RASv.2 Software Guide

Eficent Business and IT Consulting Services S.L.

This user guide serves as a reference for the initial configuration, functioning and frequently asked questions about the RAS v.2 and its software.

Contents

1 General Overview

The RFID Attendance System v2.0 or RAS v2.0 is a redesign of the RAS concept developed by **Eficent Business and IT Consulting Services S.L.** [2]for the first version, taking only the main idea and a few components from it (concretely the MFRC522 [5] and the use of a buzzer); and upgrading several functional aspects like:

- The device "brain" role is developed by the Raspberry Pi Zero W [6] instead of the NodeMCU. While the second one can only run sequential code with a strictly defined structure (setup and loop), the first one gives total freedom for programming due to its nature as a "little computer".
- The user visual feedback, generated in this case using a SH1106 OLED display [8], while at the RAS v1.0 it was created by means of two color LEDs.
- The parameters configuration is not integrated alongside the WiFi configuration anymore, because they are modified using a configuration portal through a web browser. It also means that they can be changed every time it is necessary.

There are also completely new features:

- A user menu in the own device, which appears when it is turned on, and that offers several possibilities to the user, like the common RFID-Odoo functioning, a RFID reader functionality to be able to know the ID of a card/key, a reset for the WiFi parameters... This menu is navigated by means of two buttones located at the sides of the device: right side for going down in the menu, and left side to select an option (they are capacitive buttons, so they are not visible at first sight).
- The possibility of upgrading the software to newer versions or even change the compatible Odoo version if there is a change in yours company Odoo. It can be done by installing a new image in the RPi, or through *pip*.

For now, the updating through *pip* is not finished, so the update is carried out by pulling the last version of the code from a repository.

2 RPi Zero W initial configuration

The Raspberry Pi Zero W, like the rest of boards of the RPi family, works by means of a MicroSD card in which a suitable OS must be flashed or installed.

Eficent Business and IT Consulting Services S.L. provides deployed images for the RAS v2.0 that when flahsed into the SD, allow a direct use of the device functionalities.

However, for this software guide, the complete process to configure a RPi Zero W from scratch to run the RAS v2.0 functionalities without employing the aforementioned image is explained.

Before starting, acknowledgements must be given to several free-software developers whose code has been used to create some functionalities or to control some devices. They are:

- Jason Burgett → WiFi AP point functionality.
- Louis Thiery → SPI protocol library.
- Mario Gómez → MFRC522 control library.
- Richard Hull → OLED display control library.

The following steps have been used and tested in Ubuntu 16.04. To configure the WiFi, for Mac, it can be interesting to follow Setup Pi Zero W Headless WiFi [7] alongside our guide. For Windows, you can consult How To Setup Raspberry Pi Zero W Headless WiFi [4].

2.1 RPi General Configuration

1. Download the Raspbian Lite image from the Raspberry official site [1] as a ZIP.
2. Download Etcher [3] to flash the SD card.
3. Insert the SD card in the PC.
4. Execute Etcher. Once it is opened, select the Raspbian Lite image, select the SD card as drive and click in "Flash!". It will take several minutes.
5. Once the SD card has been flashed, go to the two generated partitions (boot and rootfs), in terminal or using the filesystem (folders).
6. To enable the **ssh** (we will use it to connect to the RPi), create a blank file called "ssh" (no extension needed) at the boot partition (not the boot folder of the rootfs partition).
7. Go to the rootfs partition and open the file *etc/wpa_supplicant/wpa_supplicant.conf* (beware and be sure you are not at your computer folders, you have to modify the SD ones!).
8. Add the following lines to the end (substituting the contents into the):

```
network={
    ssid="my network name"
    psk="my network password"
}
```

**For other possible network configurations, see Setting up WiFi via the command line.*

9. Add the country code too (to choose the correct frequency bands), as follows (use the ISO ALPHA-2 Code, that is, the one which employs two letters):

```
country=XX
```

10. Go to *etc/network/* and open the *interfaces* file.
11. Find this block in the file (if it is not present, go to the next step):

```
allow-hotplug wlan0
    iface wlan0 inet manual
        wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

12. Substitute the block with (if it is not present, just add the next block to the end of the file):

```
auto wlan0
    allow-hotplug wlan0
    iface wlan0 inet dhcp
        wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

**There is the possibility of fixing the Raspberry Pi Zero W IP address (static) so it is easier to find. See David Maitland Raspberry Pi Zero Headless Setup* for more details.

