloader (class in paradrop.core.container.downloader), 151
 downloader() (in module paradrop.core.container.downloader), 152
 dump() (ConfigObject method), 123
 dumpCache() (Chute method), 138

E

EMAIL

pdtools-device-snapd-createuser command line option, 78
 pdtools-node-create-user command line option, 80
 endLogging() (Output method), 119
 ensureReady() (ProcessMonitor method), 159
 ERR (Level attribute), 118
 ErrorCommand (class in paradrop.conf.d.command), 125

ESSID

pdtools-chute-add-wifi-ap command line option, 66
 ExceptionOutput (class in paradrop.base.output), 118
 execute() (Command method), 125
 execute() (ConfigManager method), 129
 execute() (ErrorCommand method), 125
 execute() (KillCommand method), 125
 execute() (UpdateObject method), 157
 executePlans() (in module paradrop.core.plan.executionplan), 153
 exists() (in module paradrop.lib.utils.pdos), 162
 existsConfig() (UCIConfig method), 164
 explode() (in module paradrop.base.pdutils), 121
 exportAttr() (PDStorage method), 161
 extract() (Downloader method), 151
 extract_tarred_chute() (in module paradrop.backend.chute_api), 107

F

factory_reset() (ConfigApi method), 108
 FATAL (Level attribute), 118
 feedSpectrumData() (AnalyzerProcessProtocol method), 98
 fetch() (Downloader method), 151
 FILES (UCIBuilder attribute), 141
 find_change() (UpdateManager method), 157
 findByType() (ConfigObject method), 123
 findConfigFiles() (in module paradrop.conf.d.manager), 130
 findMatchingConfig() (ConfigManager method), 129
 findMatchingInterface() (in module paradrop.core.config.firewall), 143
 fixpath() (in module paradrop.lib.utils.pdos), 162
 flush() (OutputRedirect method), 120
 flush() (SpectrumReader method), 98
 flushWirelessInterfaces() (in module paradrop.core.config.devices), 141
 formatOutput() (BaseOutput method), 118
 freqlist (Scanner attribute), 98
 fulfillDeviceRequest() (in module paradrop.core.config.network), 144

G

generate() (HostapdConfGenerator method), 134
 generate() (WpaSupplicantConfGenerator method), 134
 generateConfigSections() (in module paradrop.core.config.haproxy), 143
 generateHostConfig() (in module paradrop.core.config.hostconfig), 144
 generatePlans() (in module paradrop.core.plan.executionplan), 153
 generatePlans() (in module paradrop.core.plan.hostconfig), 153

- generatePlans() (in module paradrop.core.plan.name), 153
- generatePlans() (in module paradrop.core.plan.resource), 154
- generatePlans() (in module paradrop.core.plan.router), 154
- generatePlans() (in module paradrop.core.plan.runtime), 154
- generatePlans() (in module paradrop.core.plan.snap), 154
- generatePlans() (in module paradrop.core.plan.state), 155
- generatePlans() (in module paradrop.core.plan.struct), 155
- generatePlans() (in module paradrop.core.plan.traffic), 155
- generateToken() (in module paradrop.core.config.dockerconfig), 143
- get() (PDServerRequest method), 136
- get11acOptions() (HostapdConfGenerator method), 134
- get11nOptions() (HostapdConfGenerator method), 134
- get11rOptions() (HostapdConfGenerator method), 134
- get_allowed_bearer() (in module paradrop.backend.auth), 100
- get_auth_token() (in module paradrop.core.config.zerotier), 148
- get_change() (SnapdClient method), 159
- get_chute() (ChuteApi method), 100
- get_chute_cache() (ChuteApi method), 101
- get_chute_config() (ChuteApi method), 101
- get_chutes() (ChuteApi method), 101
- get_cipher_list() (in module paradrop.conf.d.wireless), 134
- get_class_id() (ConfigClassgroup method), 131
- get_current_phy_conf() (in module paradrop.core.config.network), 145
- get_debugfs_dir() (Scanner method), 98
- get_environment() (InformationApi method), 111
- get_features() (InformationApi method), 111
- get_hostapd_status() (ChuteApi method), 102
- get_hostconfig() (ConfigApi method), 108
- get_iptables() (ConfigDefaults method), 127
- get_iptables() (ConfigRule method), 128
- get_iptables() (ConfigZone method), 128
- get_leases() (ChuteApi method), 102
- get_logs() (LogProvider method), 152
- get_network() (ChuteApi method), 103
- get_networks() (ChuteApi method), 103
- get_networks() (in module paradrop.core.config.zerotier), 148
- get_pdid() (ConfigApi method), 108
- get_provision() (ConfigApi method), 108
- get_ssid() (ChuteApi method), 104
- get_station() (ChuteApi method), 104
- get_stations() (ChuteApi method), 105
- get_telemetry() (InformationApi method), 112
- get_username_password() (in module paradrop.backend.auth), 100
- getAddress() (in module paradrop.core.config.zerotier), 148
- getAttr() (ChuteStorage method), 139
- getAuthorizedKeys() (in module paradrop.lib.misc.ssh_keys), 160
- getBridgeGateway() (in module paradrop.core.container.dockerapi), 149
- getBytesIO() (Dockerfile method), 151
- getCache() (Chute method), 138
- getCacheContents() (Chute method), 138
- getChute() (ChuteStorage method), 139
- getChuteConfigs() (UCIConfig method), 164
- getChuteList() (ChuteStorage method), 139
- getConfig() (UCIConfig method), 164
- getConfigIgnoreComments() (UCIConfig method), 164
- getConfiguration() (Chute method), 138
- getDeveloperFirewallRules() (in module paradrop.core.config.firewall), 143
- getDeviceId() (SysReader method), 140
- getDeviceReservations() (in module paradrop.core.config.reservations), 146
- getDMI() (in module paradrop.core.system.system_info), 155
- getExtraOptions() (in module paradrop.core.config.network), 144
- getFileType() (in module paradrop.lib.utils.pdos), 162
- getGatewayIntf() (in module paradrop.lib.utils.addresses), 160
- getHostConfig() (Chute method), 138
- getHostConfig() (in module paradrop.core.config.hostconfig), 144
- getID() (ChuteContainer method), 148
- getIface() (ConfigWifiIface method), 133
- getImageName() (in module paradrop.core.container.dockerapi), 149
- getInterfaceAddress() (in module paradrop.core.config.network), 144
- getInterfaceDict() (in module paradrop.core.config.network), 145
- getInterfaceReservations() (in module paradrop.core.config.reservations), 146
- getInternalIntfList() (in module paradrop.lib.utils.addresses), 160
- getIP() (ChuteContainer method), 148
- getKey() (NexusBase method), 117
- getL3BridgeConfig() (in module paradrop.core.config.network), 145
- getLineParts() (in module paradrop.lib.utils.uci), 164
- getLogsSince() (Output method), 119
- getMACAddress() (in module paradrop.core.config.devices), 141
- getMainOptions() (HostapdConfGenerator method), 134

