MODULE 4

IoT Physical Devices & End points: What is a loT Device, Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi interfaces, Programming Raspberry Pi with Python, Case Studies illustrating IoT design – Home Automation, Cities, Agriculture.

4.1 What is an IoT Device?

As described earlier, a "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phones, smart TV, computer, refrigerator, car, etc.). IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g., information sensed by the connected sensor) over a network (to other devices or servers/storage) or allow activation upon the physical entities/environment around them remotely. Some examples of IoT devices are listed below:

- A home automation device that allows remotely monitoring the status of appliances and connected machines.
- An industrial machine which sends information about its operation and health monitoring data to a server.
- A location-tracking information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

4.1.1 Basic Building Blocks of an IoT Device

An IoT device can consist of a number of modules based on functional attributes, such as:

- **Sensing:** Sensors can either be onboard IoT itself or be attached to the device. IoT device can collect various types of information from the onboard sensors or actuators and send the data to a server for storage and processing. For example, a temperature sensor can sense room temperature and transmit the data to a server.
- **Communication:** Communication modules are responsible for sending/collecting data to/from other devices or cloud-based servers and receive/send control commands.
- **Data Processing:** Analysis and processing modules are responsible for making sense of the data.

A typical IoT device will be a combination of these blocks. One of the most widely used single-board microcontroller-based IoT device (for explaining and testing IoT concepts) is the Raspberry Pi. It is common since these devices are widely accessible, inexpensive, and available from multiple vendors.

Furthermore, extensive information is available on their usage, programming, and use both on the Internet and in other textbooks.

4.2 Exemplary Device: Raspberry Pi

While primarily this discusses generic IoT devices, the same concepts apply to proprietary and specialized devices such as the **Raspberry Pi**. Before diving into the specifics of the Raspberry Pi, let's revisit the core components that make up a **single-board computer (SBC)** used as an IoT device.

A single-board computer (SBC) based IoT device typically includes the following key components:

- **Processor (CPU)** – The central unit that handles computation and logic.
- **Graphics (GPU)** – Handles rendering of visual content.
- **Memory Interfaces** – Includes NAND/NOR flash and DDR1/DDR2/DDR3 for storage and processing memory.
- **Storage Interfaces** – Interfaces like SDIO for external storage.
- **Audio/Video** – HDMI, 3.5 mm audio jack, and RCA video output for multimedia.
- **Connectivity** – USB host and RJ45/Ethernet for network and peripheral connectivity.
- **Interfaces** – UART, SPI, I2C, DC, CAN for communication with external modules.

Figure 4.1 shows a typical block diagram of such a system, highlighting the integration of CPU, GPU, RAM, storage units, and I/O interfaces.

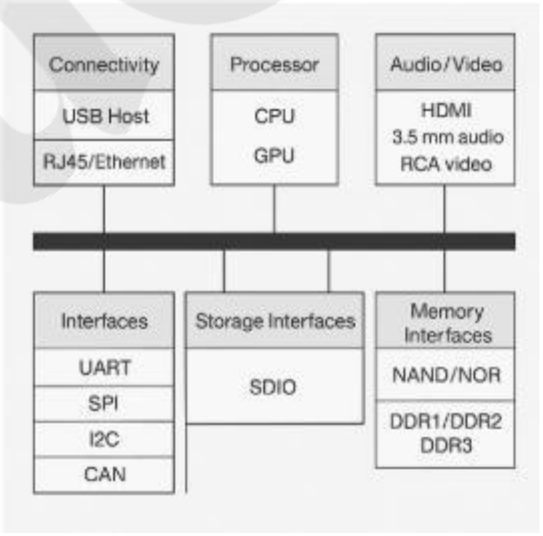


Figure 4.1: Block Diagram of an IoT device

Overview of Raspberry Pi

The **Raspberry Pi** is a compact, low-cost single-board computer, roughly the size of a credit card. It supports a wide range of functions, from basic computing tasks to more complex operations involving sensors and IoT components.