13. Unmount the partitions and eject the SD card.
14. Introduce it in the RPi Zero, plug the cable to give it power and wait for some time so that the RPi can configure itself.
15. Open a terminal in your computer to connect through SSH to the RPi. There are several ways (from quick (maybe some are more difficult) to slow but easier):
    (a) Try the next command:

    ```
    ssh pi@raspberrypi.local
    ```

    If it worked, it will ask you for a password. Introduce "raspberry" (default password), and you will be able to manage the RPi through terminal by SSH. Go to step 17 now.

    **If you have another RPi connected or you had in the past, it may not work. To clear any previous reference to raspberry.local, use:*

    ```
    ssh-keygen -R raspberrypi.local
    ```

(b) If the previous method didn't worked, we need to find the RPi IP adress. To do that, there are two possibilities:

i. Use *nmap*:

```
sudo apt-get install nmap

nmap -sn 192.168.1.0/24
```

It will show all the connected devices to the WiFi network by their IPs. The problem is that you won't know which one matches your RPi. You will need to try until you find them, or use the command before connecting the RPi, so that the IP that appears when it is connected would be the right one.

ii. Download a smartphone app called Fing. Once it is installed, when you initialize it, it will look for the connected devices to the same network than it, and it will tell you the device description. Look for **Raspberry Pi**.

(c) Once we know the RPi IP adress, use the next command with the proper IP:

```
ssh pi@192.168.1.XX
```

If it worked, it will ask you for a password. Introduce "raspberry" (default password), and you will be able to manage the RPi through terminal by SSH. Go to step 17 now.

16. Now, let's configure the RPi for our use. Introduce the following command to opne the configuration page:

```
sudo raspi-config
```

17. Use the ***Change User Password*** option, and set your own password.
18. Use the ***Advanced Options*** to ***Extend Filesystem***.
19. At ***Interfacing Options***, enable SPI and I2C.
20. Then, go to
textittextbfFinish in the main page and reboot.
21. When the RPi SSH connections comes back to be possible, log in using your new password.

**To save energy and speed up the device (optional settings):*

- Boot up into multi-user mode (disabling the GUI on boot):

```
sudo systemctl set-default multi-user.target
```

- To disable the HDMI, edit */etc/rc.local* and add (above *exit 0*):

```
/usr/bin/tvservice -o
```

22. Update and upgrade:

```
sudo apt-get -y update

sudo apt-get -y upgrade
```

23. Check if the SPI is really enabled using:

```
ls -l /dev/spidev*
```

You must see two devices, one for each SPI bus.

24. Use the following commands to install some additional packages for the I2C:

```
sudo apt-get install -y python-smbus

sudo apt-get install -y i2c-tools
```

25. Check the I2C using:

```
ls -l /dev/i2c*
```

26. Install important programs/packages:

```
sudo apt-get install git

sudo apt-get install python-dev -y
```

2.2 WiFi AP mode configuration

1. Clone the repository and enter the folder:

```
cd

git clone https://github.com/jasbur/RaspiWiFi.git

cd Raspiwifi
```

2. Run the next command (the installation will take several minutes):

```
sudo python3 initial_setup.py
```

   This script will install all necessary prerequisites, copy configuration files, and reboot. When it finishes booting it should present itself in *Configuration Mode* as a WiFi access point with the name *RaspiWiFi Setup XXXX* (the XXXX is a number that depends on the Raspberry, and it is unique for each one of them).

3. To enter again the RPi (these following steps are also useful for the normal use of the WiFi AP mode):
   - Connect to the *RaspiWiFi Setup XXXX* access point using any other WiFi enabled device.
   - Navigate to **http://10.0.0.1:9191** using any web browser on the device you connected with.
   - Select the WiFi connection you'd like your Raspberry Pi to connect to from the drop down list and enter its wireless password on the page provided. If no encryption is enabled, leave the password box blank.
   - Click the *Connect* button.
   - At this point your Raspberry Pi will reboot and connect to the access point specified.