- getMainOptions() (WpaSupplicantConfGenerator method), 134
- getModule() (paradrop.conf.d.base.ConfigObject class method), 123
- getMountCmd() (in module paradrop.lib.utils.pdos), 162
- getName() (ConfigDefaults method), 127
- getName() (ConfigDomain method), 126
- getName() (ConfigObject method), 123
- getName() (ConfigWifiIface method), 133
- getNetworkConfig() (in module paradrop.core.config.network), 145
- getNetworkConfigLan() (in module paradrop.core.config.network), 145
- getNetworkConfigVlan() (in module paradrop.core.config.network), 145
- getNetworkConfigWifi() (in module paradrop.core.config.network), 145
- getNetworkInfo() (paradrop.core.system.system_status.SystemStatus class method), 156
- getNextAbort() (PlanMap method), 154
- getNextTodo() (PlanMap method), 154
- getOSFirewallRules() (in module paradrop.core.config.firewall), 143
- getOSNetworkConfig() (in module paradrop.core.config.network), 145
- getOSVersion() (in module paradrop.core.system.system_info), 155
- getOSWirelessConfig() (in module paradrop.core.config.wifi), 148
- getPackageVersion() (in module paradrop.core.system.system_info), 155
- getPhyFromMAC() (in module paradrop.conf.d.wireless), 134
- getPhyMACAddress() (in module paradrop.conf.d.wireless), 134
- getPhyMACAddress() (in module paradrop.core.config.devices), 141
- getPID() (ChuteContainer method), 148
- getPid() (KillCommand method), 125
- getPortConfiguration() (ChuteContainer method), 148
- getPortList() (in module paradrop.core.container.dockerapi), 149
- getPreviousCommands() (ConfigManager method), 129
- getProcessInfo() (paradrop.core.system.system_status.SystemStatus class method), 156
- getRadiusOptions() (HostapdConfGenerator method), 134
- getRandomMAC() (ConfigWifiIface method), 133
- getReservations() (in module paradrop.core.config.reservations), 147
- getResourceAllocation() (in module paradrop.core.config.resource), 147
- getSections() (UCIBuilder method), 141
- getSecurityOptions() (HostapdConfGenerator method), 134
- getServerInfo() (paradrop.core.agent.http.PDServerRequest class method), 136
- getSlotName() (SysReader method), 140
- getStatus() (ChuteContainer method), 149
- getStatus() (SystemStatus method), 156
- getString() (Dockerfile method), 151
- getSubnet() (in module paradrop.lib.utils.addresses), 160
- getSubnetReservations() (in module paradrop.core.config.reservations), 147
- getSystemConfigDir() (in module paradrop.lib.utils.uci), 164
- getSystemDevices() (in module paradrop.core.config.devices), 141
- getSystemInfo() (paradrop.core.system.system_status.SystemStatus class method), 156
- getSystemPath() (in module paradrop.lib.utils.uci), 164
- getSystemTypeAndName() (ConfigObject method), 123
- getValue() (in module paradrop.lib.utils.datastruct), 161
- getVendorId() (SysReader method), 141
- getVirtDHCPSettings() (in module paradrop.core.config.dhcp), 142
- getVirtPreamble() (in module paradrop.core.config.dockerconfig), 143
- getWANIntf() (in module paradrop.lib.utils.addresses), 160
- getWebPort() (Chute method), 138
- getWifiKeySettings() (in module paradrop.core.config.network), 145
- getWirelessPhyName() (in module paradrop.core.config.devices), 141
- GithubDownloader (class in paradrop.core.container.downloader), 151
- GROUP
 - pdtools-cloud-group-add-node command line option, 69
- GROUP_ID
 - pdtools-group command line option, 79
- H
 - handleMissingWiFi() (in module paradrop.core.config.devices), 141
 - handlePrint() (Output method), 119
 - handleWireInfo() (InformationApi method), 112
 - hdrsize (SpectrumReader attribute), 98
 - HEADER (Level attribute), 118
 - header_pattern (CurlRequestDriver attribute), 135
 - home() (HttpServer method), 110
 - hostapd_control() (ChuteApi method), 105
 - HostapdConfGenerator (class in paradrop.conf.d.wireless), 134
 - HTTPRequestDriver (class in paradrop.core.agent.http), 135
 - HTTPResponse (class in paradrop.core.agent.http), 135

