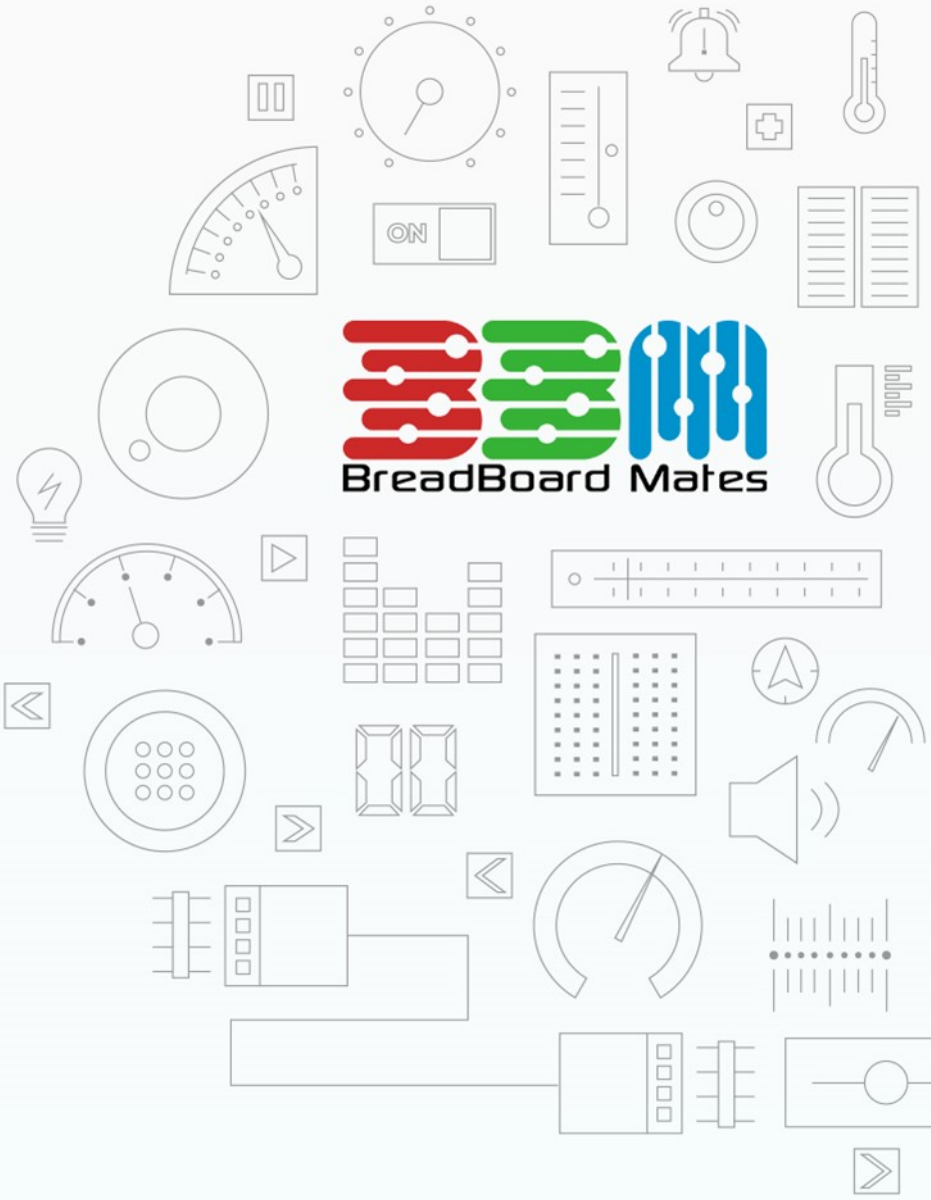


PROJECT

Raspberry Pi REPTOR-250 Status Monitor

Content may change at any time. Please refer to the resource centre for latest documentation.

Contents

Introduction

Adding a status monitor to any Raspberry Pi project can be very useful to give at a glance real time status of how the CPU is performing in terms of usage, temperature and RAM use as well as vital connectivity information. If over-clocking is your thing then this can prove extremely valuable to see what impact your settings have on the Pi and make adjustments accordingly.

The **BBM REPTOR-250** display together with the **BBM Pi adapter** are a perfect choice for this project due to it's ease of use and simplicity in code needed to get this up and running.

Additionally, touch allows the user to select a status page style by swiping through all options. BBM provide a Python library which makes the whole Python coding experience effortless.

Requirements

To proceed with the project, the following are required.

Hardware

- Raspberry Pi Model 3B or Raspberry Pi Model 4
- BBM REPTOR-250
- BBM Pi Adapter
- HDMI Monitor or TV
- Keyboard & Mouse
- uSD Card 4GB or higher
- Internet Connection

Software

- Raspberry Pi OS
  - Python3 (built into Raspberry Pi OS)
- MobaXTerm or similar
- Mates Studio

REPTOR-250 Setup

The BBM Pi Adapter will need to be attached to the Pi GPIO Header and the REPTOR-250 attached to the adapter as shown below.