Key highlights:

- Capable of running various **flavors of Linux**, making it highly versatile.
- Equipped with **multiple interfaces** such as USB, GPIO pins, HDMI, and Ethernet for easy integration with other devices.
- Can be used as a **dedicated IoT system**, performing tasks such as collecting sensor data or managing control systems.
- Because it runs a Linux-based OS, it supports **Python** and other programming languages widely used in IoT development.

The Raspberry Pi stands out for its flexibility, cost-efficiency, and ease of use, making it a preferred platform for learning, prototyping, and deploying IoT applications.

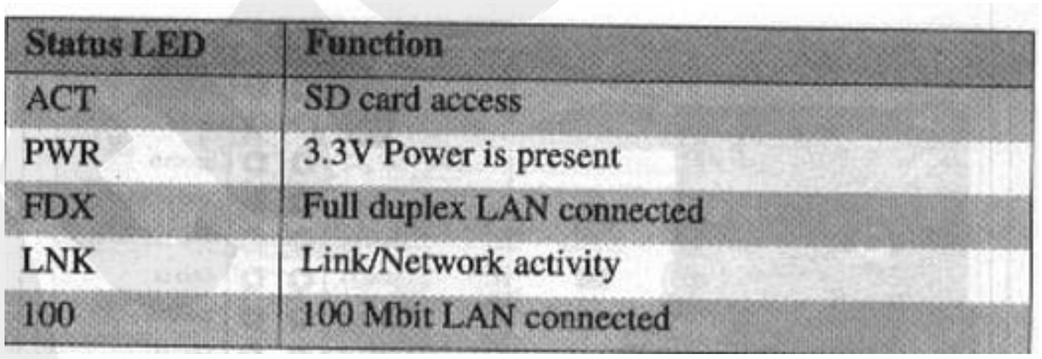
4.3 About the Board

Figure 4.2 illustrates the Raspberry Pi board, highlighting various components and peripherals. Here's an overview of its key features:

- **Processor & RAM**:
  The Raspberry Pi is built on an ARM architecture. The latest version (Model B, Revision 2.0) is equipped with a 700 MHz Low Power ARM1176JZF-S processor and 512 MB of SDRAM.
- **USB Ports**:
  It includes two USB 2.0 ports. These ports can provide a combined current of 100mA. For peripherals that require more power, a powered USB hub can be connected.
- **Ethernet**:
  A standard 10/100 Ethernet port is included for wired networking. Alternatively, a USB Wi-Fi adapter can be used for wireless connectivity.
- **HDMI Output**:
  The HDMI port provides high-quality video and audio output. This enables the board to be connected directly to HDMI displays like monitors or TVs. With a converter, it can also connect to DVI-compatible displays.
- **Composite Video Output**:
  In addition to HDMI, the Raspberry Pi supports composite video output through a standard RCA jack for older TVs and monitors that don't support HDMI.

- **GPIO Pins**:
  The board includes multiple General Purpose Input/Output (GPIO) pins, which can be programmed for a variety of custom hardware interfacing tasks such as controlling LEDs, reading sensor data, etc.
- **DSI (Display Serial Interface)**:
  The DSI port can be used to connect a dedicated Raspberry Pi LCD touchscreen.
- **Camera Serial Interface (CSI)**:
  This interface supports connection to the Raspberry Pi Camera Module for capturing images and video.
- **Audio Output**:
  A standard 3.5 mm audio jack is available for analog audio output, compatible with headphones and most speakers.
- **LED Indicators**:
  The board features status LEDs. For example, Model B has two LEDs that indicate system power and activity.
- **SD Card Slot**:
  The Raspberry Pi doesn’t have onboard storage. An SD card is required to boot and store the operating system and user data. Typically, a minimum of 8 GB is recommended when using setups like NOOBS (New Out Of the Box Software).
- **Power Input**:
  Power is supplied via a micro-USB connector, typically from a 5V mobile charger or power adapter.

