Elephant RFID Project 2021

User Manual

1. Contributors

a. Luke Evers - contact: lukeevers@gmail.com
b. Yufei Pan - contact: pan@zopyr.us
c. Shivam Patel - patel.shivam.m@gmail.com
d. Ralph Sanders
e. Yilai Yan - contact: yanyilai201708@gmail.com

2. Quick Start

1.Have a docker-capable machine, run this command:

```
docker run -v feeder_data:/code/db -p 8888:8001 -i -t --restart=always --name
rfid_feeder demonte77/rfid_feeder:latest
```

2.Once the server is started, run the following commands to restart it for good luck :)

```
docker container stop rfid_feeder
```

```
docker container start rfid_feeder
```

3.Then on the same subnet (pi's IP accessible),

Go to https://pi.zopyr.us to download the latest pi image.

Go to https://www.raspberrypi.org/software/ to download the latest Raspberry Pi Imager

4.Click Choose OS, choose the downloaded image.

Plug in a SD card > 8G, choose it as the storage media.

Click "write"

After the process finishes, plug the flashed SD card into the prepared PI.

5.If connected through ethernet, find what the PI's IP is either plug in a monitor into pi and run "ip address" command (and filter the desired ip address)

Or, go to the router for the network and find connected devices. The pi should be called "RASPBERRYPI"

Or, if wanting to connect to wireless network, plug keyboard and monitor to the pi and manually configure the pi and record the pi address (for later use)

6.If you have a monitor connected, skip to the next step,

Alternatively, connecting to the pi VIA RDP is also possible. Write the PI's IP address into the "Remote Desktop Connection" app in windows or "Microsoft remote desktop" app on mac. After connecting, the user name is "pi" and the password is "elephant". (VNC is also available but a monitor need to be plugged for the session to initialize)

7.Now double click the config.ini file on desktop, edit it according to instructions. Delete any site you do not want to be there.

8.Now open the start file on the desktop of pi and click execute in the terminal, then click setup regular. The pi will set everything up accordingly.

9.Now your PI setup is complete!

10.Go to the server's IP:8888 (on a local machine it is 127.0.0.1:8888). Go to admin page,

11.Click "Connect a raspberry pi", fill in the info accordingly, the default port for ftp connection to pi is 21, Name can be anything, and the site code is for the antenna box you are going to connect to the pi. (note, you can have multiple virtual PIs with the same IP and port number but different site code and feeders here. This should be similar to the config you wrote in config.ini on pi.

12.Add Antennas and Feeders accordingly with the PI you just created.

13.Now go to Home or scheduling to create your schedule!

14.Active schedules are in...

15.Active schedules page!

16.And obviously the data log page contains all the previous feeding datas that has been executed by the Pis.

To run server in docker:

```
docker run -v feeder_data:/code/db -p 8888:8001 -i -t --restart=always --name rfid_feeder demonte77/rfid_feeder:latest
```

To stop the docker:

```
docker container stop rfid_feeder
```

To start the docker:

```
docker container start rfid_feeder
```

How to update container (keeping the database):

```
docker container stop rfid_feeder
```

```
docker rm rfid_feeder
```

```
docker run -v feeder_data:/code/db -p 8888:8001 -i -t --restart=always --name
rfid_feeder demonte77/rfid_feeder:latest
```

How to remove the database:

```
docker volume rm feeder_data
```

To create a new custom docker image:

```
docker build --tag rfid_feeder:latest .
```

3. Organization of the Repository

In the root directory there are several files:

- run.sh
    - A script to run the whole server and pi controller
- __init__.py
    - This is used by Django to know where to look for certain files
- Dockerfile
    - The file used to create the Docker image
- .gitignore
    - This is used by github to know which files from local copies of the repository to ignore
    -

Furthermore, there are several folders:

- db
    - This folder contains the sqlite3.db database file
        - Contains minimal data
        - Only has the data that is necessary to run the web application as intended
- pi
    - Contains all the files and controllers that will be using on the Pi. Note: the Pi contains some extra system settings for file sharing and user control. If you want to build your own pi image, the best way to do so is to modify the existing image and then repackage it using the backup to image option in the control panel.
- server_controller
    - Contains all the files for the upstream controller
- Website
    - Contains all the files for the django web application

The web application was created using the python Django framework version 3.1.5. The application sits within the "website" folder in the root directory of the application. The web app is organized into the following. Within the website folder, there are several more folders:

- admin ops
- Datalog
- Elephants
- Templates
- Website

Furthermore, there is a file called "manage.py" and this is the file that is used to run the application. While in the directory that contains "manage.py", a user can type "python manage.py runserver" in order to start a server that serves our web app. Network administrators can serve our site to an ip address of their choice. There is also an __init__.py file that Django uses to determine where to look for files or confirm if it's in the right place.