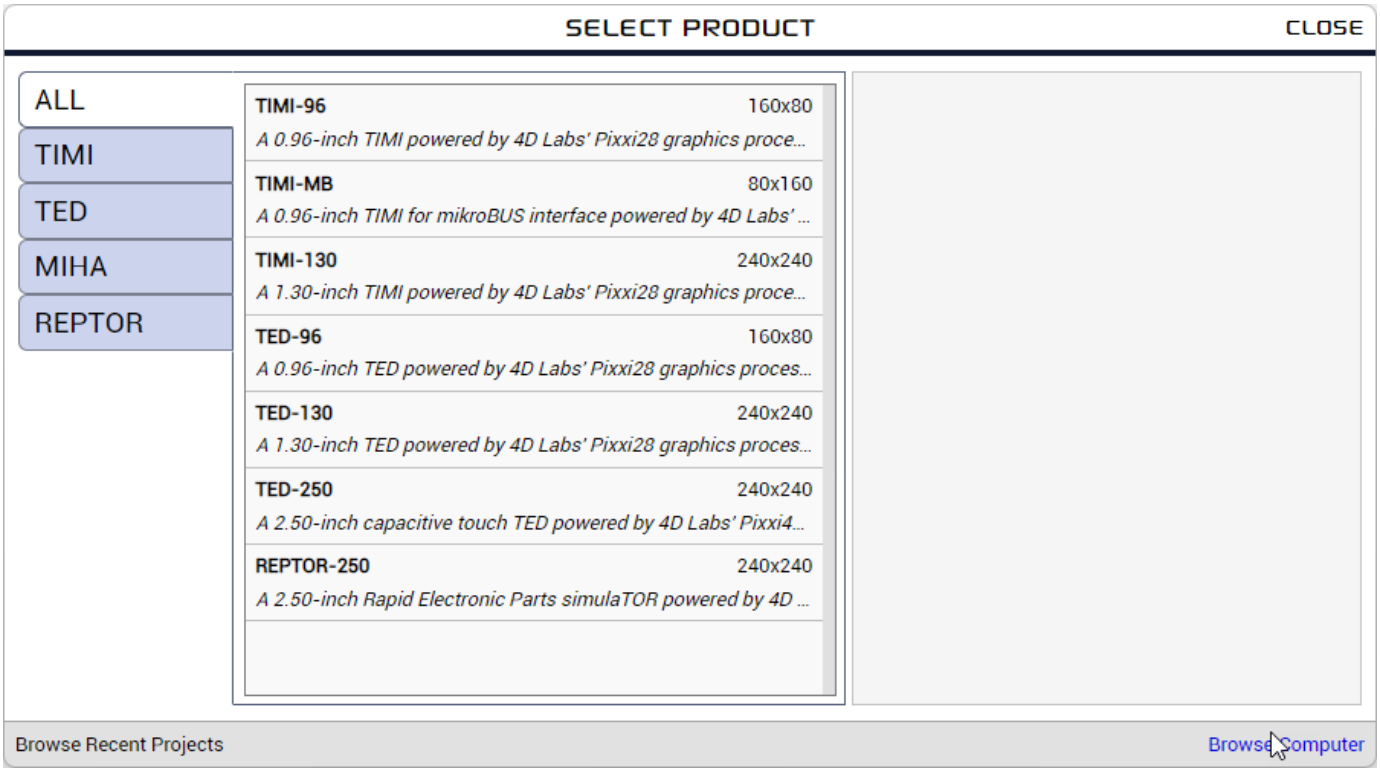
As the display needs to be configured for the Status Monitor, the switch on the Pi adapter needs to be set to PROG. Next attach the BBM Programmer to the Pi Adapter.

Connect a USB cable to the BBM Programmer and to a PC USB port. The REPTOR-250 is now ready for the Status Monitor project to be installed.