HttpServer (class in paradrop.backend.http_server), 110

I

ID

pdtools-node-provision command line option, 87

importAttr() (PDStorage method), 161

incIpaddr() (in module paradrop.lib.utils.addresses), 160

INCLUDED_PARTITIONS (SystemStatus attribute), 156

increaseDelay() (ReportSender method), 137

INFO (Level attribute), 118

InformationApi (class in paradrop.backend.information_api), 111

initialDelay (BaseClientFactory attribute), 115

inspect() (ChuteContainer method), 149

installSnap() (SnapdClient method), 159

InternalException, 116

interface (Scanner attribute), 98

interface_available() (AirsharkInterfaceManager method), 140

InterfaceReservationSet (class in paradrop.core.config.reservations), 146

interpretBoolean() (in module paradrop.conf.d.base), 124

InvalidCredentials, 116

isdir() (in module paradrop.lib.utils.pdos), 162

isfile() (in module paradrop.lib.utils.pdos), 162

isHexString() (in module paradrop.conf.d.wireless), 134

isIpAvailable() (in module paradrop.lib.utils.addresses), 160

isIpValid() (in module paradrop.lib.utils.addresses), 161

isLeaf (SnapdResource attribute), 114

isMatch() (in module paradrop.lib.utils.uci), 164

isMatchIgnoreComments() (in module paradrop.lib.utils.uci), 164

isMount() (in module paradrop.lib.utils.pdos), 162

ismount() (in module paradrop.lib.utils.pdos), 162

isRunning() (AnalyzerProcessProtocol method), 98

isRunning() (Chute method), 138

isRunning() (ChuteContainer method), 149

isStaticIpAvailable() (in module paradrop.lib.utils.addresses), 161

isValid() (Chute method), 138

isValid() (Dockerfile method), 151

isVirtual() (in module paradrop.core.config.devices), 142

isWAN() (in module paradrop.core.config.devices), 142

isWifiSSIDAvailable() (in module paradrop.lib.utils.addresses), 161

isWireless() (in module paradrop.core.config.devices), 142

J

json2str() (in module paradrop.base.pdutils), 121

jsonPretty() (in module paradrop.base.pdutils), 121

JSONReceiver (class in paradrop.core.agent.http), 135

K

KEY

pdtools-node-provision command line option, 87

kill() (in module paradrop.conf.d.command), 125

KillCommand (class in paradrop.conf.d.command), 125

L

leave() (BaseSession method), 115

Level (class in paradrop.base.output), 118

listdir() (in module paradrop.lib.utils.pdos), 162

listen() (in module paradrop.conf.d.main), 129

listSnaps() (SnapdClient method), 160

listSystemDevices() (in module paradrop.core.config.devices), 142

listWiFiDevices() (in module paradrop.core.config.devices), 142

load() (in module paradrop.core.config.hostconfig), 144

loadConfig() (ConfigManager method), 129

loadFromDisk() (PDStorage method), 161

loadPriorityMap() (in module paradrop.plan_demo), 165

loadSettings() (in module paradrop.base.settings), 122

loadYaml() (in module paradrop.base.nexus), 117

local_login() (AuthApi method), 100

lock (CurlRequestDriver attribute), 135

LogProvider (class in paradrop.core.container.log_provider), 152

logs() (HttpServer method), 110

LogSockJSFactory (class in paradrop.backend.log_sockjs), 113

LogSockJSProtocol (class in paradrop.backend.log_sockjs), 113

logToConsole() (Output method), 119

lookup() (ConfigObject method), 123

M

main() (in module paradrop.main), 165

make_iptables_cmd() (ConfigClassify method), 131

makedirs() (in module paradrop.lib.utils.pdosq), 163

makeHostapdConf() (ConfigWifiIface method), 133

makeWpaSupplicantConf() (ConfigWifiIface method), 133

manage_network() (in module paradrop.core.config.zerotier), 148

maskable (ConfigInterface attribute), 130

maskable (ConfigObject attribute), 123

maxDelay (BaseClientFactory attribute), 115

maxIpaddr() (in module paradrop.lib.utils.addresses), 161

messageToString() (Output method), 119

meta() (Downloader method), 151

meta() (GithubDownloader method), 151

meta() (WebDownloader method), 151

mkdir() (in module paradrop.lib.utils.pdos), 162

ModelNotFound, 116

MODULE

pdtools-node-load-audio-module command line option, 86
 monitor_logs() (in module paradrop.core.container.log_provider), 152
 move() (in module paradrop.lib.utils.pdos), 162

N

NAME

pdtools-cloud-create-node command line option, 67
 pdtools-cloud-delete-node command line option, 68
 pdtools-cloud-describe-node command line option, 68
 pdtools-cloud-edit-node-description command line option, 68
 pdtools-cloud-rename-node command line option, 70
 pdtools-device-audio-load-module command line option, 71
 pdtools-routers-create command line option, 92
 pdtools-store-describe-chute command line option, 93
 pdtools-store-list-versions command line option, 94
 name (ConfigOption attribute), 124

NETWORK

pdtools-device-chute-network command line option, 73
 pdtools-node-describe-chute-network-client command line option, 82
 pdtools-node-list-chute-network-clients command line option, 85
 pdtools-node-remove-chute-network-client command line option, 88
 new_config() (ConfigApi method), 108

NEW_NAME

pdtools-cloud-rename-node command line option, 70

nextId (ConfigObject attribute), 123
 nextInterfaceName() (ConfigWifiDevice method), 132

Nexus (class in paradrop.main), 165

NexusBase (class in paradrop.base.nexus), 117

NODE

pdtools-cloud-group-add-node command line option, 69
 pdtools-store-install-chute command line option, 94

O

on_analyzer_message() (AirsharkAnalyzerProtocol method), 99
 on_analyzer_message() (AirsharkManager method), 97
 on_interface_down() (AirsharkManager method), 97
 on_interface_up() (AirsharkManager method), 97
 on_spectrum_data() (AirsharkSpectrumProtocol method), 100

onChallenge() (WampSession method), 137
 onClose() (AirsharkAnalyzerProtocol method), 99
 onClose() (AirsharkSpectrumProtocol method), 100
 onConnect() (WampSession method), 137
 onDisconnect() (WampSession method), 137
 onInfoChange() (NexusBase method), 117
 onJoin() (BaseSession method), 115
 onJoin() (WampSession method), 137
 onLeave() (WampSession method), 137
 onOpen() (AirsharkAnalyzerProtocol method), 99
 onOpen() (AirsharkSpectrumProtocol method), 100
 onStart() (Nexus method), 165
 onStart() (NexusBase method), 117
 onStop() (Nexus method), 165
 onStop() (NexusBase method), 117
 open() (in module paradrop.lib.utils.pdos), 162
 OPTION

pdtools-device-hostconfig-change command line option, 76
 options (ConfigClass attribute), 131
 options (ConfigClassgroup attribute), 131
 options (ConfigClassify attribute), 131
 options (ConfigDefaults attribute), 127
 options (ConfigDhcp attribute), 126
 options (ConfigDnsmasq attribute), 126
 options (ConfigDomain attribute), 126
 options (ConfigForwarding attribute), 127
 options (ConfigInterface attribute), 130, 132
 options (ConfigObject attribute), 123
 options (ConfigRedirect attribute), 128
 options (ConfigRule attribute), 128
 options (ConfigWifiDevice attribute), 132
 options (ConfigWifiIface attribute), 133
 options (ConfigZone attribute), 128
 optionsMatch() (ConfigObject method), 123
 oscall() (in module paradrop.lib.utils.pdos), 162
 Output (class in paradrop.base.output), 118
 OutputRedirect (class in paradrop.base.output), 120