Admin ops, Datalog, and Elephants are all folders that are modules of the entire web app. Each module is organized into generally the same structure:

views/
- A folder that contains python code to serve files, handle logic, or perform database operations

tests/
- A folder that contains unit tests

static/
- A folder that contains static files such as html, css, or javascript files

templates/
- A folder that contains templates that django can use to render html

migrations/
- A folder that contains database migrations

models.py
- This file contains all the different database models in Django's model notation

admin.py
- A file that tells that django default administration page which database models that are in "models.py" to show

forms.py
- A file that contains forms written in Django's form syntax which can then be used to render html easily

urls.py
- This file routes urls to certain python functions which return an HttpResponse

apps.py
- Used in organizing urls

The website/website folder contains files that the Django server how to route urls and configuration information

4. Web Application

Administration

Getting Started

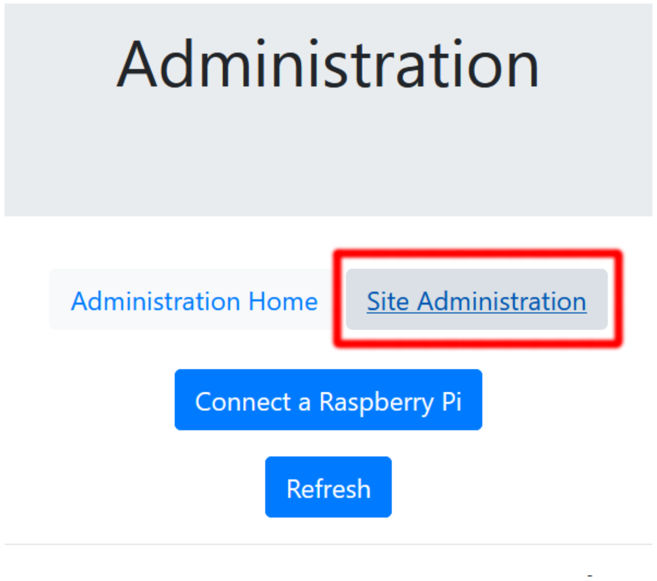
The admin interface is used for configuration of the system and management of data and accounts. It is accessible via clicking on **Admin** in the navigation bar at the top of the site. Administrative functionality requires an admin account.

The application comes with a pre-configured default administrative account:

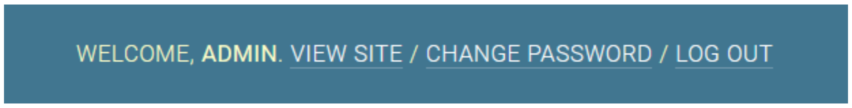
Username: **admin**

Password: **admin**

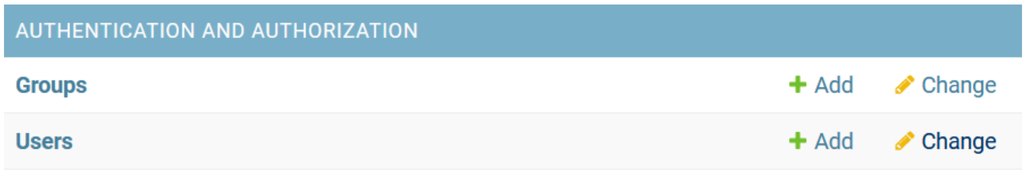
It is recommended that after starting the docker container and thus the web application, the administrator logs in and changes their password. To do this, go to **Site Administration** from the admin page.



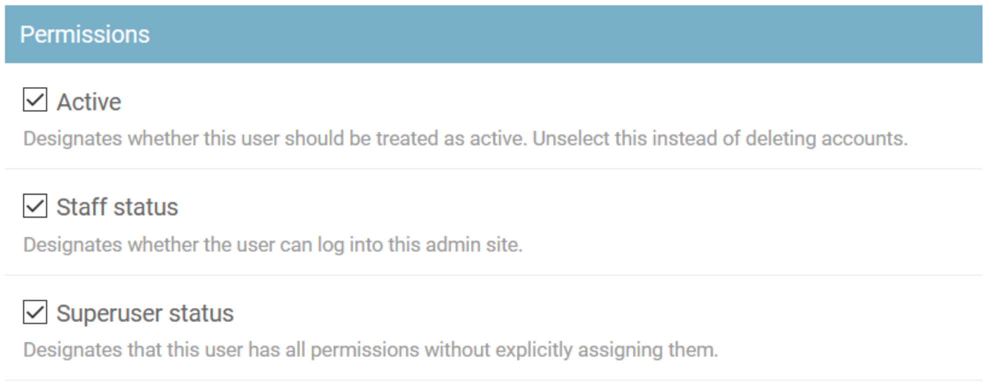
This will open the Django web app management page for the database and user accounts. At the top bar, you can click **View site** to return to the rest of the site, **Change Password** to go to a password change form, and **Log out** to log out.



If you wish to add additional admin accounts or change usernames, you can also do so by going to the Users panel.



To make sure a user has administrative rights, go to that user and ensure that the permissions below are enabled:



Permissions

☑ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☑ Staff status

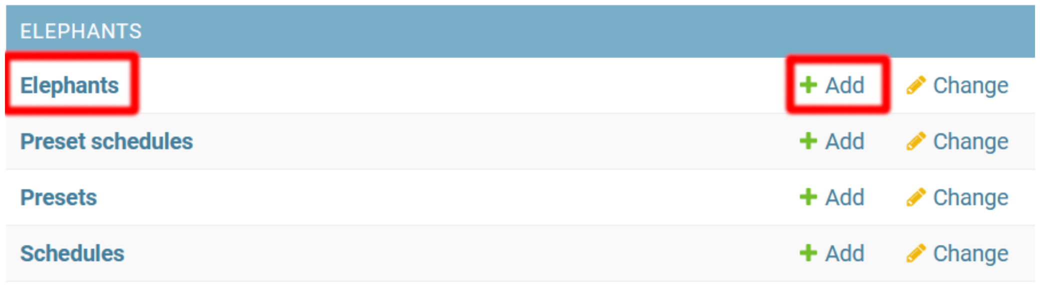
Designates whether the user can log into this admin site.