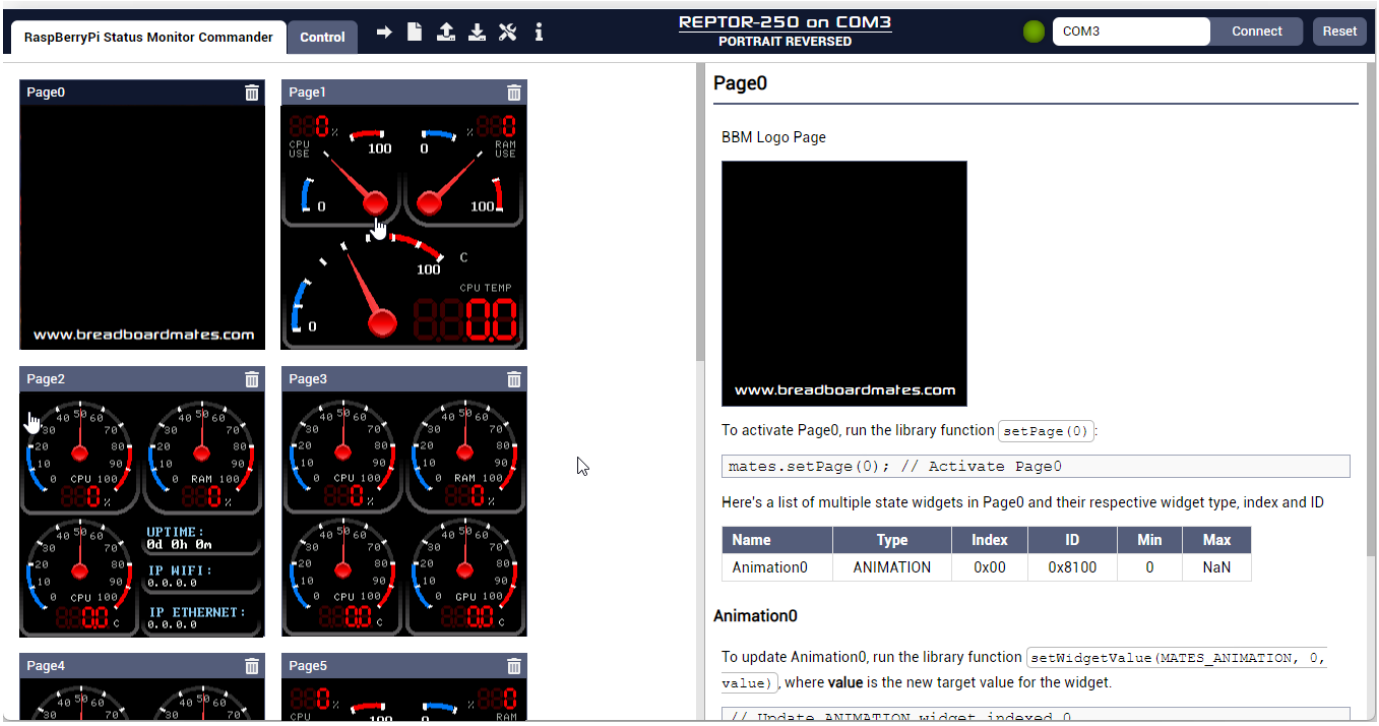
Mates Studio will be required to configure the REPTOR-250.

Graphics Design

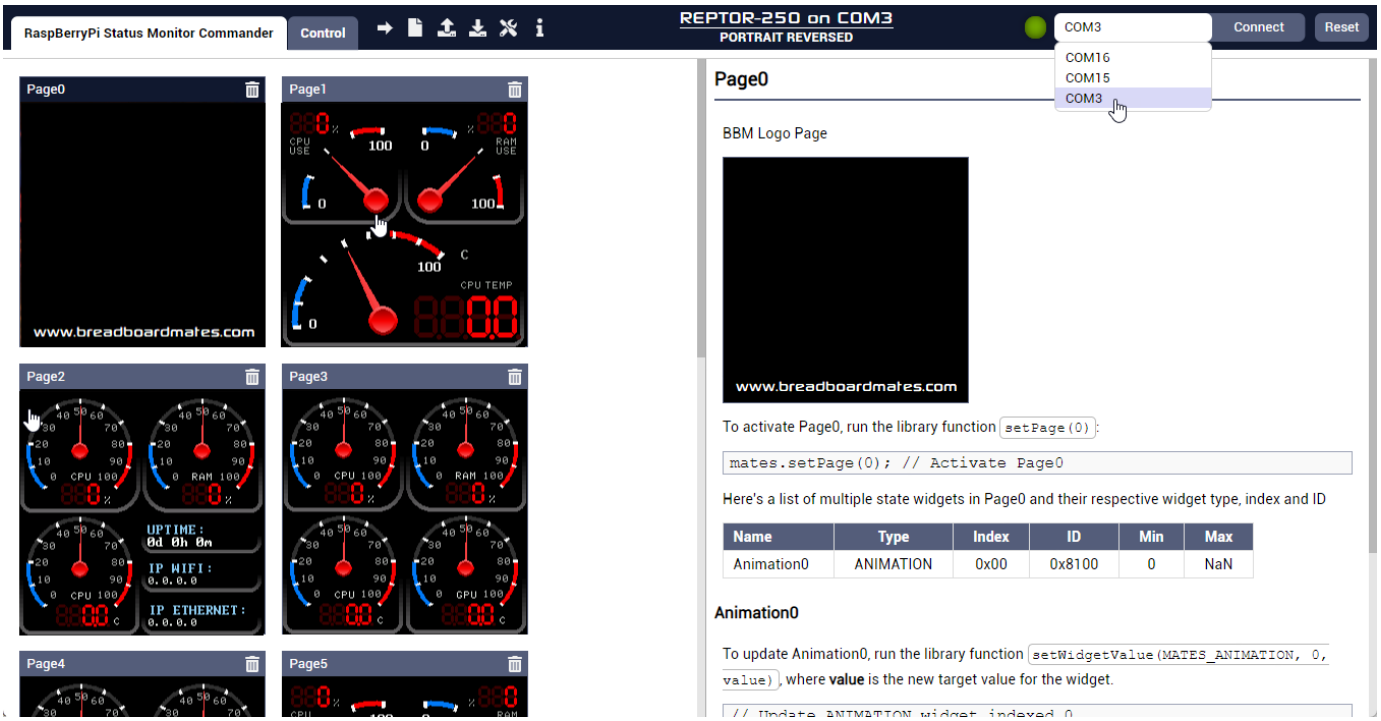
Step 1: When you start BBM Mates Studio you will be prompted to select your product. As we are using an already-created project, we can simply load the project from this screen.