P

paradrop (module), 165
 paradrop.airshark (module), 99
 paradrop.airshark.airshark (module), 97
 paradrop.airshark.analyzer (module), 97
 paradrop.airshark.scanner (module), 98
 paradrop.airshark.spectrum_reader (module), 98
 paradrop.backend (module), 115
 paradrop.backend.airshark_api (module), 99
 paradrop.backend.airshark_ws (module), 99
 paradrop.backend.auth (module), 100
 paradrop.backend.chute_api (module), 53, 100
 paradrop.backend.config_api (module), 59, 107
 paradrop.backend.cors (module), 110
 paradrop.backend.http_server (module), 110

- paradrop.backend.information_api (module), 61, 111
- paradrop.backend.log_sockjs (module), 113
- paradrop.backend.password_api (module), 113
- paradrop.backend.password_manager (module), 114
- paradrop.backend.snapd_resource (module), 114
- paradrop.backend.status_sockjs (module), 114
- paradrop.base (module), 122
- paradrop.base.cxbr (module), 115
- paradrop.base.exceptions (module), 116
- paradrop.base.nexus (module), 116
- paradrop.base.output (module), 118
- paradrop.base.pdutils (module), 121
- paradrop.base.settings (module), 122
- paradrop.conf.d (module), 135
- paradrop.conf.d.base (module), 122
- paradrop.conf.d.client (module), 125
- paradrop.conf.d.command (module), 125
- paradrop.conf.d.dhcp (module), 126
- paradrop.conf.d.firewall (module), 126
- paradrop.conf.d.main (module), 129
- paradrop.conf.d.manager (module), 129
- paradrop.conf.d.network (module), 130
- paradrop.conf.d.qos (module), 131
- paradrop.conf.d.wireless (module), 132
- paradrop.core (module), 158
- paradrop.core.agent (module), 138
- paradrop.core.agent.http (module), 135
- paradrop.core.agent.reporting (module), 137
- paradrop.core.agent.wamp_session (module), 137
- paradrop.core.chute (module), 140
- paradrop.core.chute.chute (module), 138
- paradrop.core.chute.chute_storage (module), 139
- paradrop.core.chute.restart (module), 139
- paradrop.core.config (module), 148
- paradrop.core.config.airshark (module), 140
- paradrop.core.config.configservice (module), 140
- paradrop.core.config.devices (module), 140
- paradrop.core.config.dhcp (module), 142
- paradrop.core.config.dockerconfig (module), 143
- paradrop.core.config.firewall (module), 143
- paradrop.core.config.haproxy (module), 143
- paradrop.core.config.hostconfig (module), 143
- paradrop.core.config.network (module), 144
- paradrop.core.config.osconfig (module), 146
- paradrop.core.config.power (module), 146
- paradrop.core.config.reservations (module), 146
- paradrop.core.config.resource (module), 147
- paradrop.core.config.services (module), 147
- paradrop.core.config.snap (module), 147
- paradrop.core.config.state (module), 147
- paradrop.core.config.uciutils (module), 147
- paradrop.core.config.wifi (module), 148
- paradrop.core.config.zerotier (module), 148
- paradrop.core.container (module), 152
- paradrop.core.container.chutecontainer (module), 148
- paradrop.core.container.dockerapi (module), 149
- paradrop.core.container.dockerfile (module), 151
- paradrop.core.container.downloader (module), 151
- paradrop.core.container.log_provider (module), 152
- paradrop.core.plan (module), 155
- paradrop.core.plan.executionplan (module), 152
- paradrop.core.plan.hostconfig (module), 153
- paradrop.core.plan.name (module), 153
- paradrop.core.plan.plangraph (module), 153
- paradrop.core.plan.resource (module), 154
- paradrop.core.plan.router (module), 154
- paradrop.core.plan.runtime (module), 154
- paradrop.core.plan.snap (module), 154
- paradrop.core.plan.state (module), 155
- paradrop.core.plan.struct (module), 155
- paradrop.core.plan.traffic (module), 155
- paradrop.core.system (module), 156
- paradrop.core.system.system_info (module), 155
- paradrop.core.system.system_status (module), 156
- paradrop.core.update (module), 158
- paradrop.core.update.update_fetcher (module), 156
- paradrop.core.update.update_manager (module), 157
- paradrop.core.update.update_object (module), 157
- paradrop.lib (module), 165
- paradrop.lib.misc (module), 160
- paradrop.lib.misc.pdinstall (module), 158
- paradrop.lib.misc.procmon (module), 159
- paradrop.lib.misc.resopt (module), 159
- paradrop.lib.misc.snapd (module), 159
- paradrop.lib.misc.ssh_keys (module), 160
- paradrop.lib.utils (module), 165
- paradrop.lib.utils.addresses (module), 160
- paradrop.lib.utils.datastruct (module), 161
- paradrop.lib.utils.pd_storage (module), 161
- paradrop.lib.utils.pdos (module), 162
- paradrop.lib.utils.pdosq (module), 163
- paradrop.lib.utils.uci (module), 163
- paradrop.lib.utils.uhttp (module), 165
- paradrop.main (module), 165
- paradrop.plan_demo (module), 165
- paradrop_logs() (HttpServer method), 110
- ParadropException, 116
- parse() (in module paradrop.core.update.update_object), 158
- parseLogPrefix() (in module paradrop.base.output), 120
- parseValue() (in module paradrop.base.settings), 122
- PasswordApi (class in paradrop.backend.password_api), 113
- PasswordManager (class in paradrop.backend.password_manager), 114
- patch() (PDServerRequest method), 136
- PATH
 - pdtools-chute-set command line option, 66

