HASviolet

Installation and User Guide for RPI

Hudson Valley Digital Network

15 August 2020

v2.6

Table of Contents

Introduction

Welcome to HASviolet!

Within Hudson Valley Digital Network (HVDN) we look at HASviolet as a project that is as much (if not more) about the journey as it is about the destination. The project's origins started with a member of HVDN sharing the idea "I've been wanting to do a "communicator" for a while. Something that someone could use over relatively short distances to send text messages back and forth. I have been looking for others to work on a project with too!"

After some investigation LoRa was something to consider since it "should play well with the hacker/maker crowd and possibly the IoT/computer science area." From this point we thought it was a good idea to take forward and started developing a concept around it.

We sifted through a number of existing published LoRa projects using Raspberry Pi and various microcontrollers. We found many github sites with code, libraries, and recommended hardware but most of them were just "proof of concepts", no demonstrations of use cases, or sharing of lessons learned whether as a "lone eagle" or part of a team.

Since there were going to be a number of new and flexing of existing skillsets we were going to develop along the way our differentiator with this project would be sharing the journey through articles, videos, and (hopefully) good documentation.

This guide will help you get started with HASviolet beginning with required RPi hardware and Raspberry Pi OS installation. We will follow this with HASviolet installation, configuration, and the HASviolet application themselves. All references to "RPi" in this guide are to the Raspberry Pi Zero WH hardware.

Design

When we embarked on the HASviolet project, we reviewed all the LoRa projects that were already out there. The vast majority of LoRa projects chose microcontrollers as their platform but we noticed many LoRa projects hit a roadblock after they reached basic objectives such as basic text chat between units. We came to the conclusion that unless a project is use-case driven and highlights the journey to a level of detail for others to build with you, the project just whithers over time especially with microcontroller based projects. We opted to lead the HASviolet project with a RPi solution first since it provides an optimal educational platform opportunity and would ease the "learning curve" as microcontroller options are included.

The Cayman Release (v3) introduces HASviolet Duck, a micrcontroller implementation of HASviolet. HASviolet Duck use cases include:

- a leaf-node in a radio area network (RAN) of "Ducks" managed by HASviolet (RPi)
- standalone applications such RDF beacon

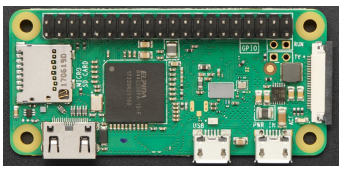
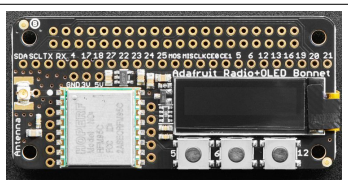
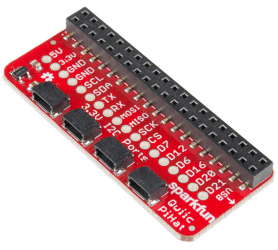
HASviolet and HASviolet Duck complement one another well in building an ecosystem – a hint into what to expect in the Delaware (v4) release.

Hardware

HASviolet consists of the following architecture components:

- Single board computer
- Display (OLED)
- Input (Pushbuttons)
- Network (WiFi)
- RF Module (LoRa)
- RF Antenna
- Power (2.5A)
    - Uses case requiring portable power find 1200maH delivers ~7 hours of RF at low power when sent at 5 second intervals.

Tested hardware inlcudes the following:

Raspberry Pi Zero WH (Wireless with Headers) • You will need to solder the 20-pin header to the board.	
Adafruit LoRa Radio Bonnet with OLED – RFM95W @915Mhz • Requires an antenna connected via a male U.FL connector	
SanDisk Ultra 16GB RAM Class 10 MicroSD • RPi require quality microSD cards at least Class 10	
Power Source 5V @ 2.5A minimum • The hardware can be powered via USB to PC as part of setup but for operation it needs to be on a standalone power source.	
900 MHz Omnidirectional "stub" Antenna • Our antenna U.FL IPEX to SMA Connector • Connect the omnidirectional antenna to the Adafruit LoRa Radio Bonnet	
If you plan on connecting i2c sensors we recommend using the Sparkfun QwiiC Connect System adding the Sparkfun Qwiic HAT for RPI to your hardware stack.	