4. Finally, enter the RPi again using SSH as it was explained before.

2.3 SPI and MFRC522 configuration

1. Clone the following repository and enter its folder:

```
cd

git clone https://github.com/lthiery/SPI-Py.git
```

```
cd SPI-py
```

2. Install the SPI module necessary for the MFRC522 library:

```
sudo python setup.py install
```

3. Then, clone the MFRC522 repository and enter its folder:

```
cd

git clone https://github.com/mxgxw/MFRC522-python.git

cd MFRC522-python
```

4. Connect the RFID Reader to the RPi Zero GPIO pins. The template for the explanation is: ***RPI-GPIO-pin:RFIDReader-pin***. Concretely (I will refer here to the physical position of the GPIO pins, it can be found at RPi Zero W Pinout).

GPIO (BOARD)	**RFID pin**
*GPIO*17	3.3V
*GPIO*19	MOSI
*GPIO*20	GND
*GPIO*21	MISO
*GPIO*22	RST
*GPIO*23	SCK
*GPIO*24	SDA

Table 1: GPIO (RPi) - RFID pins convsersion table

5. Make a test to check that the installation has been succesful:

```
python Read.py
```

Pass a card next to the RFID card, and it should be written at the terminal.

2.4 SH1106 OLED screen configuration

1. Install some necessary packages:

```
 cd

 sudo apt-get install python-dev python-pip libfreetype6-
dev libjpeg-dev build-essential
```

2. Install the luma.oled package (it will take several minutes):

```
 sudo -H pip install --upgrade luma.oled
```

3. Wire up the display to the RPi (same explanation template than for the MFRC522):

GPIO (BOARD)	OLED pin
*GPIO*1	VDD
*GPIO*3	SDA
*GPIO*5	SCK
*GPIO*6	GND

Table 2: GPIO (RPi) - SH1106 OLED display pins convsersion table

4. Execute the following commands to confugire the user rights and install some necessary packages:

```
 sudo usermod -a -G i2c,spi,gpio pi

 sudo apt install python-dev python-pip libfreetype6-dev
libjpeg-dev build-essential

 sudo apt install libsdl-dev libportmidi-dev libsdl-ttf2.0-
dev libsdl-mixer1.2-dev libsdl-image1.2-dev
```

5. Reboot (*sudo reboot*) and enter again the RPi through SSH.

6. Clone the repository and enter its folder:

```
 cd

 git clone https://github.com/rm-hull/luma.examples.git
```

7. Install the luma libraries using (it will take several minutes):

```
sudo -H pip install -e .
```

8. To test that everything is properly installed, run one of the examples:

```
cd examples

python pi_logo.py -d sh1106
```

If the Raspberry Pi logo is displayed at the screen properly, the installation was succesful.

9. To make the SH1106 the default display (now, the default one is the SSD1306), let's edit one of the luma.core files:

```
sudo nano /usr/local/lib/python2.7/dist-packages/luma/core
/cmdline.py
```

Go to the *create_parser*(*description*) method, and change the following line:

```
general_group.add_argument('--display', '-d', type=str, default=
display_choices[0], help='Display type, supports real devices or
emulators. Allowed values are: {0}'.format(', '.join(
display_choices)), choices=display_choices, metavar='')
```

By:

```
general_group.add_argument('--display', '-d', type=str, default=
display_choices[6], help='Display type, supports real devices or
emulators. Allowed values are: {0}'.format(', '.join(
display_choices)), choices=display_choices, metavar='')
```

So that the default field is loaded with the 6th element of the *display_choices* list (SH1106) instead of the 0th (SSD1306).

10. Finally, test again the screen functioning but using:

```
python pi_logo.py
```

If the Raspberry Pi logo is displayed at the screen properly, this final configuration was succesful.