- pdtools-device-sshkeys-add command line option, 78
- pdtools-node-import-configuration command line option, 84
- pdtools-node-import-ssh-key command line option, 84
- pdtools-node-set-configuration command line option, 89
- PCI_BUS_ID (SysReader attribute), 140
- pdconf() (ConfigApi method), 109
- pdconf_reload() (ConfigApi method), 109
- PDID (NexusBase attribute), 117
- PdidError, 116
- PdidExclusionError, 116
- PdServerException, 116
- PDServerRequest (class in paradrop.core.agent.http), 135
- PDServerResponse (class in paradrop.core.agent.http), 136
- PDStorage (class in paradrop.lib.utils.pd_storage), 161
- pdtools-chute-add-wifi-ap command line option
 - force, 65
 - password <password>, 65
 - ESSID, 66
- pdtools-chute-export-configuration command line option
 - f, -format <format>, 66
- pdtools-chute-set command line option
 - PATH, 66
 - VALUE, 66
- pdtools-cloud-claim-node command line option
 - TOKEN, 67
- pdtools-cloud-create-node command line option
 - claim <claim>, 67
 - orphaned, -not-orphaned, 67
 - NAME, 67
- pdtools-cloud-delete-node command line option
 - NAME, 68
- pdtools-cloud-describe-node command line option
 - NAME, 68
- pdtools-cloud-edit-node-description command line option
 - NAME, 68
- pdtools-cloud-group-add-node command line option
 - GROUP, 69
 - NODE, 69
- pdtools-cloud-rename-node command line option
 - NAME, 70
 - NEW_NAME, 70
- pdtools-device command line option
 - ADDRESS, 70
- pdtools-device-audio-load-module command line option
 - NAME, 71
- pdtools-device-audio-sink command line option
 - SINK_NAME, 71
- pdtools-device-audio-sink-volume command line option
 - CHANNEL_VOLUME, 71
- pdtools-device-audio-source command line option
 - SOURCE_NAME, 72
- pdtools-device-audio-source-volume command line option
 - CHANNEL_VOLUME, 72
- pdtools-device-chute command line option
 - CHUTE, 72
- pdtools-device-chute-network command line option
 - NETWORK, 73
- pdtools-device-chute-network-station command line option
 - STATION, 74
- pdtools-device-hostconfig-change command line option
 - OPTION, 76
 - VALUE, 76
- pdtools-device-provision command line option
 - server <server>, 77
 - wamp <wamp>, 77
 - ROUTER_ID, 77
 - ROUTER_PASSWORD, 77
- pdtools-device-snapd-createuser command line option
 - EMAIL, 78
- pdtools-device-sshkeys command line option
 - user <user>, 78
- pdtools-device-sshkeys-add command line option
 - PATH, 78
- pdtools-device-watch command line option
 - CHANGE_ID, 79
- pdtools-group command line option
 - GROUP_ID, 79
- pdtools-group-add-router command line option
 - ROUTER_ID, 79
- pdtools-node command line option
 - t, -target <target>, 80
- pdtools-node-create-user command line option
 - EMAIL, 80
- pdtools-node-describe-chute command line option
 - CHUTE, 81
- pdtools-node-describe-chute-cache command line option
 - CHUTE, 81
- pdtools-node-describe-chute-configuration command line option
 - CHUTE, 81
- pdtools-node-describe-chute-network-client command line option
 - CHUTE, 82
 - CLIENT, 82
 - NETWORK, 82
- pdtools-node-edit-chute-configuration command line option
 - CHUTE, 82
- pdtools-node-edit-chute-variables command line option
 - CHUTE, 83
- pdtools-node-export-configuration command line option

- f, --format <format>, 83
- pdtools-node-generate-configuration command line option
 - f, --format <format>, 83
- pdtools-node-import-configuration command line option
 - PATH, 84
- pdtools-node-import-ssh-key command line option
 - u, --user <user>, 84
 - PATH, 84
- pdtools-node-install-chute command line option
 - d, --directory <directory>, 84
- pdtools-node-list-chute-network-clients command line option
 - CHUTE, 85
 - NETWORK, 85
- pdtools-node-list-chute-networks command line option
 - CHUTE, 86
- pdtools-node-list-ssh-keys command line option
 - u, --user <user>, 86
- pdtools-node-load-audio-module command line option
 - MODULE, 86
- pdtools-node-open-chute-shell command line option
 - CHUTE, 87
- pdtools-node-provision command line option
 - c, --controller <controller>, 87
 - w, --wamp <wamp>, 87
 - ID, 87
 - KEY, 87
- pdtools-node-remove-chute command line option
 - CHUTE, 88
- pdtools-node-remove-chute-network-client command line option
 - CHUTE, 88
 - CLIENT, 88
 - NETWORK, 88
- pdtools-node-restart-chute command line option
 - CHUTE, 89
- pdtools-node-set-configuration command line option
 - PATH, 89
 - VALUE, 89
- pdtools-node-set-sink-volume command line option
 - SINK, 89
 - VOLUME, 89
- pdtools-node-set-source-volume command line option
 - SOURCE, 90
 - VOLUME, 90
- pdtools-node-start-chute command line option
 - CHUTE, 90
- pdtools-node-stop-chute command line option
 - CHUTE, 90
- pdtools-node-update-chute command line option
 - d, --directory <directory>, 91
- pdtools-node-watch-change-logs command line option
 - CHANGE_ID, 91
- pdtools-node-watch-chute-logs command line option
 - CHUTE, 91
- pdtools-routers-claim command line option
 - TOKEN, 92
- pdtools-routers-create command line option
 - claim <claim>, 92
 - orphaned, --not-orphaned, 92
 - NAME, 92
- pdtools-routers-delete command line option
 - ROUTER_ID, 93
- pdtools-store-describe-chute command line option
 - NAME, 93
- pdtools-store-install-chute command line option
 - v, --version <version>, 94
 - CHUTE, 94
 - NODE, 94
- pdtools-store-list-versions command line option
 - NAME, 94
- pdtools-store-register command line option
 - public, --not-public, 95
- PERF (Level attribute), 118
- pktsize (SpectrumReader attribute), 98
- Plan (class in paradrop.core.plan.plangraph), 153
- PlanMap (class in paradrop.core.plan.plangraph), 153
- pool (TwistedRequestDriver attribute), 136
- post() (PDServerRequest method), 136
- prepare() (StateReportBuilder method), 137
- prepare() (TelemetryReportBuilder method), 137
- prepare_environment() (in module paradrop.core.container.dockerapi), 149
- prepare_port_bindings() (in module paradrop.core.container.dockerapi), 149
- prepareHostConfig() (in module paradrop.core.config.hostconfig), 144
- PrintLogThread (class in paradrop.base.output), 120
- PRIO_CONFIG_IFACE (ConfigObject attribute), 122
- PRIO_CONFIG_QDISC (ConfigObject attribute), 122
- PRIO_CREATE_IFACE (ConfigObject attribute), 122
- PRIO_CREATE_QDISC (ConfigObject attribute), 123
- PRIO_CREATE_VLAN (ConfigObject attribute), 123
- PRIO_IPTABLES_RULE (ConfigObject attribute), 123
- PRIO_IPTABLES_TOP (ConfigObject attribute), 123
- PRIO_IPTABLES_ZONE (ConfigObject attribute), 123
- PRIO_START_DAEMON (ConfigObject attribute), 123
- prioritizeConfigs() (ConfigObject static method), 123
- process (Scanner attribute), 98
- processEnded() (AnalyzerProcessProtocol method), 98
- ProcessMonitor (class in paradrop.lib.misc.procmon), 159
- progress() (UpdateObject method), 158
- provision() (ConfigApi method), 109
- provision() (NexusBase method), 117
- provisioned() (NexusBase method), 117
- publish() (BaseSession method), 115