Software

Operating System

The Raspberry Pi OS (32-bit) Lite image (RPi OS Lite) is chosen because it does not include desktop overhead and the unnecessary services and corresponding “bloat” on CPU and memory that go with it. HASviolet does not use the desktop.

Applications

The HASviolet suite of applications are written in Python 3.X and shell scripts (Bash.) IDE used is a combination of Vi and Visual Studio Code. Shell scripts are used for HASviolet installation and operating system level service (systemd) on boot.

Installation

An overview of in building HASviolet start with:

- microSD card preparation
  - From a computer download and install RPI OS Lite on a microSD card.
  - Mount the microSD card and create/edit files so you can connect to the RPi
    - Connecting directly, WiFi, or USB
- RPi Setup
  - Insert microSD card into RPi and boot
  - Connect to RPi and run the Raspi-config tool to configure the RPi
- HASviolet Install
  - Reboot RPI and download the HASviolet install script
  - Run HASviolet install script then HASviolet config tool

Preparation

Start with downloading the RPI OS Lite on a computer you will be connecting the microSD card to. Whether your computer us Windows, Linux, or OSX, the Raspberry Pi site provides you instructions and tools on installing the image on the microSD card.

Once the image has been created on the microSD card, you will need to mount the card and create/edit files if you plan to remotely connect to the RPi via WiFi or USB. If you plan on using a directly connected monitor and keyboard you will require a mini HDMI adapter for your monitor and a USB OTG cable to attach a keyboard. If this is how you will connect to setup your Pi then you can jump to the RPi Setup section.

Remote Connectivity

Via IP through Wireless Connection

- You can use this method if you wish to access the RPi via an existing WiFi network
- Change to the **boot** directory on the SD card
- Enable **ssh** access by creating a blank file named **ssh**

```
cd /media/sd-card/boot
touch ssh
```

- Within the boot directory, create a file called **wpa_supplicant.conf** and edit

```
vi wpa_supplicant.conf
```

- When you have opened the new file**,** add the configuration at right and save
- Be sure to replace **SSID** with your local wireless network SSID

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="MyWiFiNetwork"
        psk="aVeryStrongPassword"
        key_mgmt=WPA-PSK
}
```

- Change to your home directory, run **sync** as sudo, and unmount the SD card
- Remove the SD card

```
cd ~
sudo sync
umount /media/sd-card/root
umount /media/sd-card/boot
```

Via IP through USB connection

If WiFi is not available, using zero-configuration networking services may be an option if you are running OSX or Windows with Apple BonJour Services installed. Linux users will need to consult their distribution documentation on adding USB as an IP interface and any configuration changes to the installed Avahi software.

- Change to the **boot** directory on the SD card
- Edit the **config.txt** file

```
cd /media/sd-card/boot

vi config.txt

          Append the following line:

          dtoverlay=dwc2

          Then save the file.
```

- While in the boot directory, edit the **cmdline.txt** file, replace a line, then save file.

```
vi cmdline.txt

Replace with the following all as one continuous line:

          dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1
          root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline
          fsck.repair=yes rootwait modules-load=dwc2,g_ether quiet
          init=/usr/lib/raspi-config/init_resize.sh

Than save the file.
```

- Remaining in the boot directory, enable **ssh** access by creating a blank file named **ssh**