2.5 RFID - OLED - ODOO and Configuration Server program configuration

1. Clone the repository:

   ```
   cd

   git clone https://github.com/lurobe94/Raspberry_Code.git
   ```

2. Install Flask and Flask-Babel for the server:

   ```
   pip install Flask

   easy_install Flask-Babel
   ```

3. Test the server to create the ***data.json*** file, which the main program will read to get the Odoo parameters:

   ```
   cd Raspberry_Code/Config-Portal/

   sudo python config-server.py
   ```

   Enter using any web browser to the Raspberry IP address (192.168.1.XX) (it may change depending on the country), and the login portal will appear. Enter the credentials (they can be changed in hello-template.py) and the configuration portal will appear. Enter then the Odoo parameters, a RFID card ID for the administrator, and do not select the update option, as you just clone it.

   The data.json file must have been generated in the repository. Check the parameters are right.

4. Test the main program by:

   ```
   python launcher.py
   ```

   The program should work as expected. You can use the administrator RFID card to end the program.

5. To configure the RPi to run the program at boot, edit the *rc.local* file:

   ```
   sudo nano /etc/rc.local
   ```

First, add these lines below the line "By default this script does nothing", to generate a log file for debugging errors.

```
exec 2> /tmp/rc.local.log
exec 1>&2
set −x
```

Add at the end, but before the ***exit 0*** line, the following:

```
python /home/pi/Raspberry_Code/launcher.py &
pyhton /home/pi/Raspberry_Code/Config−Portal/config−server.py &
```

Check the path, it can be different if you structured the folders in another way. Don't forget the &, so that the programs runs in separated processes and the RPi can initialize itself without problems.

2.6 Final test

1. To test that everything is properly configured, use the following commands:

```
rm -f data.json

cd

python Raspberry_Code/manual_reset.py
```

So that the data.json is deleted (and you can test the whole functioning of the device) and the Raspberry is rebooted, as well as the WiFi parameters are erased. In this way, you must see first the screen that tells you to enter to the WiFi network, and that once you configure the WiFi, the program asks you to go to the configuration portal, and once you enter the parameters, the main program is executed properly.

3 Software operation processes

In this section, the different functionalities of the main program, with all its subcomponents, and the configuration server are explained, as well as the possible feedback the user can receive, his/her interaction with the device... A diagram showing these functionalities and relations is presented in Figure 1.

The Raspberry Pi Zero W file **rc.local** allows to run all the commands written on it at boot, so two programs are called from there: the launcher of the main program (or RFID-OLED-Odoo program) and the configuration portal.

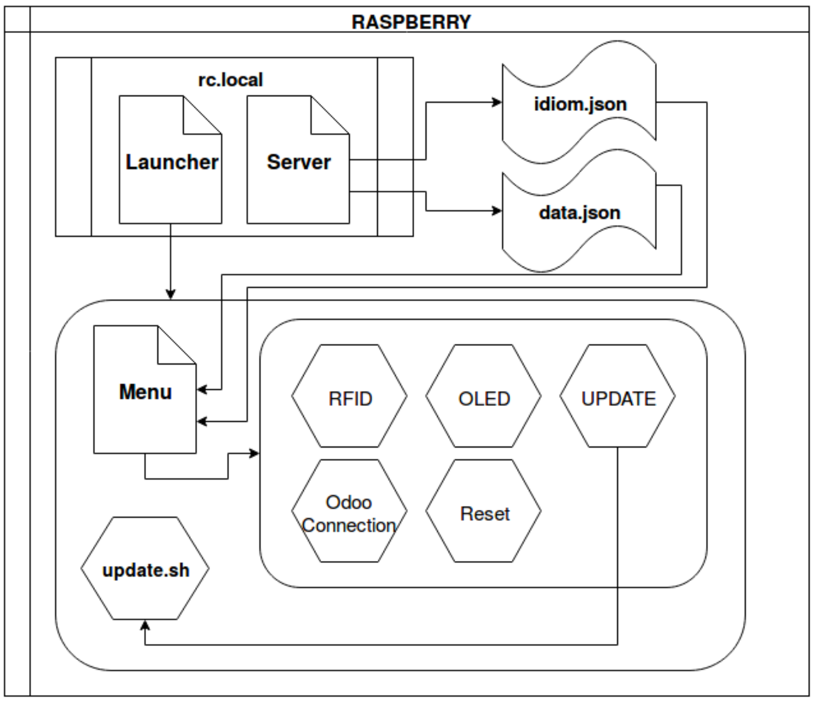

Figure 1: General functioning scheme

Both programs communicate using a text file known as **data.json**. It stores the parameters introduced using the configuration site, and they are read from this file by the main program. There is other shared file, **idiom.json**, which stores the language defined by the user from the displayed options.