☑ Superuser status

Designates that this user has all permissions without explicitly assigning them.

Adding Elephants

To add elephants to the system so that they can be referenced in schedules, go to **Elephants** from the **Site Administration** page. You can view elephants with **Elephants** or go straight to adding an elephant with **Add.**



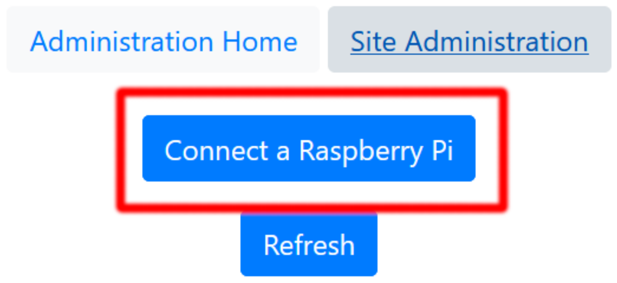
ELEPHANTS		
Elephants	+ Add	Change
Preset schedules	+ Add	Change
Presets	+ Add	Change
Schedules	+ Add	Change

Clicking **Add** will open a form requesting a Name and RFID tag. Enter the name of the elephant (this is for users) and the RFID tag (this is for the system and must be the correct RFID tag).

Repeat this for every elephant you wish to use with the system.

Adding Raspberry Pi Configurations

Next, you need to add Raspberry Pi configuration information so that the server controller can communicate with them and distribute tasks to execute. From the main Admin page, click on **Connect a Raspberry Pi**.

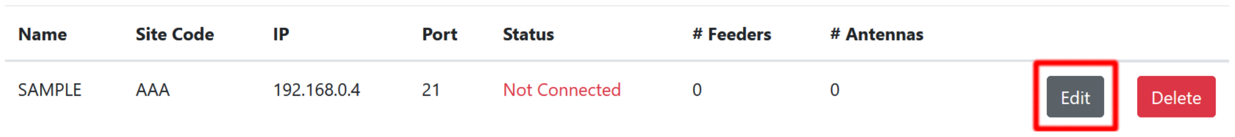


This will open a form for a Raspberry Pi. Enter a nickname for the Pi, its local IP, a port number for FTP communication, and a site code.

The **port number** defaults to 21, and you likely won't want to change this as this is the default port for FTP.

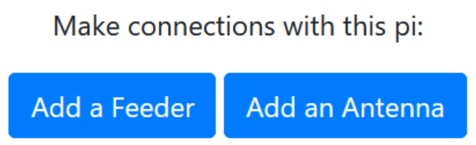
Site code should be the unique site code of the RFID reader box connected to the Pi. It will also be the site code used to refer to the Pi, its feeders, and its reader box as a unit.

After creating the Pi, you'll see it added to the list on the Admin page. Click **Edit** to further configure the Pi.



Name	Site Code	IP	Port	Status	# Feeders	# Antennas		
SAMPLE	AAA	192.168.0.4	21	Not Connected	0	0	Edit	Delete

You should now click **Add Feeder** at the bottom of the page.



This will open a form to add a feeder connected to this Pi. Enter a name for your reference, a tag, and a pin number

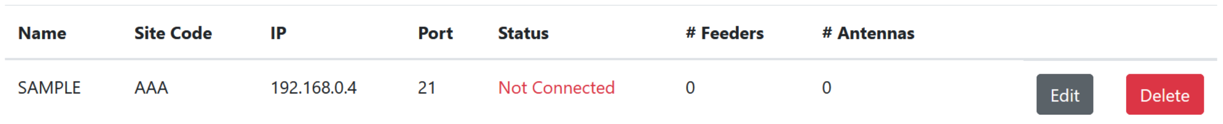
The **tag** should start with an "F". It should be unique to each Feeder across the entire system. This will be used to denote a particular feeder on the back-end, but you may see it in raw data logs.

The **pin number** is the pin-out on the Raspberry Pi used to activate the feeder.

As for antennae, these can be added for your own reference, but there is currently no internal dependency on the antenna configurations.

Verifying Pi Connections

If the server controller is running (as long as the container is running, it should be), the connection status of a Pi will be kept up-to-date in the **Status** column. If you're trying to test or debug a Pi, you can click **Refresh** to force-check if the status has updated.



Name	Site Code	IP	Port	Status	# Feeders	# Antennas		
SAMPLE	AAA	192.168.0.4	21	Not Connected	0	0	Edit	Delete

If the Pi was flashed with the correct image, and all its necessary services are running, it should also be running an FTP server (see sections 5 and 6 for more information) to allow transfer of files containing schedules, configs, and data logs. If the server controller is able to connect to this FTP server, the status will display **Connected**. If it says **Not Connected**, the Pi may still be on or functional, but the controller can't transfer data to or from the Pi. You may need to restart the Pi or check that the FTP server is running via an SSH terminal session.