Step 2: Simply Click on Browse Computer and navigate to the downloaded project file. The project will open in the Commander environment.



Step 3: To upload the project to the REPTOR we need to first select the correct COM port from the drop down menu.



Step 4: Finally, click on the Upload button to upload the Status Monitor to the REPTOR-250.

The Status Monitor will now be displayed on the REPTOR-250.

The USB lead and Mates Programmer can now be removed from the Pi Adapter. The Pi Adapter switch can now be set to HOST, ready to receive commands from the Pi.

Setting Up the Raspberry Pi

Raspberry Pi setup

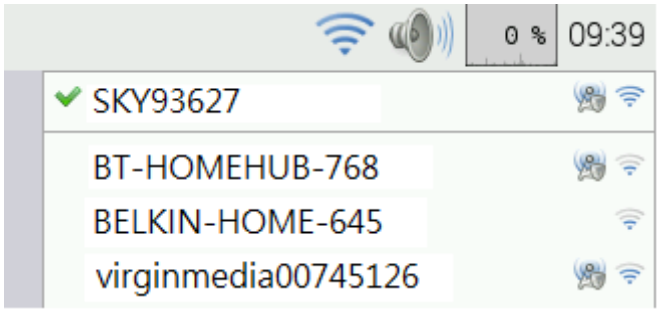
Setting up the Pi is very easy to achieve by visiting https://www.raspberrypi.com/software/ and following the instructions for installing the OS. You can choose a suitable OS for your own requirements, and we will cover a Desktop setup or if preferred, there is an excellent tutorial for a Headless setup here with a VNC viewer desktop.

When the OS has been transferred to the uSD card remove it from the PC and insert into the Pi and then power it up. Raspberry Pi's can consume quite a lot of current which may exceed the available current from a PC USB port so it may be necessary to use the official Raspberry Pi power supply.

The Pi OS will need to be configured to connect to the internet, SHH, and also to enable the Serial port (UART) that we will use to talk to the BBM REPTOR-250.

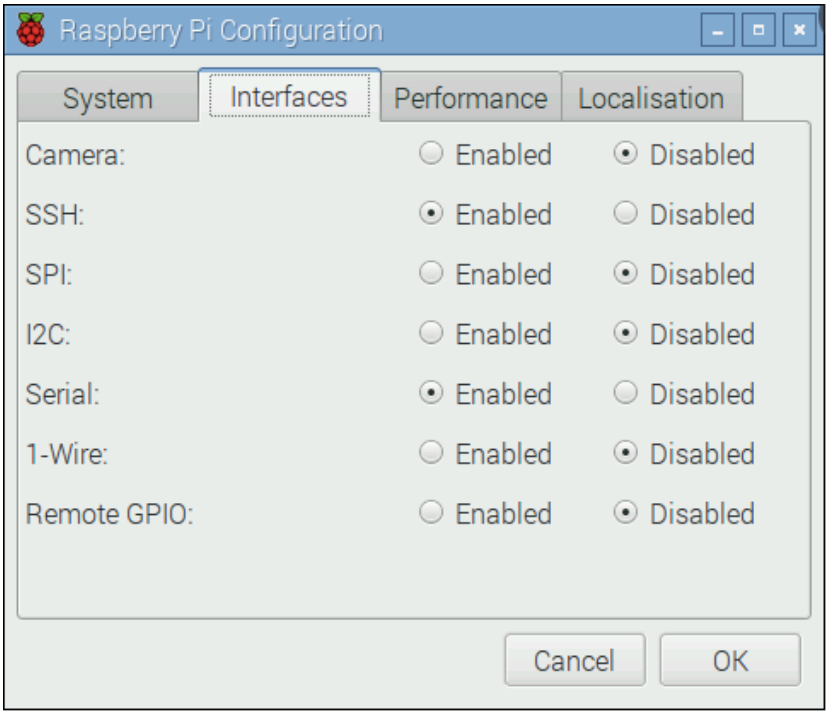
WiFi Setup

WiFi Setup can usually be done during the Raspberry Pi OS setup and can also be configured using the provided GUI to set up your WiFi connection.



SSH and Serial Setup

SSH and the Serial port can be enabled using the Raspberry Pi Configuration tool.



Installing the Python app

All recent Raspberry Pi OS Distro's are pre-loaded with **Python 3** so we can install the required Python libraries using PIP.