Regarding the launcher, it first runs the program which executes the main functionalities, which are the RFID reading, the OLED siplay control, the connection with Odoo, the RAS v2.0 menu, the reset of the parameters, the update of the software...

If the update option is used at this main program, before exiting, the launcher will update the local repository with the last version of the code, and reboot the RPi.

The configuration portal uses another file called **credentials.json**, where the user and password of the configuration portal are stored.

3.1 RFID-OLED-Odoo program

This piece of software acts as the main program, running all the processes except the configuration server. It controls the OLED display, the MFRC522, the communication with Odoo..., as well as it allows user interface interaction due to its menu.

To understand it, let's review the different functions that conform it:

Libraries, global variables initialization and hardware configuration

The first two included libraries are necessary to be capable of using the RPi GPIOs and the MFRC522 RFID reader. Then, the next three are employed for the connection to Odoo, as well as the following one is utilized for the WiFi checkings. Finally, the last libraries have different uses through the program, so their specific usage will be explained later.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import RPi.GPIO as GPIO
import MFRC522

import xmlrpclib
import socket
import urlparse

try:
    import httplib
except:
    import http.client as httplib

import binascii
import random
import os, sys, time
```

Then, the OLED display management libraries are included, as well as another one for being able to read and write *.json* files.

```
from demo_opts import get_device
from luma.core.render import canvas
from PIL import ImageFont
from PIL import Image
from datetime import datetime

import json
```

Some of the global variables are initialized here. Their meaninng will be clarified later, while explaining their role at the different functions.

```
error = False
card_found = False

cnt_found = 0

turn_off = False
adm = True
elapsed_time=0.0
pos = 0
enter = False
```

```
 reset = False
on_Down = False
on_OK = False
```

The GPIOs to manage the buttons are defined and configured in BOARD mode, so that the number of the pin is the physical position it has at the pinout.

```
GPIO.setmode(GPIO.BOARD)  # Set's GPIO pins to BOARD GPIO numbering

INPUT_PIN_DOWN = 31             # Pin for the DOWN button
GPIO.setup(INPUT_PIN_DOWN, GPIO.IN)  # Set our input pin to be an input

INPUT_PIN_OK = 29               # Pin for the OK button
GPIO.setup(INPUT_PIN_OK, GPIO.IN)  # Set our input pin to be an input
```

Several dictionaries are defined, mostly to have all the displayed messages in the same format:

- The first two dictionaries are the ones storing the messages for the normal functioning of the RFID - Odoo functionality, one for Spanish and other for English. The format of the stored data is: ***key: [message, x position of the first line, number of lines, x position for the second line, x position of the third line, letter size]***.

  The x position of the line is referred to the location of the sentence at the OLED display, regarding the x coordinate.

- The next two dictionaries store the messages for the error situations, one for Spanish and other for English. The template is: ***key: [number of messages, first message, number of lines (1),x position for the first line (1), x position for the second line (1), x position of the third line (1), second message, number of lines (2),x position for the first line (2), x position for the second line (2), x position of the third line (2), ..., letter size]***.

  The dotted line implies that more than two messages are possible.

- The following two dictionaries generate a bigger dictionary for each one of the two types of previous dictionaries (normal functioning or error). Like this, the language can be accessed using a variable.

- The last dictionary is employed to set the timezone for the time showed at the display.