pull_update() (UpdateFetcher method), 156
 put() (PDServerRequest method), 136

R

read_raw_samples() (AirsharkManager method), 97
 read_samples() (SpectrumReader method), 98
 read_uevent() (SysReader method), 141
 readConfig() (ConfigManager method), 130
 readConfig() (UCIConfig method), 164
 readFile() (in module paradrop.lib.utils.pdos), 162
 readHostconfigVlan() (in module paradrop.core.config.devices), 142
 readHostconfigWifi() (in module paradrop.core.config.devices), 142
 readHostconfigWifiInterfaces() (in module paradrop.core.config.devices), 142
 readMode() (HostapdConfGenerator method), 134
 readSysFile() (in module paradrop.core.config.devices), 142
 reboot() (in module paradrop.core.config.power), 146
 receive() (CurlRequestDriver method), 135
 receive() (TwistedRequestDriver method), 136
 receiveHeaders() (CurlRequestDriver method), 135
 receiveResponse() (PDServerRequest method), 136
 reclaimNetworkResources() (in module paradrop.core.config.network), 145
 reconfigureProxy() (in module paradrop.core.config.haproxy), 143
 refreshCpuLoad() (SystemStatus method), 156
 refreshDiskInfo() (SystemStatus method), 156
 refreshMemoryInfo() (SystemStatus method), 156
 refreshNetworkTraffic() (SystemStatus method), 156
 register() (BaseSession method), 115
 registerSkip() (PlanMap method), 154
 releaseInterfaceName() (ConfigWifiDevice method), 133
 reload() (in module paradrop.conf.d.client), 125
 reloadAll() (in module paradrop.conf.d.client), 125
 reloadAll() (in module paradrop.core.config.configservice), 140
 reloadChutes() (in module paradrop.core.chute.restart), 139
 remove() (in module paradrop.lib.utils.pdos), 162
 remove_analyzer_observer() (AirsharkManager method), 97
 remove_message_observer() (UpdateObject method), 158
 remove_observer() (AirsharkInterfaceManager method), 140
 remove_spectrum_observer() (AirsharkManager method), 97
 remove_user() (PasswordManager method), 114
 removeAllChutes() (in module paradrop.core.config.state), 147

removeAllContainers() (in module paradrop.core.container.dockerapi), 150
 removeChute() (in module paradrop.core.container.dockerapi), 150
 removeFromBridge() (ConfigInterface method), 130
 removeFromParents() (ConfigObject method), 124
 removeNewContainer() (in module paradrop.core.container.dockerapi), 150
 removeNewImage() (in module paradrop.core.container.dockerapi), 150
 removeOldContainer() (in module paradrop.core.container.dockerapi), 150
 removeOldImage() (in module paradrop.core.container.dockerapi), 150
 render() (SnapdResource method), 114
 ReportSender (class in paradrop.core.agent.reporting), 137
 request() (CurlRequestDriver method), 135
 request() (HTTPRequestDriver method), 135
 request() (PDServerRequest method), 136
 request() (TwistedRequestDriver method), 136
 required (ConfigOption attribute), 124
 requiredFields (Dockerfile attribute), 151
 requires_auth() (in module paradrop.backend.auth), 100
 reset() (PasswordManager method), 114
 reset_interface() (AirsharkInterfaceManager method), 140
 resetToken() (paradrop.core.agent.http.PDServerRequest class method), 136
 resetWirelessDevice() (in module paradrop.core.config.devices), 142
 resolveInfo() (in module paradrop.base.nexus), 117
 resolveWirelessDevRef() (in module paradrop.core.config.devices), 142
 restart() (ProcessMonitor method), 159
 restart_chute() (ChuteApi method), 105
 restartChute() (in module paradrop.core.container.dockerapi), 150
 restore() (UCIConfig method), 164
 restoreConfigFile() (in module paradrop.core.config.uciutils), 147
 revert() (ConfigClassify method), 131
 revert() (ConfigDefaults method), 127
 revert() (ConfigDnsmasq method), 126
 revert() (ConfigForwarding method), 127
 revert() (ConfigInterface method), 130, 132
 revert() (ConfigObject method), 124
 revert() (ConfigRedirect method), 128
 revert() (ConfigRule method), 128
 revert() (ConfigWifiDevice method), 133
 revert() (ConfigWifiIface method), 133
 revert() (ConfigZone method), 128
 revertChute() (in module paradrop.core.config.state), 147