Table4.1: Rasberry Pi Status LEDs



Status LED	Function
ACT	SD card access
PWR	3.3V Power is present
FDX	Full duplex LAN connected
LNK	Link/Network activity
100	100 Mbit LAN connected

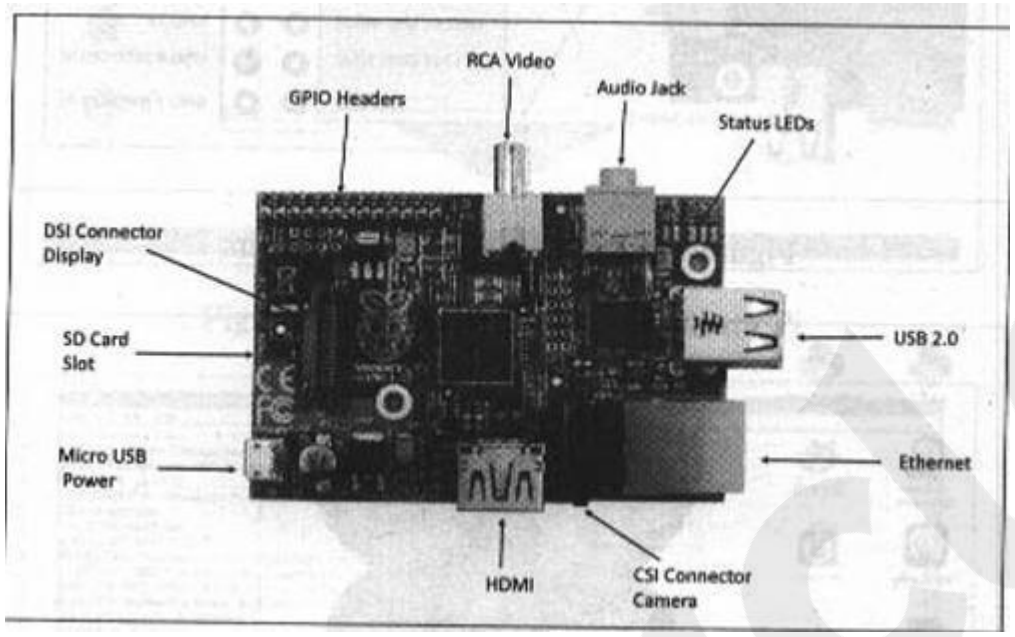


Fig 4.2 Rasberry Pi Board

4.4 Linux on Raspberry Pi

Raspberry Pi is a small, affordable computer that runs on Linux-based operating systems. It is widely used in education, programming, IoT projects, and prototyping. Installing and using Linux on Raspberry Pi helps users learn about system administration, networking, and basic scripting.

Raspberry Pi supports various flavors of Linux including:

- **Raspbian**: Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi. This is the recommended Linux for Raspberry Pi. Appendix-1 provides instructions on setting up Raspbian on Raspberry Pi.
- **Arch**: Arch is an Arch Linux port for AMD devices.
- **Pidora**: Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
- **RaspBMC**: RaspBMC is an XBMC media-center distribution for Raspberry Pi.
- **OpenELEC**: OpenELEC is a fast and user-friendly XBMC media-center distribution.
- **RISC OS**: RISC OS is a very fast and compact operating system.