Datalog

Data Log, Completed Tasks.
Data log displays unix time, site code, antenna tag, and RFID tag number. Completed Tasks displays task uuid, execute after unix time, target site code, target feeder number, amount, if receive from antenna number, if receive from tag number, interval time, expire time, repeat X times, completed time. They retrieve data from the database which is the same with what recorded in raspberry pi.
Click refresh button to retrieve data if the administrator edits the database.
Click download button to save data as a csv file.

Elephants

<u>Scheduling</u>
This option on the navbar allows you to create an independent schedule between an elephant and a feeder with a series of parameters. First, you will select the elephant that you wish this schedule to apply to. The next parameters are the start date, start time, end date, and end time. Here you must select the time and date that you wish the schedule to become active and the time and date that you wish it to become inactive. Using this, you can set your schedule to activate and deactivate at specific times today or at any dates and times in the future. Next, you will enter information about the interval of time between feedings. This time is recorded in minutes. You will then select the maximum number of times that you wish the elephant to be fed during the entire duration of the schedule. If the maximum number of feedings is reached while the schedule is active, then the schedule will cease to dispense food, even if the time interval allows for it. The final schedule parameter that you must select is the feeder. Once you click submit, this schedule between the selected elephant and the selected feeder will automatically activate at whatever time and date that you chose.

<u>Presets</u>
A preset is a group of schedules. You can use a preset essentially as default schedules to run. On the home page there are three preset buttons and when you click one, the schedules that belong to that preset all become active. Since a schedule is for a particular day, when you execute a preset the schedules that belong to it execute on the current day. Thus the start and end times for a schedule remain the same but the date will update to the current day.

Clicking the edit button next to a preset will allow you to add or remove schedules from that preset. When adding a schedule to the preset, the process is similar to the scheduling page above. The only difference is that preset schedules do not allow you to select a date. This is because preset schedules are reusable, and will automatically activate for the current day when the preset is activated.

Presently there are three presets by default, and if you add or remove any the screen looks messy. This was a design flaw because the preset buttons are essentially hard coded in their layout so any number other than three will make things look bad. To add, delete, or change a preset name, go to the admin page to manually make those changes. There is not an option on the front end to add or delete a preset or change the name.

Each preset is tied to an edit button. If you click that edit button it will take you to a screen that shows which schedules belong to that preset and from here you can add or delete new schedules. All additions or deletions will affect only that preset from which you clicked edit.

Main Screen Tiles

On the main screen you will see tiles that are quick snippets of data. There is a tile for each elephant which you can see the name of at the top of the tile. In addition, you'll see information for when the elephant was last fed, the number of times the elephant has been fed on that day, and the time that a schedule ends.

Active Schedules

This option in the navbar allows you to see all the schedules that are currently active. These include both schedules that were activated through a preset, and independent schedules that were created with the navbar scheduling option. The ID column is referring to the unique schedule ID as referenced in the Django database. This page is also useful as a check to use to ensure that your intended active schedules are in fact active.

5. Raspberry Pi (Downstream) Controllers

The controllers on the Raspberry Pi interact with two local folders. One is called shared_data. Which mainly hosts all the datas the server will be interested in. One is called rawdata. This folder will include miscellaneous files that the pi will need to run and some debugging information. All the running scripts will be in the main user's home folder which is /home/pi.

Basic functionality

The Pi will try to understand the tasks provided in shared_data/schedule.todo.templet and combine this information with antenna input and feeder configuration trigger the appropriate pins at a suitable time.

Connect to the Pi

The pi have Samba and ftp open for connection when booted. The passwords is stored in passwords.txt. To connect to the pi via Samba, go to "//[IP of the Pi]/shared_data" and "//[IP of the Pi]/rawdata" using username pi and password elephant_data. To connect to the Pi via ftp, go to "ftp://[IP of the Pi]" with username pi and password elephant. The default ftp directory is /home/pi, which in this directory you can see all the controllers and folder shared_data and rawdata.
To get a CLI interface, connect to the pi via ssh, the pi is default open to connect with user name pi and password elephant.
To get a GUI interface, connect to the Pi via Windows RDP (available on windows, macos, linux) and VNC. CAUTION! To use VNC, you need a monitor already connected to the PI's display output otherwise the pi will not initialize the user session and you will get a black screen. To avoid this, use RDP or buy a fake display dongle and plug it into the pi. (RDP is generally more stable and give a better user experience than VNC)
If connected to the PI via GUI, you can click the start button on the desktop of pi. This will start the ui.py script in the home folder. Start the control panel with a terminal (to allow input to the terminal) by selecting the option. CAUTION: please configure the pi first by editing config.ini (on the desktop which linked to the file in shared_data folder)

Controllers' functionality

UI.py
This is the frontend user interface for iterating with the pi's controllers via system commands. All the commands used in this script can be used individually. It is recommended for advanced users to read though this document and use the command line interface to directly interact with the controllers but this UI will provide basic management functionalities to the PI.
Setup regular
This button runs initializer.py without any extra comments. This means the initializer will setup the pi in default configuration defined by config.ini. It will setup the necessary system services to quickly start listening for serial inputs from the antenna box and listening for schedules send by the upstream server and prepare to interact with them.
Setup no logger
Performs the exact same initial setup without setting up the listener(logger) service for the serial connections from the antenna box. If testing (not physically connected to any antenna boxes) please use this option. The option will prevent the Pi to setup and automatically restart the logger services and endlessly printing out error logs in rawdata/logs folder. In addition, it will also inform the loggers to use the demo file defined in serialLogger.py for testing purposes.
STOP!
Big red button to stop everything from running and emergency halt all running tasks if something terrible had happened. It will also delete all the caches and files and clear any services that will normally try to restart itself. Note any logged data will be preserved but not any running statistics and pin task files etc. Clicking the setup button is needed if want the pi back in working order.
STOP!, but keeping files
Small red button stopping the controllers from running but keep all the running files as they are useful for debugging or something not that terrible had happened and user want to investigate.