- ul style="list-style-type: none; padding-left: 0;">
- revertConfig() (in module paradrop.core.config.osconfig), 146
- revertHostConfig() (in module paradrop.core.config.hostconfig), 144
- revertResourceAllocation() (in module paradrop.core.container.dockerapi), 150
- ROUTER_ID
 - pdtools-device-provision command line option, 77
 - pdtools-group-add-router command line option, 79
 - pdtools-routers-delete command line option, 93
- ROUTER_PASSWORD
 - pdtools-device-provision command line option, 77
- routes (AirsharkApi attribute), 99
- routes (AuthApi attribute), 100
- routes (ChuteApi attribute), 105
- routes (ConfigApi attribute), 109
- routes (InformationApi attribute), 113
- routes (PasswordApi attribute), 114
- run() (PrintLogThread method), 120
- run_thread() (in module paradrop.conf.d.main), 129
- ## S
- satisfies_requirements() (in module paradrop.core.config.network), 145
 - save() (in module paradrop.core.config.hostconfig), 144
 - save() (NexusBase method), 117
 - save() (UCIConfig method), 164
 - saveChute() (ChuteStorage method), 139
 - saveChute() (in module paradrop.core.config.state), 147
 - saveKey() (NexusBase method), 117
 - saveToDisk() (PDStorage method), 161
 - sc_wide (SpectrumReader attribute), 98
 - Scanner (class in paradrop.airshark.scanner), 98
 - SECURITY (Level attribute), 118
 - select_brlan_address() (in module paradrop.core.config.devices), 142
 - select_chute_subnet_pool() (in module paradrop.core.config.network), 145
 - sem (TwistedRequestDriver attribute), 136
 - send() (ReportSender method), 137
 - sendCommand() (in module paradrop.lib.misc.pdinstall), 158
 - sendStateReport() (in module paradrop.core.agent.reporting), 137
 - sendTelemetryReport() (in module paradrop.core.agent.reporting), 137
 - set_chute_config() (ChuteApi method), 105
 - set_freqs() (Scanner method), 98
 - set_interface() (AirsharkInterfaceManager method), 140
 - set_ssid() (ChuteApi method), 106
 - set_update_fetcher() (paradrop.core.agent.wamp_session.WampSession class method), 137
 - setAttr() (ChuteStorage method), 139
 - setAutoUpdate() (in module paradrop.core.config.hostconfig), 144
 - setCache() (Chute method), 138
 - setConfig() (in module paradrop.core.config.devices), 142
 - setConfig() (in module paradrop.core.config.uciutils), 147
 - setHeader() (HTTPRequestDriver method), 135
 - setHostConfig() (in module paradrop.core.config.hostconfig), 144
 - setL3BridgeConfig() (in module paradrop.core.config.network), 145
 - setOnChange() (AttrWrapper method), 117
 - setOSFirewallRules() (in module paradrop.core.config.firewall), 143
 - setOSNetworkConfig() (in module paradrop.core.config.network), 145
 - setOSWirelessConfig() (in module paradrop.core.config.wifi), 148
 - setResourceAllocation() (in module paradrop.core.container.dockerapi), 150
 - setSystemDevices() (in module paradrop.core.config.devices), 142
 - setup() (ConfigClassgroup method), 131
 - setup() (ConfigInterface method), 131
 - setup() (ConfigObject method), 124
 - setup() (ConfigWifiDevice method), 133
 - setup() (ConfigZone method), 128
 - setup_http_server() (in module paradrop.backend.http_server), 111
 - setup_net_interfaces() (in module paradrop.core.container.dockerapi), 150
 - setVirtDHCPSettings() (in module paradrop.core.config.dhcp), 142
 - shutdown() (in module paradrop.core.config.power), 146
 - silentLogPrefix() (in module paradrop.base.output), 120
 - singleConfigMatches() (in module paradrop.lib.utils.uci), 164
 - SINK
 - pdtools-node-set-sink-volume command line option, 89
 - SINK_NAME
 - pdtools-device-audio-sink command line option, 71
 - snappd() (HttpServer method), 110
 - SnappdClient (class in paradrop.lib.misc.snappd), 159
 - SnappdResource (class in paradrop.backend.snappd_resource), 114
 - software_info() (InformationApi method), 113
 - sort() (PlanMap method), 154
 - SOURCE
 - pdtools-node-set-source-volume command line option, 90
 - SOURCE_NAME
 - pdtools-device-audio-source command line option, 72
 - spectrum_reader (Scanner attribute), 98