```
touch ssh
```

Setup

• Insert SD card • Connect RPI and Power on • Log in with default pi:raspberry	If accessing via IP over USB, connect PC USB port to RPi USB Data port (next to mini-HDMI port) ssh -l pi <yourpi> If connecting via SSH for first time click **yes** to **accept fingerprint**
• Run Configuration tool	`sudo raspi-config`

• Navigate through each menu making selections as noted then exit tool	`1. Change User Password to <something>`
	`2. Network options`
	`N1 Change hostname to your call + number [1-15]`
	`yourcall-5`
	`3. Boot Options`
	`B1 Desktop/CLI`
	`choose B1 Console`
	`4. Localization`
	`I1 Change Local to <your-local>.UTF-8`
	`I2 Change Timezone`
	`5. Interfacing options`
	`P2 Enable SSH`
	`P4 Enable SPI`
	`P5 Enable I2C`
	`7. Advanced Options`
	`A1 Expand filesystem`
	`A3 memory Split`
	`Reduce GPU from 64 to 16`
	`8. Update`

• Run **sync** as sudo, and **reboot**	`sudo sync` `sudo reboot`

Install

• SSH into your Pi
• Run OS update and upgrade
• Run sync and reboot
	`sudo apt-get update`
	`sudo apt-get -y upgrade`
	`sudo sync`
	`sudo reboot`

• Log back in as pi and retrieve the install script from Github	`wget https://raw.githubusercontent.com/hudsonvalleydigitalnetwork/hasviolet/master/HASviolet_install.sh`

• Make the install script executable
• Run the install script Once the script is complete, change directories to **/home/pi/HASviolet**
• Run **./HASviolet-config.py**
	`chmod 755 HASviolet_install.sh`
	`./HASviolet_install.sh`
	`cd ~/HASviolet`

• (1)Change **NOCALL** to your call or handle
• (2) Change **SSID** to a number 50 – 99
• (3) If needed increase **Transmit power** from 5 to as high as 23
• (5) Change **beacon** as you see fit
• (9) to save
• (0) to exit
	`----- HASviolet INI Config Tool (HASviolet-config) -----`
	
	`1 Change Callsign-Handle NOCALL`
	`2 Change SSID 1`
	`3 Change Frequency 911.25`
	`4 Change Transmit Power 5`
	`5 Change Beacon quack quack`
	`6 Change Modem Bw31_25Cr48Sf512`
	`7 Show current HASviolet.ini`
	`- - - - - - - - - - - - - - - - - - - - - - -`
	`8 About HASviolet-config`
	`9 Write changes to HASviolet.ini`
	`0 Exit program`
	
	`Select menu item:`