Test **
Test the corresponding controllers with needed test options.
Hard reset
It will download the newest files form github, download it and overwrite everything in the local folder CAUTION: this will delete all the data that is not synced yet to the server and will reset all the files to example files. Please only use if everything had broken. If want to update files, please read the update log and just update the corresponding python files.

Backup to Image
This runs the backup script on the pi. Pay attention to the console output. The console will prompt the user to input the external drive's label.
To begin backup, first, click the drive in file explorer, mount the drive and make sure the drive is mounted at /media/pi/ , then input the mounted name (folder name) to the terminal, the pi will begin copying the entire file system to the external drive (make sure it have enough space bigger than the SD card size in pi), then run a script to shrink it. (pishrink.sh, available as a MIT Licenced project on github on Drewsif/PiShrink: Make your pi images smaller! (github.com). The image already have this installed so it will generate a completed image after the task is finished.

initializer.py
This file wil:
1. Set itself as an always running system service, start it, and exit the current session.
2. If it is running as a system service:
    a. Read the config file, if anything is changed:
        i. re-configure feeder status file with appropriate fields
        ii. According to the previously read config file, reconfigure the pin status files in rawdata/tasksrunning folder
        iii. Clear and set services to run the serialLogger for each section (site) present in config.ini
    b. Run controller and activator if not in demo mode
    c. Print out logs and running statistics every few minutes
    d. Repeat step 2 if not demomode.

Use -d append to the end for demomode, -h for help

serialLogger.py
Usage:
serialLogger.py <device path>
Ex.: serialLogger.py /dev/ttyUSB0
serialLogger.py -d
Demomode, use the demo file defined inside this file called demofilename
serialLogger.py -h
Display usage information
This file will:
1. Start serial service using parameters defined in the file, by default, it will use Oregon RFID's serial parameters.

2. Send “DMIF” to the serial console, telling the Oregon RFID control box to send the detection data in real time
3. Loop and read the input data, format it and append it to these three files:
    a. rawdata/serialdata.csv: this file is what is sent to the pi via serial console. This is a direct copy of that data.
    b. rawdata/rfidstatus.txt: only contains the most recent approached data for each antenna. Used by the pi for what elephant/animal is there at the station
    c. shared_data/logdata.csv: this file is what the server will pull as log. Include all approaching data for all connected antennas.

ACTIVATOR.py
The activator will be triggered by initializer.py to execute the job controller has written.
There are no options for this script as it is not intended to be used by the end user.
The ACTIVATOR will read the feeder status from shared_data/feeder.status and pin todo files in rawdata/tasks_running. It will start each thread for each task waiting to be executed. This status is indicated by the shared_data/feeder.status file have a “BUZY” flag for the pin. The process will start a thread for each pin that needs to be triggered. Each thread will go to the file and execute the task by turning on and off the pin X times determined by the task details. The trigger behavior can be changed by modifying the riggerPin method in this file.
The file is designed this way to minimize overlap writing and reading and minimize waiting time for each task. (as each task can take seconds to minutes to complete. This file will only modify the files in rawdata/tasks_running, if a task has completed executing, it will delete all lines in the file for the pin (should only have one line) to indicate to the controller that a task has been done. In addition, it will also cross validate the task detail with feeder.status file to make sure no file has been corrupted before executing.

Controller.py
This is the brain of the pi’s operation.
Controllers will be triggered by initializer.py to modify the files to a ready to be triggered state if needed.
The behavior of the controller can be described below.

This file aims at the purpose of reading from the TODO file, read from the rfid status, read from the feeder status,PIN status, then determine what to do.
It's action will be described as below:
1. Read lines from feeder status file, schedule TODO file, rfid status file, schedule TODO Template File and store them into caches (lists)
2. Go through the feeder status cache. For each line:
    a. If the line indicate the task is COMPLETED, reset the status to WAITING, store the task ID in completed tasks
    b. If the line indicate the task is BUZY, go to the corresponding file for the pin,
        i. If the task is not there: reset the status to WAITING, store the task ID in completed tasks

            ii. If the task is still there: do nothing.
        c. If the line indicates the pin in WAITING, do nothing.
3. Read content of schedule TODO Template File, store all the unique UUIDs and tasks in it.
4. Go through each line of the schedule TODO cache,
    a. If a task was removed from schedule TODO Template File, remove it as well (need to flush pin todo and status file) (record it to completed task for good practice)
    b. If a task is present, remove it from schedule TODO Template cache
    c. If a task have pasts its EXPIRE_TIME, then delete the task from file, record task to completed task
    d. If a task have an ID that was indicated completed, reduce one count from its REPEAT_X_TIMES
    e. If a task have a REPEAT_X_TIMES smaller than 1, then delete the task from file cache, record task to completed task
    f. If a task did not met its activation requirement, skip the task
    g. If a task met its activation requirement, execute the task.
5. If reached the end of file,
    a. Go through the completed task, remove all tasks remaining in the schedule TODO Template cache that is present there.
    b. Go through the remaining lines of schedule TODO Template File, append the tasks to the end of the schedule TODO file
    c. Then, write the modified file cache to disk, terminate the controller and wait for next activation