The psutil library can be installed by running:

```
pip3 install psutil
```

Next, we can install the Breadboard Mates Controller library by running the following command:

```
pip3 install rpi-mates-controller
```

Clone the Python code from Github by running the following command:

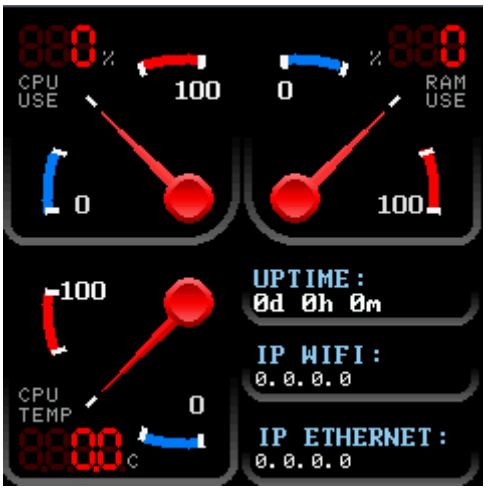
```
git clone https://github.com/BreadBoardMates/RPi-REPTOR-Status-Monitor.git
```

Project Discussion

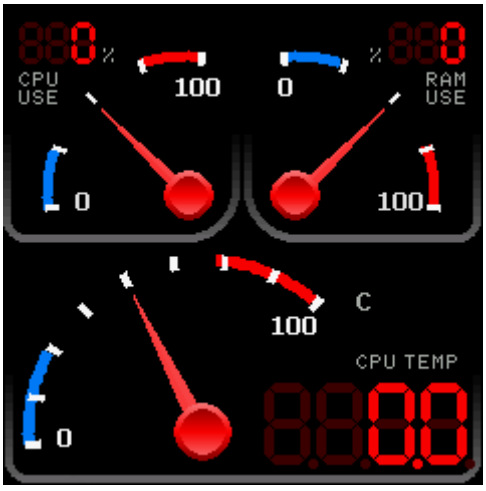
Commander Graphics Design

Page 0: Contains animation and image widgets which both need to source to display on the screen. The animation shows a 120 frames of BBM logo.

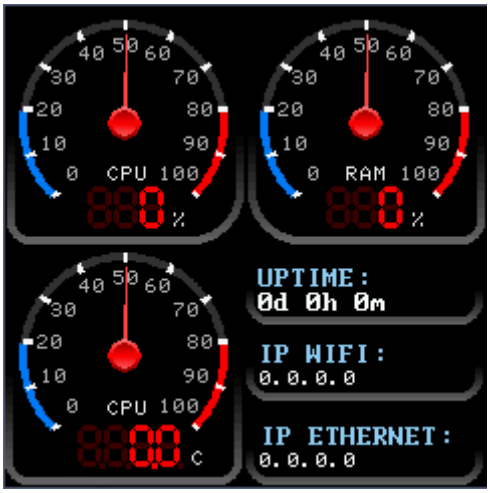
Page 1: Contains led digits, Media Gauges and Text Area for displaying CPU and RAM usage, CPU temperature and IP when using WiFi/Ethernet.



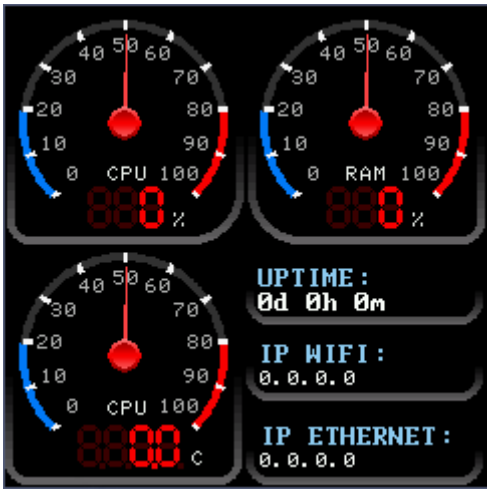
Page 2: Contains Media Gauges and Led Digits for displaying CPU and RAM usage, and CPU Temperature.



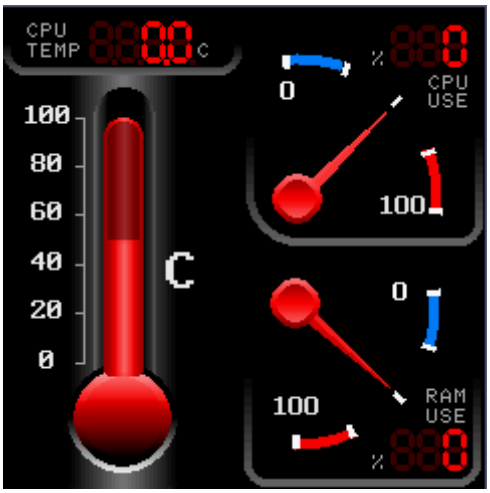
Page 3: Contains led digits, Media Gauges and Text Area for displaying CPU and RAM usage, CPU temperature and IP when using WiFi/Ethernet.