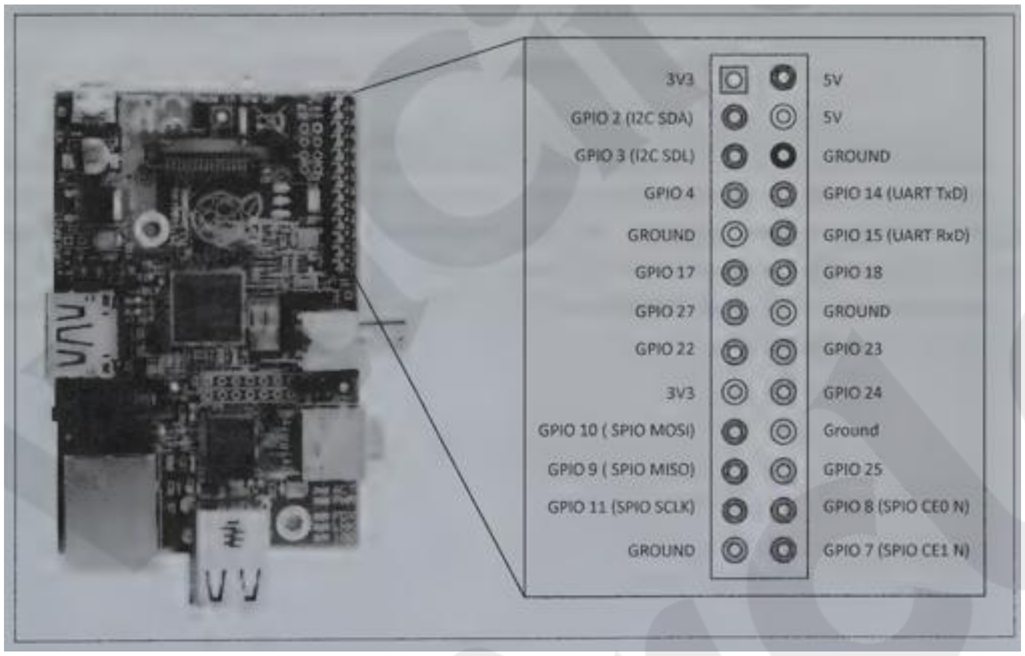


Fig 4.3 Rasberry Pi GPIO headers

Figure 4.3 shows the Rasberry Pi GPIO headers.Figure 4.4 shows the Raspbian Linux desktop on Raspberry Pi. Figure 4.5 shows the default file explorer on Raspbian. Figure 4.6 shows the default console on Raspbian. Figure 4.7 shows the default browser on Raspbian. To configure Raspberry Pi, the raspi-config tool is used which can be launched from command line as ($raspi-config) as shown in Figure 4.8. Using the configuration tool you can expand root partition to fill SD card, set keyboard layout, change password, set locale and timezone, change memory split, enable or disable SSH server and change boot behavior. It is recommended to expand the root file-system so that you can use the entire space on the SD card.