Activation requirement include:
1. It have passed EXECUTE_AFTER_UNIX_TIME
2. Feeder TARGET_FEEDER_NUMBER at TARGET_SITE_CODE is WAITING
3. IF_RECIEVE_FROM_ANTENNA_NUMBER and IF_RECIEVE_FROM_TAG_NUMBER had been satisfied

Execution procedure:
1. Write into feeder status file cache that the target feeder is BUZY
2. Write task to feeder PIN TODO file (not cached)
3. Add INTERVAL_TIME to the task EXECUTE_AFTER_UNIX_TIME

Files caches will be write in the following order:
1. Feeder status file,updating them to the new status
2. Schedule TODO file updating the status of tasks

This file is designed to be resilient and resistant to file corruption and racing conditions. No two processes from the server and on pi will be writing to the same file at the same time if overwriting not addition. The previous list is not exhaustive and if the user wants to tinker with the functionality of the controller it is recommended to write extra files and extra functions to hook into the existing functionalities. But as the program is written in python and has pretty complete in line comments it is also possible to modify the code directly.

BIG_RED_STOP_BUTTON.py
This file is exactly what it said it is. A Big Red Stop Button.
Usage:
-kf: keep files in place,
-h: display usage information.
This process will first stop, disable, delete the initializer service and script, then it will stop, remove and delete the logger services,
Then it will kill all the controllers that might be running.
Then, if not keeping files, it will delete all the pin todo files and feeder and rfid status file.

createImage.sh
This is a simple shell script to create an image of the current pi system (for distribution) to an external connected storage.
Usage:
First mount the external drive to /media/pi/ this can be done easily by clicking the external drive in the file browser in raspbian, next input the drive's label (the mounted folder name) into the prompting console, the the script will automatically start creating a image with the current time as its name at the location using dd. Then it will start another script by project piShrink to shrink down the size of the image in the external media.

Hard_reset.sh
This is a shell script aim to do last resort recovery or deployment of the whole system. It will download the project from github and place it into /home/pi/rawdata/original_files/. Then it will use the downloaded release branch file to overwrite all files in /home/pi and setup the soft links to the desktop (config.ini and ui.py as start).

demo/ui.py
A alias of ui.py intended to demo the functionality of the python scripts notably it have a setup and cleanup button responsible to setup display the necessary component to show the functionality of the python scripts. If running on non-production pi, it is recommended the user to play with this ui to better understand what all the controllers do on a pi.

<u>Data Files the controllers will be working with</u>

shared_data/schedule.todo.template and shared_data/schedule.todo
Template file stores all the schedules sent to the pi. The pi will not be modifying this template file in any way. It will only make sure it's copy, schedule.todo is up to date with this file. The schedule.todo will be updated constantly by the pi to reflect and store the execution states of the current iteration.
The data fields are as follows:

TASK_UUID,EXECUTE_AFTER_UNIX_TIME,TARGET_SITE_CODE,TARGET_FEEDER_NUMBER,AMOUNT(times of feeder activates in one go), IF_RECIEVE_FROM_ANTENNA_NUMBER,IF_RECIEVE_FROM_TAG_NUMBER,INTERVAL_TIME,EXPIRE_TIME,REPEAT_X_TIMES
In general. the program will activate feeder AMOUNT times if the program loads this list after the EXECUTE_AFTER_UNIX_TIME.
Last five data fields can be skipped and will default to 1,A0,0_0,43200,1; which means to dispense one dose of food when any signal had been received from the antenna, the task can be executed once per hour and will expire in 12 hours (43200 seconds) from the start time.
By default the INTERVAL_TIME is going to be 1 hour (3600 seconds). This means the elephant can only get food one hour after they are first detected.
Note: as the program is stateless, the EXECUTE_AFTER_UNIX_TIME is going to be updated in schedule.todo once an elephant is present. The program is going to write a new time using the current time + INTERVAL_TIME.
The TODO file interpreter (controller) will only execute one TODO task at one time for each feeder connected if conditions met. The lines before other lines will be executed first before other lines (this makes a general case line should be placed at bottom if wanted)
The TODO file interpreter (controller) will subtract count from the list if the task have been completed and all tasks with a count of one (or lower) will be removed
The TODO file interpreter (controller) will read feeder status from files under folder feeder_status.
For example, The following line means for all time after UNIX Epoch +1s, if the station have a code of AAA, when any antenna detected tag 900_226000923030, activate all feeders connected dispensing one dose, the task can be repeated every second for 365k times and the task will expire around year 2077
2393e77f-cb82-4ab1-8e9d-5519ebfba8ee,1,AAA,F0,1,A0,900_226000923030,1,3376684800,365000
Note: F0 and A0 and 0_0 are special tags indicating all feeders, all antennas, and all tags.