- SpectrumReader (class in paradrop.airshark.spectrum_reader), 98
- sshKeys() (ConfigApi method), 109
- start() (paradrop.base.cxbr.BaseSession class method), 115
- start() (Scanner method), 98
- start_chute() (ChuteApi method), 106
- start_long_poll() (UpdateFetcher method), 156
- start_polling() (UpdateFetcher method), 156
- start_update() (ConfigApi method), 109
- startChute() (in module paradrop.core.container.dockerapi), 150
- started() (UpdateObject method), 158
- startLogging() (Output method), 119
- startOldContainer() (in module paradrop.core.container.dockerapi), 150
- STATE_DISABLED (Chute attribute), 138
- STATE_FROZEN (Chute attribute), 138
- STATE_INVALID (Chute attribute), 138
- STATE_RUNNING (Chute attribute), 138
- STATE_STOPPED (Chute attribute), 138
- StateReport (class in paradrop.core.agent.reporting), 137
- StateReportBuilder (class in paradrop.core.agent.reporting), 137
- STATION
 - pdtools-device-chute-network-station command line option, 74
- status() (AirsharkApi method), 99
- status() (AirsharkManager method), 97
- status() (HttpServer method), 110
- StatusSockJSFactory (class in paradrop.backend.status_sockjs), 114
- StatusSockJSProtocol (class in paradrop.backend.status_sockjs), 114
- statusString() (ConfigManager method), 130
- stealStdio() (Output method), 120
- stimestr() (in module paradrop.base.pdutils), 121
- stockCall() (BaseSession method), 115
- stockPublish() (BaseSession method), 116
- stockRegister() (BaseSession method), 116
- stockSubscribe() (BaseSession method), 116
- stop() (AnalyzerProcessProtocol method), 98
- stop() (Scanner method), 98
- stop_chute() (ChuteApi method), 106
- stopChute() (in module paradrop.core.container.dockerapi), 150
- str2json() (in module paradrop.base.pdutils), 121
- stringify() (in module paradrop.lib.utils.uci), 164
- stringifyOptionValue() (in module paradrop.lib.utils.uci), 164
- SubnetReservationSet (class in paradrop.core.config.reservations), 146
- subscribe() (BaseSession method), 116
- success() (Command method), 125
- in success() (ErrorCommand method), 125
- symlink() (in module paradrop.lib.utils.pdos), 162
- SysReader (class in paradrop.core.config.devices), 140
- SystemStatus (class in paradrop.core.system.system_status), 156
- systemStatus() (in module paradrop.conf.d.client), 125
- T
- tarfile_is_safe() (in module paradrop.backend.chute_api), 107
- TelemetryReportBuilder (class in paradrop.core.agent.reporting), 137
- timedur() (in module paradrop.base.pdutils), 121
- timeflt() (in module paradrop.base.pdutils), 121
- timeint() (in module paradrop.base.pdutils), 121
- Timer (class in paradrop.base.pdutils), 121
- timestr() (in module paradrop.base.pdutils), 122
- toJSON() (StateReport method), 137
- TOKEN
 - pdtools-cloud-claim-node command line option, 67
 - pdtools-routers-claim command line option, 92
- token (PDServerRequest attribute), 136
- trueWrite() (OutputRedirect method), 120
- TwistedException (class in paradrop.base.output), 120
- TwistedOutput (class in paradrop.base.output), 120
- TwistedRequestDriver (class in paradrop.core.agent.http), 136
- type (ConfigOption attribute), 124
- typename (ConfigClass attribute), 131
- typename (ConfigClassgroup attribute), 131
- typename (ConfigClassify attribute), 132
- typename (ConfigDefaults attribute), 127
- typename (ConfigDhcp attribute), 126
- typename (ConfigDnsmasq attribute), 126
- typename (ConfigDomain attribute), 126
- typename (ConfigForwarding attribute), 127
- typename (ConfigInterface attribute), 131, 132
- typename (ConfigObject attribute), 124
- typename (ConfigRedirect attribute), 128
- typename (ConfigRule attribute), 128
- typename (ConfigWifiDevice attribute), 133
- typename (ConfigWifiIface attribute), 133
- typename (ConfigZone attribute), 129
- U
- UCIBuilder (class in paradrop.core.config.devices), 141
- UCIConfig (class in paradrop.lib.utils.uci), 163
- UHTTPConnection (class in paradrop.lib.utils.uhttp), 165
- unlink() (in module paradrop.lib.utils.pdos), 162
- unload() (ConfigManager method), 130
- update() (WampSession method), 137
- update_chute() (ChuteApi method), 106
- update_fetcher (WampSession attribute), 137
- update_hostconfig() (ConfigApi method), 109

updateApply() (ConfigDefaults method), 127
 updateApply() (ConfigInterface method), 131
 updateApply() (ConfigObject method), 124
 updateApply() (ConfigWifiIface method), 133
 UpdateChute (class in paradrop.core.update.update_object), 157
 UpdateEncoder (class in paradrop.backend.chute_api), 107
 UpdateFetcher (class in paradrop.core.update.update_fetcher), 156
 UpdateManager (class in paradrop.core.update.update_manager), 157
 updateModuleList (UpdateChute attribute), 157
 updateModuleList (UpdateObject attribute), 158
 updateModuleList (UpdateRouter attribute), 158
 updateModuleList (UpdateSnap attribute), 158
 UpdateObject (class in paradrop.core.update.update_object), 157
 updatePaths() (in module paradrop.base.settings), 122
 updateRevert() (ConfigDefaults method), 127
 updateRevert() (ConfigInterface method), 131
 updateRevert() (ConfigObject method), 124
 updateRevert() (ConfigWifiIface method), 134
 UpdateRouter (class in paradrop.core.update.update_object), 158
 UpdateSnap (class in paradrop.core.update.update_object), 158
 updateSnap() (in module paradrop.core.config.snap), 147
 updateSnap() (SnapdClient method), 160
 updatesPending() (WampSession method), 137
 updateStatus() (in module paradrop.core.chute.restart), 139
 urlDecodeMe() (in module paradrop.base.pdutils), 122
 urlEncodeMe() (in module paradrop.base.pdutils), 122
 urlEncodeParams() (in module paradrop.core.agent.http), 136
 USAGE (Level attribute), 118
 USB_BUS_ID (SysReader attribute), 140

V

validateInfo() (in module paradrop.base.nexus), 117
 VALUE
 pdtools-chute-set command line option, 66
 pdtools-device-hostconfig-change command line option, 76
 pdtools-node-set-configuration command line option, 89
 VERBOSE (Level attribute), 118
 verify_password() (in module paradrop.backend.auth), 100
 verify_password() (PasswordManager method), 114
 VERSION (NexusBase attribute), 117
 VOLUME

W

waitSystemUp() (ConfigManager method), 130
 waitSystemUp() (in module paradrop.conf.d.client), 125
 WampSession (class in paradrop.core.agent.wamp_session), 137
 WARN (Level attribute), 118
 WebDownloader (class in paradrop.core.container.downloader), 151
 WpaSupplicantConfGenerator (class in paradrop.conf.d.wireless), 134
 write() (in module paradrop.lib.utils.pdos), 162
 write() (OutputRedirect method), 120
 write() (UCIBuilder method), 141
 writeAuthorizedKeys() (in module paradrop.lib.misc.ssh_keys), 160
 writeConfigFile() (in module paradrop.core.config.haproxy), 143
 writeDockerConfig() (in module paradrop.core.container.dockerapi), 150
 writeFile() (Dockerfile method), 151
 writeFile() (in module paradrop.lib.utils.pdos), 162
 writeHeader() (ConfGenerator method), 132
 writeHeader() (HostapdConfGenerator method), 134
 writeHeader() (WpaSupplicantConfGenerator method), 134
 writeOptions() (ConfGenerator method), 132
 writeYaml() (in module paradrop.base.nexus), 118