• Installation is complete. Apps are run from **/home/pi/HASviolet**	

HASviolet is now installed!

Using HASviolet RPi

Overview

HASviolet is a data communications application suite designed to be used on RF networks such as LoRa. The working directory for all applications is installed in ***/home/pi/HASviolet***

The application suite is developed in Python and includes the following core applications;

- **HASviolet-config.py** to configure your station with your call sign/handle
- **HASviolet-beacon.py** sends a repeating broadcast message
- **HASviolet-chat.py** is a half-duplex messaging app
- **HASviolet-tx.py** sends a message to another station
- **HASviolet-rx.py** listens for messages from other stations

Core and other applications have the dependency on the following files also stored in the working directory;

- **HASviolet.ini** is a configuration file generated by **HASviolet-config.py**
- **HASvioletRF.py** is a RF interface library used by all HASviolet applications to support abstraction from the variety of RF libraries/modules expected to be support edover time.
- **HASvioletHID.py** is a HID library (OLED, Buttons, etc) used by all HASviolet applications to support abstraction from the variety of modules/methods to be supported over time.
- **rf95.py** is a HOPE RFM95 library referenced by the **HASvioletRF** library
- **font5x8.bin** is a font file for OLEDs as used by the **HASvioletHID** library

HASviolet was designed with portable use in mind. Upon RPI boot a service is started that makes HASviolet usable via buttons and an OLED. The applications that support this include;

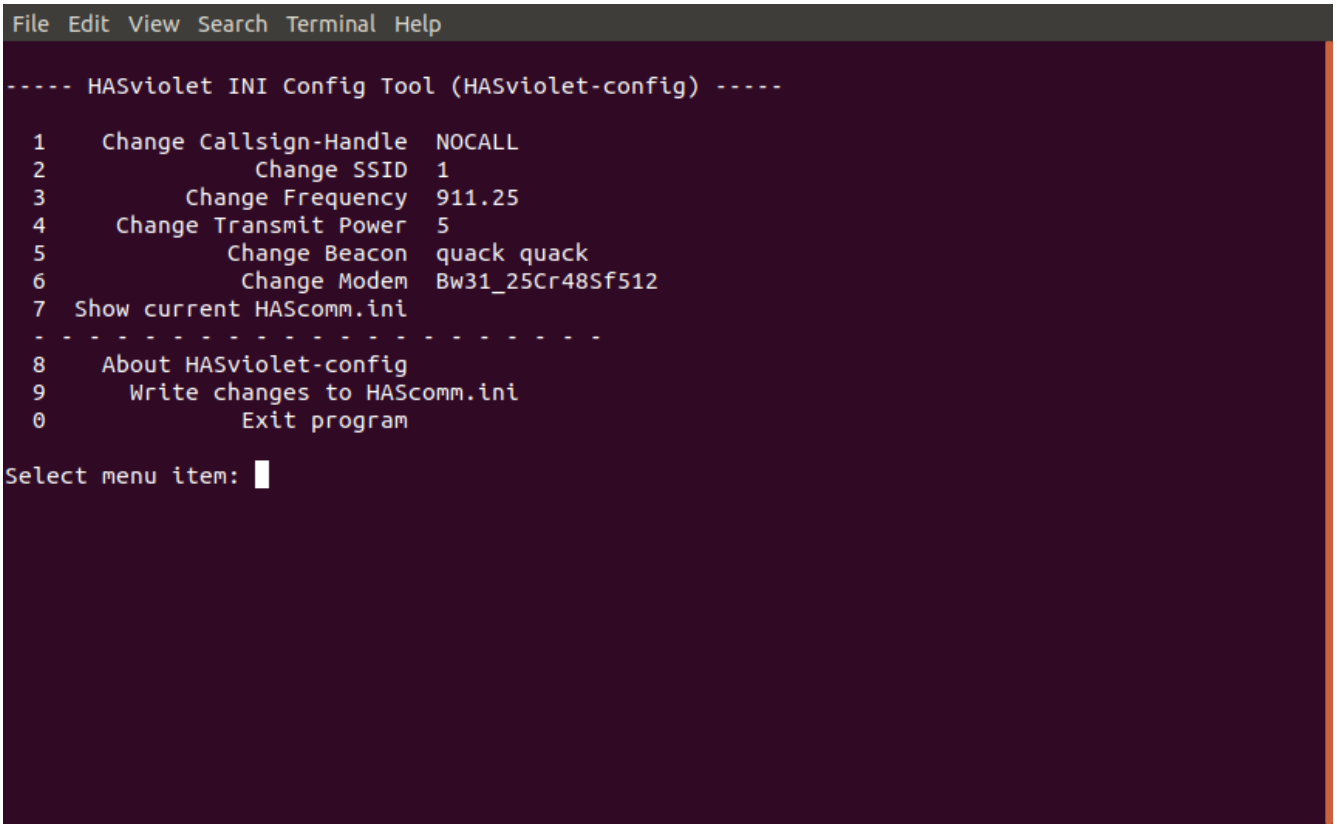
- **HASviolet-handheld.py** is run on RPi bootup providing various RX/TX functions that uses OLED and three buttons as the interface.
    - Among the OLED menu options in HASviolet-handheld.py you will find a game called Duckhunt. More on Duckhun in its own section.
- **HASviolet.service** is a systemd file installed that starts HASviolet-handheld.py
- **HASviolet-service.sh** is a script that can start, stop, and remove the HASviolet service

Core Applications

HASviolet-config.py

Configure your station. Updates HASviolet.ini