If a task first appears in schedule.todo.template, the pi will find that task not in schedule.todo, then proceed to add it to the file and do all the checking and updates if necessary.
If a task is in bothe schedule.todo.template and schedule.todo, the pi will not modify the task in template and do behavior described above and potentially change the task in schedule.todo (the count and the start time). Note, as long as the UUID is the same both in schedule.todo.template and in schedule.todo, the pi will see them as the same task and not verify the content of the task. Thus for the server to modify an already sent task, the only way to do it is delete the current task and send a new one with a new UUID.
If a task is deleted (or the UUID has changed) (This means a task with its UUID is only present in schedule.todo but not in schedule.todo.template) , the controller will execute an emergency halt going to the feeder.status and pin todo files, finding those tasks and delete them if they are currently being executed. Then the controller will move the now deprecated task to completed.todo regardless of status (count/expire time)

After completion, (the task was removed from schedule.todo.template / the task have used up all of its count / the task have passed its' expire time), the task will be added to the end of completed.todo file.

Completed.todo
This file has the same structure as schedule.todo and schedule.todo.template, the only twist is that at the end, the completed.todo also has an extra field COMPLETE_TIME. This indicates when the task has been moved into completed.todo. Generally a task will be moved there after it is completed/ expired, but it can also be the time the pi realized it had been deleted from the schedule.todo.template file thus has been made inactive from the server.
Note the EXECUTE_AFTER_UNIX_TIME and AMOUNT field is likely been altered by the controller thus if the amount is 0 or below, then the EXECUTE_AFTER_UNIX_TIME - INTERVAL_TIME indicate when did the last time before the very last trigger had been activated.
If the EXECUTE_AFTER_UNIX_TIME is higher than EXPIRE_TIME and AMOUNT is positive, this means it is likely the elephant did not always went to the designated station for food and likely skipped some if designed properly. (but it did went some times)
Combined with the original tasks that should still be stored in the servers' database as the original task sent, there can be lots of information pulled from this file.
The data fields are as follows:
TASK_UUID,EXECUTE_AFTER_UNIX_TIME,TARGET_SITE_CODE,TARGET_FEEDER_NUMBER,AMOUNT(times of feeder activates in one go), IF_RECIEVE_FROM_ANTENNA_NUMBER,IF_RECIEVE_FROM_TAG_NUMBER,INTERVAL_TIME,EXPIRE_TIME,REPEAT_X_TIMES,COMPLETE_TIME

shared_data/Config.ini or /home/pi/desktop/config.ini (soft linked)
This file defines all the configs that need to be sent to the pi. This file can be generated by the server if the config is changed there and it can also be manually changed on the pi. Note that the server will overwrite the file if config is changed there thus if wanting to manually modify the file please do not change the config on the server side.
The file is designed to be as below: (<> enclosed content means you want to change it to your preference)

```
[<site name>]
IP = <IP of the station>
SITE_CODE = <Three letter code from the antenna box for reference>
SerialAdapterAddress = <location of the address for the serial device>
FEEDER_NUMBER = <number of feeder you want to be activated>
# Only the first $FEEDER_NUMBER amount of pin will be used. (if want all to be considered, use 0)
# Feeder pin name format: F(any number of characters in ASCII other than 0)_PIN
# The pin number here is the GPIO# in the official diagram
# To verify, use “pinout” in the pi console to see the pinouts.
# F0 means for all pins connected, thus it is an invalid feeder name here
F<custom string>_PIN = <integer number of the GPIO pin number>
……..(repeating F*_PIN configs)
```

```
[<another different site name>]
# below is an example of the config
IP = 192.168.1.254
SITE_CODE = AAA
SerialAdapterAddress = /dev/ttyUSB0
FEEDER_NUMBER = 3
F1_PIN = 15
F2_PIN = 16
Fexample_PIN = 17
Frepeated_PIN = 17
# the following pin is invalid on rasp pi 4 as it only have 27 GPIO pins
Finvalid_PIN = 30
```

Feeder.status
This file stores the status of the available feeders for controllers on the Pi to extract and write information from and to.
Data fields:
TARGET_SITE_CODE,FEEDER_NUMBER,GPIO_PIN_NUMBER,STATUS,ACTIVE_TASK_UUID
status can be: WAITING; BUZY; COMPLETE

1. WAITING status means the feeder is idle and ready for task
2. BUZY means the feeder is actively trying to dispense food
3. COMPLETE means the feeder just completed a task and is waiting the TODO list interpreter to remove the active task

Example:
```
AAA,F1,2,BUZY,1605f11d-83e0-4f33-8c6f-c126e72196b2
AAA,F2,3,WAITING
```

This file is designed in this way so that it may be easier to implement cross-pi interaction (so the TODO task status can be synced across multiple Pis)
It is also possible to have multiple tasks completed but not deleted(subtracted counts) if so wanted later in development
This file only signals one completion of task, aka the tasks with a REPEAT_X_TIMES counter higher than 1 will not be deleted from the TODO list even it is signaled completed here.
The WAITING tag ensures the feeder ACTIVATOR TODO file is empty and can be write into.
The COMPLETE tag ensures the feeder ACTIVATOR TODO file is empty and is waiting to be reset to be waiting. (now deprecated)
The BUZY tag is fail back default indicating the TODO file should not be accessed. The lack of presence of such feeder will be treated as busy.
For example, the task asks for feeder F4 but feeder is not available thus not present in this file thus the task will not be executed unless feeder A4 become available

logdata.csv
This file logs the presence of elephants. This file only records the time, site, Antenna Number and tag number when it is first detected, and will not record any information about such entities leaving the detection range. It will be constantly appended on the loggers.