```
dic_en={' ': [" ",0,1,0,0,24], 'check_in': ['CHECKED IN',6,1,0,0,22],...}
dic_es={' ': [" ",0,1,0,0,24], 'check_in': ['ENTRADA REGISTRADA'
  ,20,2,3,0,22],...}

dicerror_en = {' ': [1," ",1,0,0,0,24], 'error1': [2,'Odoo communication
 failed',3,41,5,40,'Check the parameters',3,41,53,20,19],...}
dicerror_es = {' ': [1," ",1,0,0,0,24], 'error1': [2,'Error de
 comunicacion',3,47,54,15,'Chequea los parametros',3,28,50,20,19],...}
```

```
dic = {'es': dic_es, 'en': dic_en}
dicerror = {'es': dicerror_es, 'en': dicerror_en}

tz_dic = {'-12:00': "Pacific/Kwajalein", '-11:00': "Pacific/Samoa",...}
```

The MFRC522 class is initialized, creating an instance to work with the device. Moreover, another dictionary is created, but this one is employed to select the functionalities from the menu. In this case, the parameter for each key is the name of a function.

```
MIFAREReader = MFRC522.MFRC522()

ops = {'0': rfid_hr_attendance, '1': rfid_reader, '2': settings, '3':
  back, '4': reset_settings, '5': change_language}
```

Starting function

First, let's show the complete function:

```
def m_functionality():
    global device, lang
    global update
    global reset
    global host, port, user_name, user_password, dbname
    global admin_id, https_on
    try:
        device = get_device()
        img_path =    os.path.abspath(os.path.join(os.path.dirname(
        __file__),'images', 'ef5.png'))
        logo = Image.open(img_path).convert("RGBA")
        fff = Image.new(logo.mode, logo.size, (0,) * 4)

        background = Image.new("RGBA", device.size, "black")
        posn = ((device.width - logo.width) // 2, 0)

        img = Image.composite(logo, fff, logo)
        background.paste(img, posn)
        device.display(background.convert(device.mode))

        json_file = open('/home/pi/Raspberry_Code/idiom.json')
        json_data = json.load(json_file)
        json_file.close()
        lang = json_data["language"][0]
        print lang
        print json_data["language"]
        lang2 = json_data["language"]
        if lang2 == "es":
            lang = "es"
        else:
            if lang2 == "en":
                lang = "en"
        time.sleep(4)
        welcome_msg(device,17)
        time.sleep(4)
        if have_internet():
            while not os.path.isfile("/home/pi/Raspberry_Code/data.json")
:
                screen_drawing(device,"config1")
                time.sleep(2)
                screen_drawing(device,"config2")
                time.sleep(2)
            if os.path.isfile("/home/pi/Raspberry_Code/data.json"):
                json_file = open('/home/pi/Raspberry_Code/data.json')
                json_data = json.load(json_file)
                json_file.close()
                host = json_data["odoo_host"][0]
                port = json_data["odoo_port"][0]
                user_name = json_data["user_name"][0]
                user_password = json_data["user_password"][0]
                dbname = json_data["db"][0]
                admin_id = json_data["admin_id"][0]
                timezone = json_data["timezone"][0]
                os.environ['TZ'] = tz_dic[timezone]
                time.tzset()
                print time.strftime('%X %x %Z')
                if "https" not in json_data:
                    https_on = False
                else:
                    https_on = True

                if "update" not in json_data:

                    update = False
                else:
                    update = True
                print "THIS IS UPDATE: " + str(update)
            else:
                raise ValueError("It is not a file!")
        main()
        if update == True:
            screen_drawing(device,"update")
            time.sleep(2)
            screen_drawing(device," ")
            GPIO.cleanup()
        screen_drawing(device,"Bye!")
        time.sleep(3)
        screen_drawing(device," ")
        GPIO.cleanup()
        if reset == True:
            screen_drawing(device," ")
            reset_lib.reset_to_host_mode()
        return update

    except KeyboardInterrupt:
        GPIO.cleanup()
```

```
            pass
```

First, some global variables are included so that they can be used inside the function.

To briefly describe them:

- *device*: it is necessary to create an instance of the display in the luma library.
- *lang*: used to define the actual language.
- *update*: it shows if the update option has been selected or not.
- *reset*: idem, but for resetting the WiFi parameters.
- *admin_id*: idem, but for the detection of the administrator card.
- *https_on*: idem, but for the usage of HTTPS or HTTP when communicating with Odoo.
- *host, port, user_name, user_password, dbname*: they are the Odoo parameters taken from the *data.json* file.