```
Usage: HASviolet-config.py
```



HASviolet-beacon.py

Beacon a LoRa message

```
Usage: HASviolet-beacon.py -c COUNT -t DELAY "message"

OPTIONS
  -h, --help            show this help message and exit
  -d DESTINATION, --destination DESTINATION
                        Destination
  -m MESSAGE, --message MESSAGE
                        Message in quotes
```

HASviolet-chat.py

Half-duplex LoRa messaging app

```
Usage: ./HASviolet-chat [-r] [-s]

OPTIONS
    -h, --help      show this help message and exit
    -r, --raw_data  Receive raw data
    -s, --signal    Signal Strength
```

HASviolet-tx.py

Send a LoRa message

```
Usage: HASviolet-tx.py -d DESTINATION "message"

OPTIONS
  -h, --help            show this help message and exit
  -d DESTINATION, --destination DESTINATION
                        Destination
  -m MESSAGE, --message MESSAGE
                        Message in quotes
```

HASviolet-rx.py

Listens for messages from other LoRa stations

```
Usage: ./HASviolet-rx.py -r -s

OPTIONS
    -h, --help      show this help message and exit
    -r, --raw_data  Receive raw data
    -s, --signal    Signal Strength
```

Libraries

For ease of development by end-users, HASviolet core applications in the previous release (Antigua) have been ported into two libraries to be used in your applications. The library names are **HASvioletRF.py** and **HASvioletHID.py**

HASvioletRF.py

All RF functions are performed through this library. It has dependencies on RF specific modules that currently include **rf95.py** for the HOPE RFM95 module currently used on the Adafruit Radio bonnet. Support for the ST126X module is planned for the next release and will include a **ST126X.py** that will be referenced by HASvioletRF.py. Some of the variables available to user applications include the following

Variable	Type	Description
self.freqmhz	float	Frequency within 868 or 900 MHz bands in MHz (ie 911.25)
self.txpwr	integer	Value from 5 to 23
self.mycall	string	Callsign or Handle (ie VIOLET)
self.ssid	integer	Recommended 50-99 (ie 50)
self.station	calculated	Concatenation of HASviolet.mycall and HASviolet.ssid separated by a hyphen (ie VIOLET-50)
self.beacon	string	Default beacon message loaded from HASviolet.ini (ie quack quack)
self.receive	byte array	Payload (in Bytes)
self.receive_rssi	string	RSSI of payload
self.receive_string	string	Payload (in ASCII)

HASvioletHID.py

All HID functions are performed through this library. This includes GPIO addressable buttons and OLED displays. This library has dependencies on the 128x32 SSD1306 display driver as available through the **adafruit_ssd1306** library. Some of the methods available to user applications include the following

Variable	Type	Description
self.OLED	method	Frequency within 868 or 900 MHz bands in MHz (ie 911.25)
self.OLED.fill	method	Fill OLED screen
self.OLED.show	method	Show on OLED screen
self.btnLeft.value	method	Left Button position
self.btnMid.value	method	Middle Button position
self.btnRight.value	method	Right Button position
self.logo	method	Displays HVDN Logo on OLED

Sensors

There are four interfaces that can be used to physically connect sensors and other devices to HASviolet. They are:

- **SPI** which we reserve for the RF module given speed and full-duplex support
- **i2c** which is our preference for use with sensors connected using the Qwiic connect system.
- **microUSB** which we prefer for GPS devices
- **GPIO** as a last resort

The following “proof of concept” sensor applications support named i2c Qwiic connector sensor devices as well as GPS via USB.

- **HASviolet-atmos.py** for use with the Sparkfun BME280 Atmospheric sensor (Qwiic)
- **HASviolet-distance.py** for use with the Sparkfun Distance Sensor (Qwiic)
- **HASviolet-gps.py** for use with USB accessible GPS/GLONASS
- **HASviolet-sensors.py** for use with all three of the aforementioned sensors simultaneously

HASviolet-atmos.py