rawdata/current_config.ini
Stores the current working copy of the config to detect change of the config

rawdata/exampledata.csv
An example of expected data captured from the Oregon RFID antenna box used as input for testing of the loggers

rawdata/rfidstatus.txt
Stores which entity is currently present at the stations. When entities leave (indicated by having a duration time field longer than 0 in the input data fields), the entity is removed from this file. Data fields follows time, site, Antenna Number and tag number.

rawdata/serialdata.csv
Stored the input data from the antenna box complete with the satellite time form the box in a csv format.

rawdata/logs
This folder includes the error output and logs output from serial loggers and initializer.

rawdata/tasks_running
This is the directory stores pin todo files

rawdata/tasks_running/PIN*.todo
Only one line should be in this file, indicating what this feeder needs to output now.
If there are more than one line, only the task listed in the feeder status file will be executed and all other tasks will be cleared.
Data fields follows:

```
TARGET_SITE_CODE,FEEDER_NUMBER,GPIO_PIN_NUMBER,ACTIVE_TASK_UUID,AMOUNT
AAA,F1,3,1605f11d-83e0-4f33-8c6f-c126e72196b2,1
```

6. Server (upstream) Controller

The server-side controller is a program in the “server_controller” folder that manages data logs, schedule distribution, and connecting to the Raspberry Pis. It connects to the same database used by the web server to retrieve schedules and configurations as well as update connection statuses and data logs.

Use

The server controller runs automatically with the web server in the docker container. The docker container automatically runs the command “python3 main.py” to start the server controller. However, the server controller can be run manually or independently on a non-containerized setup or via a terminal connection to the container. To that end, there are several optional command-line parameters when running the server controller, and they are:

- -v : this runs the controller in verbose logging mode, displaying all warnings and info messages
- -s : this suppresses all warnings and info by the controller
- -c : this cleans the data logging state of each Raspberry Pi. The controller will start pulling all data logs at the very beginning, regardless of whether those logs have already been entered into the database. This is useful if Raspberry Pis have been reset. Beware that data duplication may occur.
- -ch : this cleans the data logging state of each Raspberry Pi AND deletes all data logs in the database. This essentially starts the data logging from the beginning as if you had done a clean install. This is useful if Raspberry Pis have been reset. Beware that data loss may occur, so you’ll want to manually download any data logs you want from the database because they will be deleted upon server controller initialization.

The user will likely never need to use these command-line arguments, as most problems can be solved by manually clearing data logs and restarting the docker container.

Configuration

The server controller can be configured via a python file “config.py” found in the server_controller/pi_manager folder. This file has various parameters that can be changed if necessary. These parameters relate to the Pi’s FTP servers:

- USERNAME: this is the username used to connect to each FTP server running on the Raspberry Pis
- PASSWORD: this is the password used to connect to each FTP server running on the Raspberry Pis

These parameters relate to directories and file names the controller expects to find or create on the Pis, and should not be changed except during further development of the application. They are currently set for compatibility with the provided Raspberry Pi image and Pi downstream controller.

- SCHEDULES_TEMPLATE_DIR
- SCHEDULES_TEMPLATE_NAME
- LOGDATA_DIR
- LOGDATA_NAME
- COMPLETED_DIR

- COMPLETED_NAME
- CONFIG_DIR
- CONFIG_NAME

We also have the default serial adapter address for each Raspberry Pi. This is the serial adapter address for connecting RFID reader boxes. As of writing this manual, RFID reader box connections are untested, so this parameter may need to be changed.

- DEFAULT_SERIAL = "/dev/ttyUSB0"

Functionality

The server controller performs two primary tasks: it pulls data logs from the Raspberry Pis, and it sends schedules to the Raspberry Pis if needed. This is a repeated job, done on 10 second intervals (data pulling) and 2 second intervals (schedule distribution).

Pi Connection

Connection to a Raspberry Pi is first verified via logging into its FTP server. If the login is successful, the Pi's connection status will be updated in the database, making the change visible on the website. The controller repeatedly polls for changes to the Raspberry Pi configurations stored in the database and set by the Administrator.

If any FTP operation fails, the Pi will be marked "Not Connected" and the controller will try again until it succeeds or the faulty Pi is removed from the configuration visible in the Admin panel.

Pulling Data Logs

The controller pulls data logs from each Pi periodically, and if any new information has been logged, it saves it to the database. It pulls from "shared_data/logdata.csv" and "shared_data/completed.todo" on the Pi. "logdata.csv" contains the RFID logs as they come from the RFID reader box. "completed.todo" tracks the tasks completed by the Pi; these are the schedules that have been completed and, hopefully, led to Elephants being offered food.

Distributing Schedules

The controller periodically polls the database to check for active schedules. If it finds new or changed active schedules, it sends a "schedules.todo.template" file containing all active schedules to each Pi.