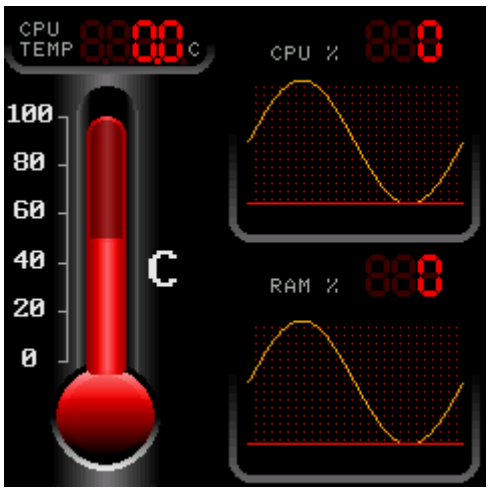
Page 4: Contains Media Gauges, and Led Digits for displaying CPU, RAM and HDD usage, and CPU temperature.



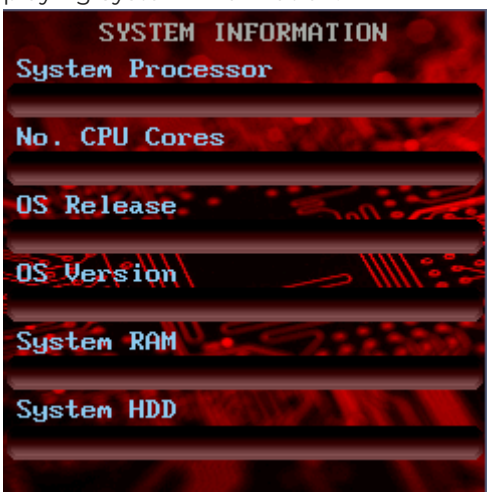
Page 5: Contains Themperature Widget and Led Digits to diplay CPU temperature. Media Gauge and Led Digits to display CPU and RAM usage.



Page 6: Contains Themperature Widget and Led Digits to diplay CPU temperature. Scope Widget and Led Digits to display CPU and RAM usage.



Page 7: Used TextArea Widget for displaying System Information.



Page 8: Used TextArea Widget for displaying WiFi Information.

The Python Program

The python script need to be execute to run the project. It is using the `psutil` module to gather information from the Raspberry Pi and display the result to REPTOR-250 using the mates module.

The program starts by defining helper functions in getting network and system information, uptime, temperature and storage information.

WiFi Info

```
def getSSID():
    ssid = os.popen("iwconfig wlan0 \
            | grep 'ESSID' \
            | awk '{print $4}' \
            | awk -F\\\" '{print $2}'").read()
    return ssid
```

System Info

```
def getSystemInfo():
    mates.updateTextArea(13, "Total number of Cores: " + str(psutil.cpu_count(logical=True),), True)
    mates.updateTextArea(14, platform.release(), True)
    mates.updateTextArea(15, str(platform.version()), True)
    mates.updateTextArea(12, str(platform.machine()), True)
    mates.updateTextArea(16, "Total RAM: "+ str(round(psutil.virtual_memory().total / (1024.0 **3)))+" GB",
True)
    hdd = psutil.disk_usage('/')
    mates.updateTextArea(17, "Total HDD Capacity: %d GB" % (hdd.total / (2**30)), True)
```

Network Information

```
def getNetworkInfo():
    mates.updateTextArea(18, getSSID(), True)
    mates.updateTextArea(19, socket.gethostname(), True)
    mates.updateTextArea(20, get_interface_ipaddress('wlan0'), True)
    mates.updateTextArea(21, ':'.join(re.findall('..', '%012x' % uuid.getnode())), True)
    iostat = psutil.net_io_counters(pernic=False)
    mates.updateTextArea(22, str(iostat[1]) + " bytes", True)
    mates.updateTextArea(23, str(iostat[0]) + " bytes")
```

Uptime

```
def up():
    t = int(time.clock_gettime(time.CLOCK_BOOTTIME))
    days = 0
    hours = 0
    min = 0
    out = ''
    days = int(t / 86400)
    t = t - (days * 86400)
    hours = int(t / 3600)
    t = t - (hours * 3600)
    min = int(t / 60)
    out += str(days) + 'd '
    out += str(hours) + 'h '
    out += str(min) + 'm'
    return out
```

Ethernet Info

```
def get_interface_ipaddress(network):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
                                struct.pack('256s',
                                network[:15].encode('utf-8')))[20:24])  # SIOCGIFADDR
    except OSError:
        return '0.0.0.0'
```

Temperature

```
def get_temp(sensor: str):
    cpu_temp = psutil.sensors_temperatures()
    cpu_temp = psutil.sensors_temperatures()[sensor]
    cpu_temp = psutil.sensors_temperatures()[sensor][0]
    return int(cpu_temp.current)
```

Storage

```
def get_hdd():
    hdd = psutil.disk_partitions(0)
    drive = psutil.disk_usage('/')
    percent = drive.percent
    return percent
```

It also needs additional helper functions for updating the display.