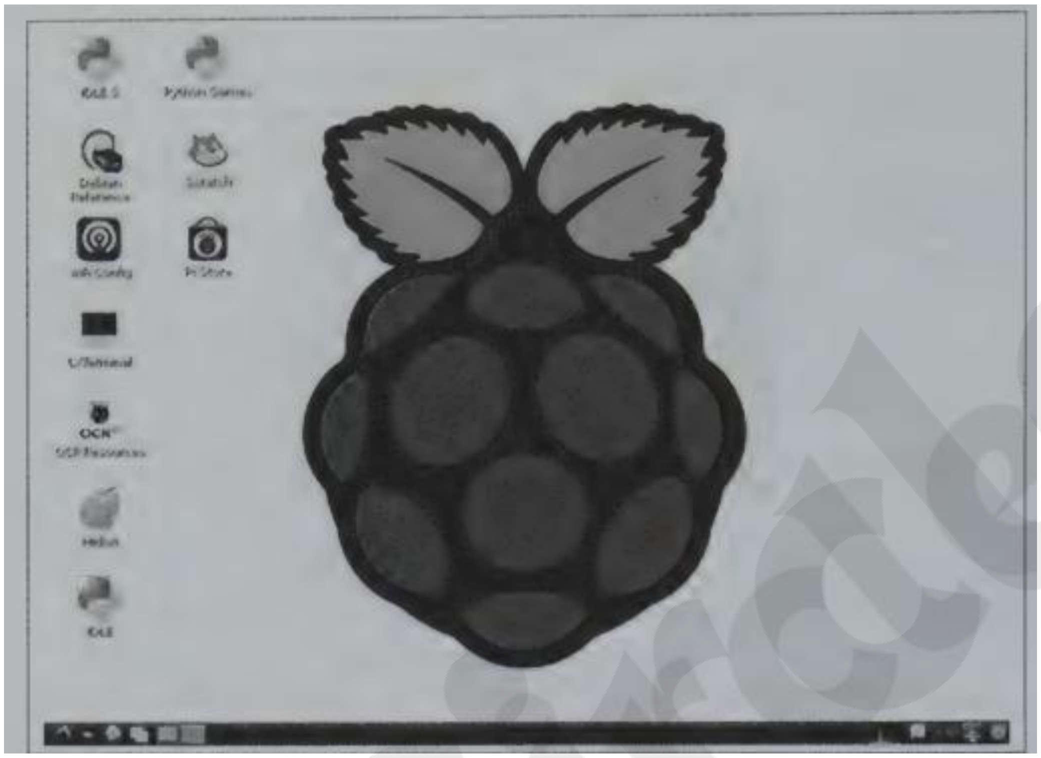


Figure 4.4:Rasbian Linux Desktop

Though Raspberry Pi comes with an HDMI output, it is more convenient to access the device with a VNC connection or SSH. This does away with the need for a separate display for Raspberry Pi and you can use Raspberry Pi from your desktop or laptop computer. Appendix-A provides instructions on setting up VNC server on Raspberry Pi and the instructions to connect to Raspberry Pi with SSH. Table 4.2 lists the frequently used commands on Raspberry Pi.