```
usage: HASviolet-atmos.py [-h] [-d DESTINATION]
HASviolet Atmos Sensor
  -h, --help            show this help message and optional arguments:
  -d DESTINATION, --destination DESTINATION
```

HASviolet-distance.py

```
usage: HASviolet-distance.py [-h] [-d DESTINATION]
HASviolet Distance Sensor
  -h, --help            show this help message and optional arguments:
  -d DESTINATION, --destination DESTINATION
```

HASviolet-gps.py

```
usage: HASviolet-gps.py [-h] [-d DESTINATION]
HASviolet GPS SensorHASviolet-distance.py
  -h, --help            show this help message and optional arguments:
  -d DESTINATION, --destination DESTINATION
```

HASviolet-sensors.py

```
usage: HASviolet-sensors.py [-h] [-d DESTINATION] [-a] [-f] [-g]
HASviolet Sensor TX
  -h, --help            show this help message and optional arguments:
  -d DESTINATION, --destination DESTINATION
  -a, --atmosphere      Atmosphere Sensor
  -f, --distance        Distance Sensor
  -f, --distance        Distance Sensor
```

Appendices

XARPS

XARPS stands for e**X**tensible **A**mateur **R**adio **P**ayload **S**pecification. Referencing the **Open Systems Interconnection model (OSI model)**, XARPS is an application layer protocol with link-level awareness for use by RF systems providing application level connectivity to other networked systems. ASCII (UTF-8) is used throughout the specification

Payload Fields

Source	Destination	Options	Data Type	Data
9 bytes (string)	9 bytes (string)	1 byte (hex)	1 byte (hex)	235 bytes (string)

Source and Destination

Each contain callsign/handle with SSID. Field size supports IARU decision for 7 character callsigns. Example source and destination data include W1FCC50, PURPLE53.

BEACON99 is a reserved word in the destination field for broadcast messages.

Options

Value	Option	Description
0x00 - 0x05	Reserved	
0x06	ACK Response	Response to ACK Request
0x07	ACK Request	Used when positive verification is required that a payload was received

Data Type

Value	Type	Description
0x00	Reserved	Reserved
0x01	Battery	Battery voltage update
0x02	Time	Current time payload
0x03	Position Update	GPS position update
0x04	Weather Update	Weather station data
0x05	Text Message	Text message between stations
0x06	Broadcast	Broadcast Message to all stations
0x07	Last Seen Stations	Digest of recent stations
0x08	Binary Data	Binary Datat Transport
0x08 - 0x0F	Reserved	Reserved
0x46	Gopher	Gopher
0x50	HTTP	HTTP

Value	Type	Description
0x71	Ident	Ident
0x77	NNTP	NNTP
0x7B	NTP	NTP
0xC2	IRC	IRC

0x00 Reserved

Not used

0x01 Battery

This payload type provides battery voltage in ASCII numeric format, which may contain a floating point.

0x02 Time

This payload contains the time in epoch UTC

0x03 Position Update

This message is used to provide position updates, such as GPS locations of any station and is formatted as follows:

[time in UTC],[latitude],[longitude],[altitude],[speed],[direction],[station type]

Station types include:

Value	Type
0	handheld
1	pedestrian
2	car
3	truck
4	van
5	emergency vehicle
6	ambulance
7	fire truck
8	command vehicle
9	officer
10	aircraft
11	boat
12	quadcopter UAV
13	fixed wing UAV
14	balloon
15	float
16	landmark

Value	Type
17	road closed
18	accident
19	hazard
20	Perimeter Marker
21-255	reserved for future use

0x04 Weather Update

Weather updates are still in the definition process, but should utilize METAR format

0x05 Text Message

Free text field for text message between stations.

0x06 Broadcast Message

Broadcast and beacon messages. While BEACON99 is a reserved for any broadcast, it is recommended to BEACON and another SSID if you wish to send specific broadcast types like club bulletins, etc.

0x07 Last Seen Stations

Contains a comma delimited list of recently heard stations.

0x08 Binary Data

Binary Data transport

0x0F Reserved

Not used