```
    global device, lang
    global update
    global reset
    global host, port, user_name, user_password, dbname
    global admin_id, https_on
```

All the next parts of the code are inside the try method, and as we will see later, the exception is the keyboard interruption, allowing the programmer to stop the program with ease.

The first task we take on consists of displaying the Eficent logo on the OLED. We will not enter in the functioning of the *luma* library methods. They can be found in the aforementioned documentation.

A picture about the result can be found at Figure 2.

```
  try:
        device = get_device()
        img_path =     os.path.abspath(os.path.join(os.path.dirname(
        __file__),'images', 'ef5.png'))
        logo = Image.open(img_path).convert("RGBA")
        fff = Image.new(logo.mode, logo.size, (0,) * 4)

        background = Image.new("RGBA", device.size, "black")
        posn = ((device.width - logo.width) // 2, 0)

        img = Image.composite(logo, fff, logo)
        background.paste(img, posn)
        device.display(background.convert(device.mode))
```

Then, it is time to configure the language. Hence, the *idiom.json* file is read, extracting the correct idiom. A conversion id made due to some difficulties in the extraction of the

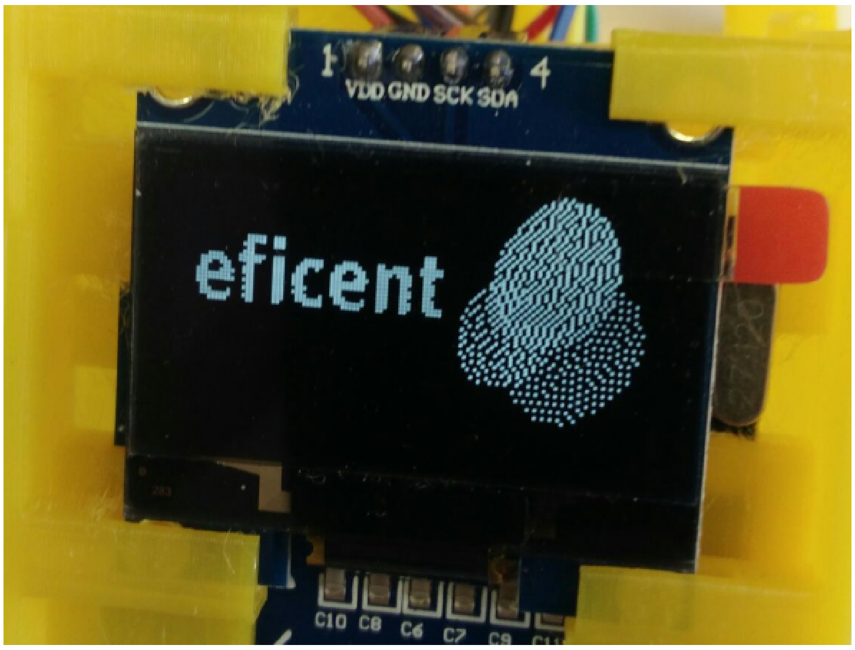

Figure 2: Eficent logo

language depending if it was changed the last time from the menu or from the configuration portal.

Then, after waiting for four seconds in which the Eficent logo is displayed, a welcome message is shown during other four seconds.

The result can be seen at Figure 3

```
json_file = open('/home/pi/Raspberry_Code/idiom.json')
    json_data = json.load(json_file)
    json_file.close()
    lang = json_data["language"][0]
    print lang
    print json_data["language"]
    lang2 = json_data["language"]
    if lang2 == "es":
        lang = "es"
    else:
        if lang2 == "en":
            lang = "en"