Set Page

```
def set_Page(newPage: int):
    mates.setPage(newPage, 3000)
    currPage = newPage
```

CPU usage

```
def set_GaugeLedDigitPercent(num: int, val: int):
    mates.setLedDigitsShortValue(num, val)
    mates.setWidgetValueByIndex(MatesWidget.MATES_MEDIA_GAUGE_B, num, val)
```

CPU Temp

```
def set_GaugeLedDigitTemp(num: int, val: int):
    mates.setLedDigitsShortValue(num, val)
    mates.setWidgetValueByIndex(MatesWidget.MATES_MEDIA_GAUGE_B, num, int(val / 10))
```

Update Page

```
def update_Page(page: int, data: int):
    if page == 1:
        if data == ALL_DATA or data == CPU_USE: set_GaugeLedDigitPercent(1, lastCpuUse)
        if data == ALL_DATA or data == CPU_TEMP: set_GaugeLedDigitTemp(0, lastlTemp)
        if data == ALL_DATA or data == RAM_USE: set_GaugeLedDigitPercent(2, lastRamUse)
    if page == 2:
        if data == ALL_DATA or data == CPU_USE: set_GaugeLedDigitPercent(4, lastCpuUse)
        if data == ALL_DATA or data == CPU_TEMP: set_GaugeLedDigitTemp(3, lastlTemp)
        if data == ALL_DATA or data == GPU_TEMP: set_GaugeLedDigitTemp(6, lastlTempG)
        if data == ALL_DATA or data == RAM_USE: set_GaugeLedDigitPercent(5, lastRamUse)
    if page == 3:
        if data == ALL_DATA or data == CPU_USE: set_GaugeLedDigitPercent(7, lastCpuUse)
        if data == ALL_DATA or data == CPU_TEMP: set_GaugeLedDigitTemp(9, lastlTemp)
        if data == ALL_DATA or data == RAM_USE: set_GaugeLedDigitPercent(8, lastRamUse)
    if page == 4:
        if data == ALL_DATA or data == CPU_USE: set_GaugeLedDigitPercent(10, lastCpuUse)
        if data == ALL_DATA or data == CPU_TEMP: set_GaugeLedDigitTemp(12, lastlTemp)
        if data == ALL_DATA or data == HDD_USE: set_GaugeLedDigitPercent(13, lastHDD)
        if data == ALL_DATA or data == RAM_USE: set_GaugeLedDigitPercent(11, lastRamUse)
    if page == 5:
        if data == ALL_DATA or data == CPU_USE: set_GaugeLedDigitPercent(14, lastCpuUse)
        if data == ALL_DATA or data == CPU_TEMP:
            mates.setWidgetValueByIndex(MatesWidget.MATES_MEDIA_THERMOMETER, 0, int(lastlTemp / 10))
            mates.setLedDigitsShortValue(16, lastlTemp)
        if data == ALL_DATA or data == RAM_USE: set_GaugeLedDigitPercent(15, lastRamUse)
    if page == 6:
        if data == ALL_DATA or data == CPU_USE: mates.setLedDigitsShortValue(18, lastCpuUse)
        if data == CPU_USE_CONT: mates.setWidgetValueByIndex(MatesWidget.MATES_SCOPE, 0, int(lastCpuUse*29/
5/10))
        if data == ALL_DATA or data == CPU_TEMP:
            mates.setWidgetValueByIndex(MatesWidget.MATES_MEDIA_THERMOMETER, 1, int(lastlTemp / 10))
            mates.setLedDigitsShortValue(17, lastlTemp)
        if data == ALL_DATA or data == RAM_USE: mates.setLedDigitsShortValue(19, lastRamUse)
        if data == RAM_USE_CONT: mates.setWidgetValueByIndex(MatesWidget.MATES_SCOPE, 1, int(lastRamUse*29/
5/10))
```

Ready Page

```
def set_PageReady(page: int):
    if page < 7: update_Page(currPage, ALL_DATA)
    if page == 7: getSystemInfo()
    if page == 8: getNetworkInfo()
```

The Python code below will create a Mates Controller instance and start it at 115200 baud.

```
mates = MatesController('/dev/ttyS0')
mates.begin(115200)
```

A set of variables are then created which will be set after each time the various states are updated. This enables the main loop to compare the status that has just read with its last state and then update the corresponding widget only if it has changed in value.

```
if __name__ == '__main__':

    mates = MatesController('/dev/ttyS0')

    mates.begin(115200)

    gtime = up()
    lastCpuUse = 0
    lastTemp = 0
    lastTempG = 0
    lastlTemp = 0
    lastlTempG = 0
    lastRamUse = 0
    lastHDD = 0
    lastWIPaddr = '0.0.0.0'
    lastEIPaddr = '0.0.0.0'
    lastPage = -1
    currPage = 0
    lastbytesRX = 0
    lastbytesTX = 0
    MaxPage = 8
    RefreshVal = 0

    showLogo = 1

    lgpu = 50.5

    mates.updateTextArea(3, get_interface_ipaddress('wlan0'))
    mates.updateTextArea(9, get_interface_ipaddress('wlan0'))

    mates.updateTextArea(5, gtime, True)
    mates.updateTextArea(11, gtime, True)

    IPinterval = 0

    lcpu = int(get_temp("cpu_thermal") * 10)
    lgpu = int(get_temp("cpu_thermal") * 10)
    #lgpu = int(get_temp("gpu_thermal") * 10)

    IPinterval = 0
```