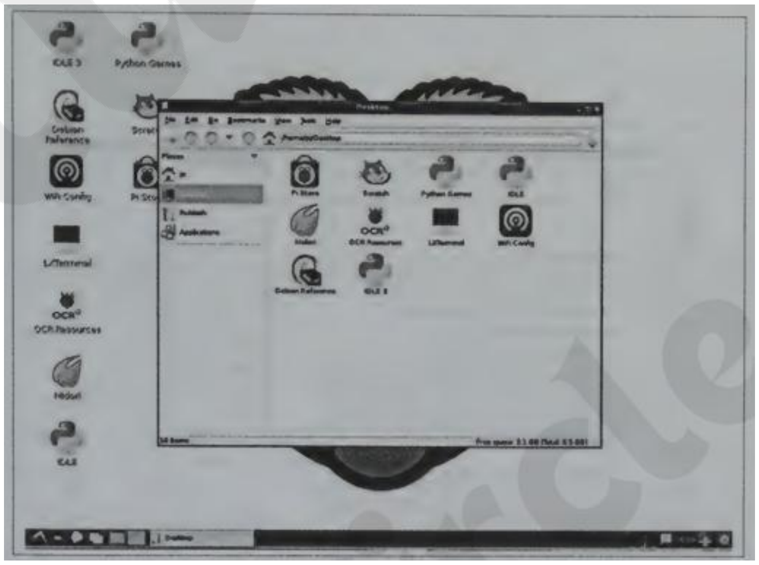
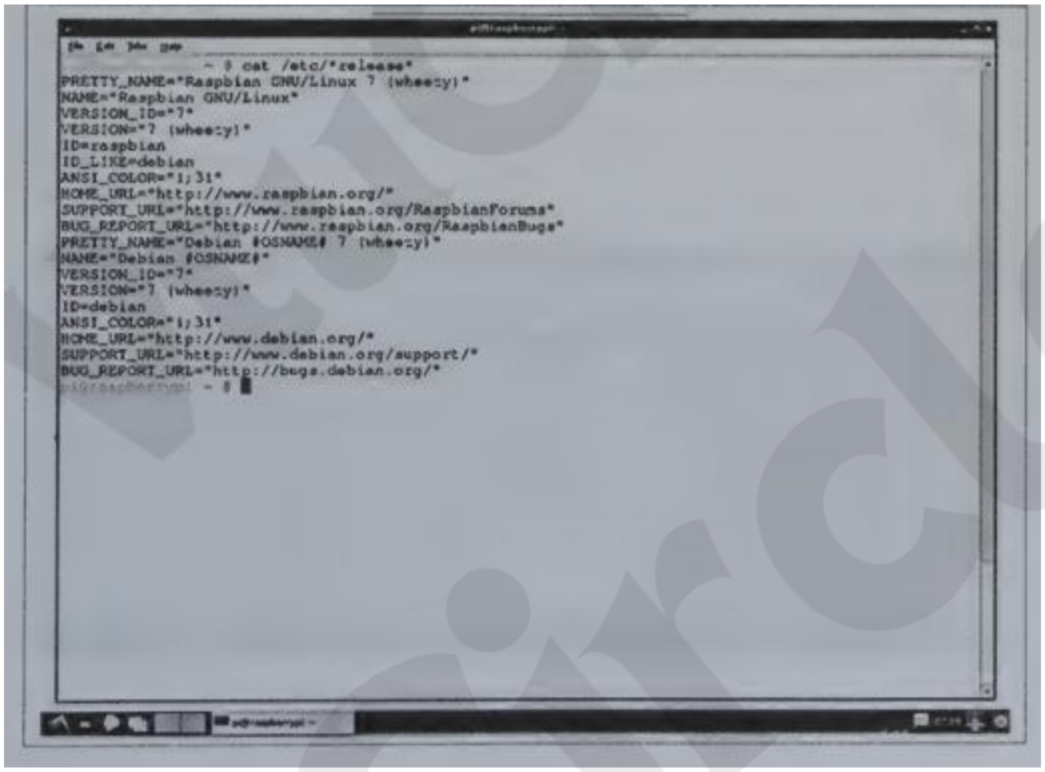
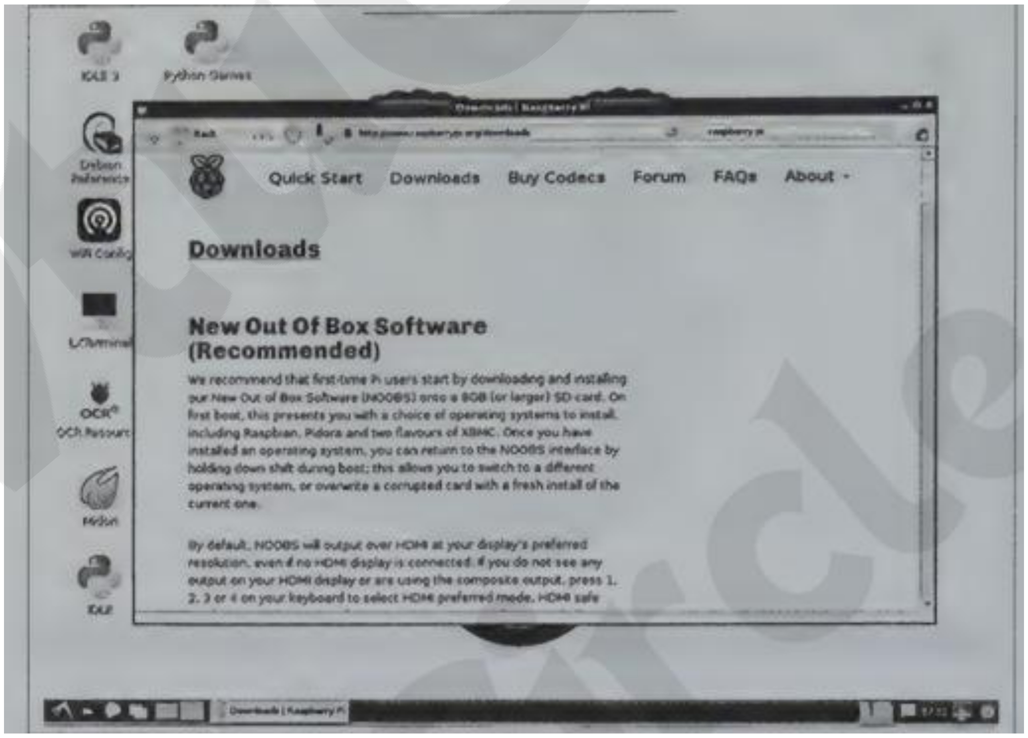