    time.sleep(4)
    welcome_msg(device,17)
    time.sleep(4)
```

First, it is checked if there is WiFi connection (the function is explained later). Regarding there is connection, it is checked if the *data.json* file with the different parameters does exist, because if not, it is printed a message on the screen inquiring us to go to the configuration portal (see Figure ??). When the configuration file does exist, the different

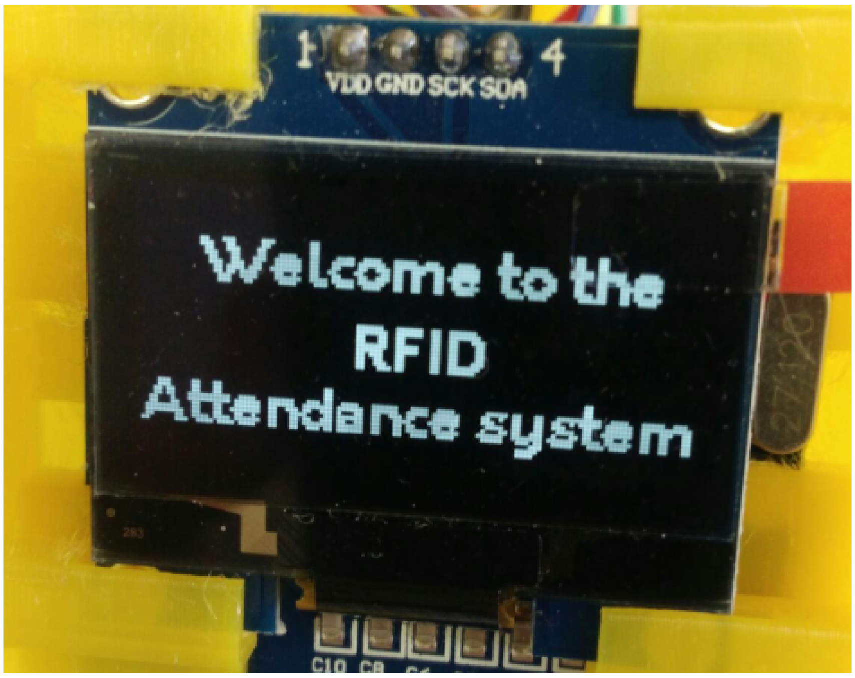

Figure 3: Welcome message

parameters explained before are read from it, as well as different flags values are updated (*https_on, update*). Moreover, the timezone is set so that the device clock is correctly shown.

```
    if have_internet():
            while not os.path.isfile("/home/pi/Raspberry_Code/data.json")
    :
                screen_drawing(device,"config1")
                time.sleep(2)
                screen_drawing(device,"config2")
                time.sleep(2)
            if os.path.isfile("/home/pi/Raspberry_Code/data.json"):
                json_file = open('/home/pi/Raspberry_Code/data.json')
                json_data = json.load(json_file)
                json_file.close()
                host = json_data["odoo_host"][0]
                port = json_data["odoo_port"][0]
                user_name = json_data["user_name"][0]
                user_password = json_data["user_password"][0]
                dbname = json_data["db"][0]
                admin_id = json_data["admin_id"][0]
                timezone = json_data["timezone"][0]
                os.environ['TZ'] = tz_dic[timezone]
                time.tzset()
                print time.strftime('%X %x %Z')
                if "https" not in json_data:
                    https_on = False
                else:
                    https_on = True
                if "update" not in json_data:

                    update = False
                else:
                    update = True
                print "THIS IS UPDATE: " + str(update)
            else:
                raise ValueError("It is not a file!")
```

References

[1] Downloads - Raspberry Pi. `https://www.raspberrypi.org/downloads/raspbian/`.

[2] Eficent - About us. `http://www.eficent.com/about-us/`.

[3] Etcher - Official Site. `https://etcher.io/`.

[4] How To Setup Raspberry Pi Zero W Headless WiFi. `https://core-electronics.com.au/tutorials/raspberry-pi-zerow-headless-wifi-setup.html`.

[5] MFRC522 Datasheet. `https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf`.

[6] Raspberry Pi Zero W Official Site. `https://www.raspberrypi.org/products/raspberry-pi-zero-w/`.

[7] Setup Pi Zero W Headless WiFi. `https://desertbot.io/blog/setup-pi-zero-w-headless-wifi`.

[8] SH1106 132x64 Dot Matrix OLED/PLED Datasheet. `https://www.elecrow.com/download/SH1106%20datasheet.pdf`.