In the loop, it will check for the current page that will show the right information for the page. At the same time if swiped is detected either right or left swipe will update the dispay.

```
while True:

    currPage = mates.getPage()
    if lastPage != currPage:
        if currPage == 0 and showLogo == 1:
            for n in range(0, 74):
                mates.setWidgetValueByIndex(MatesWidget.MATES_ANIMATION, 0, n)
            time.sleep(3.000)
            for n in range(74, 120):
                mates.setWidgetValueByIndex(MatesWidget.MATES_ANIMATION, 0, n)
            time.sleep(2.000)
            set_Page(1)
            showLogo = 0
        if lastPage != -1: RefreshVal = 1
```

```
        lastPage = currPage

    swiped = mates.getSwipeEventCount()
    if swiped > 0:
        swipe = mates.getNextSwipeEvent()
        if swipe == MatesSwipeConsts.MATES_SWIPE_WEST:
            currPage = currPage + 1
            if currPage > MaxPage: currPage = 1
            set_PageReady(currPage)
            set_Page(currPage)
        if swipe == MatesSwipeConsts.MATES_SWIPE_EAST:
            currPage = currPage - 1
            if currPage < 1: currPage = MaxPage
            set_PageReady(currPage)
            set_Page(currPage)

    lcpu = int(get_temp("cpu_thermal") * 10)
    lgpu = int(get_temp("cpu_thermal") * 10)
    #lgpu = int(get_temp("gpu_thermal") * 10)

    cpuuse = int(psutil.cpu_percent())
    ramuse = int(psutil.virtual_memory().percent)
    hdd = get_hdd()

    if currPage == 8:
        iostat = psutil.net_io_counters(pernic=False)
        if lastbytesRX != iostat[1]:
            lastbytesRX = iostat[1]
            mates.updateTextArea(22, str(iostat[1]) + " bytes", True)
        if lastbytesTX != iostat[0]:
            lastbytesTX = iostat[0]
            mates.updateTextArea(23, str(iostat[0]) + " bytes", True)

    if currPage == 6:
        update_Page(currPage, CPU_USE_CONT)
        update_Page(currPage, RAM_USE_CONT)

    if cpuuse != lastCpuUse:
        lastCpuUse = lastCpuUse - increment(cpuuse, lastCpuUse)
        update_Page(currPage, CPU_USE)

    if lcpu != lastlTemp:
        lastlTemp = lastlTemp - increment(lcpu, lastlTemp)
        update_Page(currPage, CPU_TEMP)

    if lgpu != lastlTempG:
        lastlTempG = lastlTempG - increment(lgpu, lastlTempG)
        update_Page(currPage, GPU_TEMP)

    if ramuse != lastRamUse:
        lastRamUse = lastRamUse - increment(ramuse, lastRamUse)
        update_Page(currPage, RAM_USE)

    if hdd != lastHDD:
        lastHDD = lastHDD - increment(hdd, lastHDD)
        update_Page(currPage, HDD_USE)

    if IPinterval > 20 or RefreshVal == 1:
        tempIPaddr = get_interface_ipaddress('eth0')
        if tempIPaddr != lastEIPaddr or RefreshVal == 1:
            if currPage == 1: mates.updateTextArea(1, tempIPaddr)
            if currPage == 3: mates.updateTextArea(7, tempIPaddr)
            lastEIPaddr = tempIPaddr
            if RefreshVal != 1: getNetworkInfo()
        tempIPaddr = get_interface_ipaddress('wlan0')
        if tempIPaddr != lastWIPaddr or RefreshVal == 1:
            if currPage == 1: mates.updateTextArea(3, tempIPaddr)
            if currPage == 3: mates.updateTextArea(9, tempIPaddr)
            lastWIPaddr = tempIPaddr
            if RefreshVal != 1: getNetworkInfo()
```

```
        IPinterval = 0

    IPinterval = IPinterval + 1
    time.sleep(0.040)

    tempTime = up()
    if tempTime != gtime or RefreshVal == 1:
        if currPage == 1: mates.updateTextArea(5, gtime, True)
        if currPage == 3: mates.updateTextArea(11, gtime, True)
        gtime = tempTime

    RefreshVal = 0

    time.sleep(0.020)
```

Running the Project

Go to RPi-REPTOR-Status-Monitor folder:

```
cd RPi-REPTOR-Status-Monitor
```

Then run the application by running the following command:

```
python3 RPiReptorStatusMonitor.py
```

The REPTOR-250 should first reset to an off screen and then start showing the boot logo followed by status of CPU use, CPU temp and RAM use along with connected IP address and uptime. You can then swipe through the available pages.

Enjoy your shiny Raspberry Pi monitor

This project can be simply altered or improved to get the desired look by creating a new page in Mates Studio and changing the Python code to match with any new widgets used. The only limit is imagination.

Downloadable Resources

Here are the links to the software applications, libraries and completed project files.

- Mates Studio
- Python Mates Controller Library
- Project Files