Figure 4.5:File Explorer on Rasberry Pi



```
                   ~ $ cat /etc/*release*
PRETTY_NAME="Raspbian GNU/Linux 7 (wheezy)"
NAME="Raspbian GNU/Linux"
VERSION_ID="7"
VERSION="7 (wheezy)"
ID=raspbian
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
PRETTY_NAME="Debian #OSNAME# 7 (wheezy)"
NAME="Debian #OSNAME#"
VERSION_ID="7"
VERSION="7 (wheezy)"
ID=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support/"
BUG_REPORT_URL="http://bugs.debian.org/"
pi@raspberrypi ~ $
```

Figure 4.6:Console on Rasberry Pi



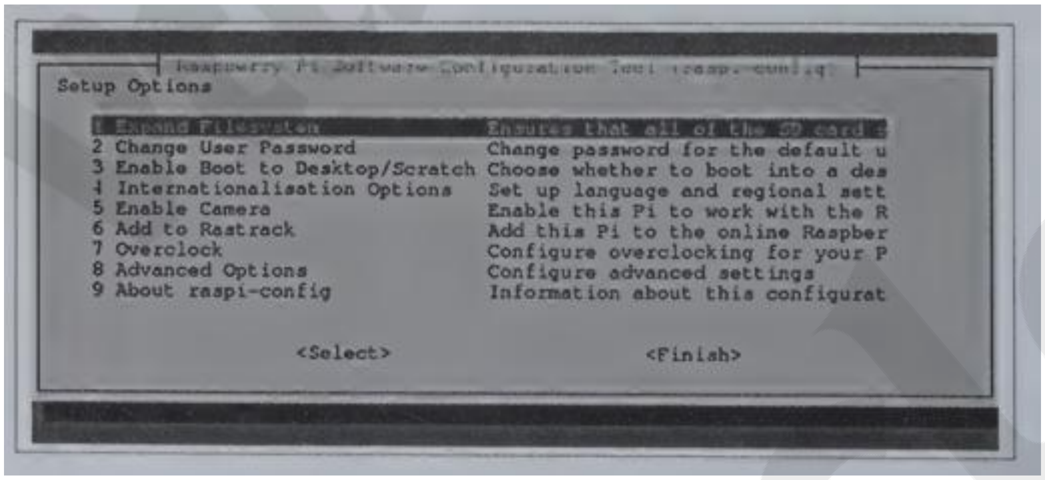
Downloads

New Out Of Box Software (Recommended)

We recommend that first-time Pi users start by downloading and installing our New Out of Box Software (NOOBS) onto a 8GB (or larger) SD card. On first boot, this presents you with a choice of operating systems to install, including Raspbian, Pidora and two flavours of XBMC. Once you have installed an operating system, you can return to the NOOBS interface by holding down shift during boot; this allows you to switch to a different operating system, or overwrite a corrupted card with a fresh install of the current one.

By default, NOOBS will output over HDMI at your display's preferred resolution, even if no HDMI display is connected. If you do not see any output on your HDMI display or are using the composite output, press 1, 2, 3 or 4 on your keyboard to select HDMI preferred mode, HDMI safe

Figure 4.7: Browser on Rasberry Pi

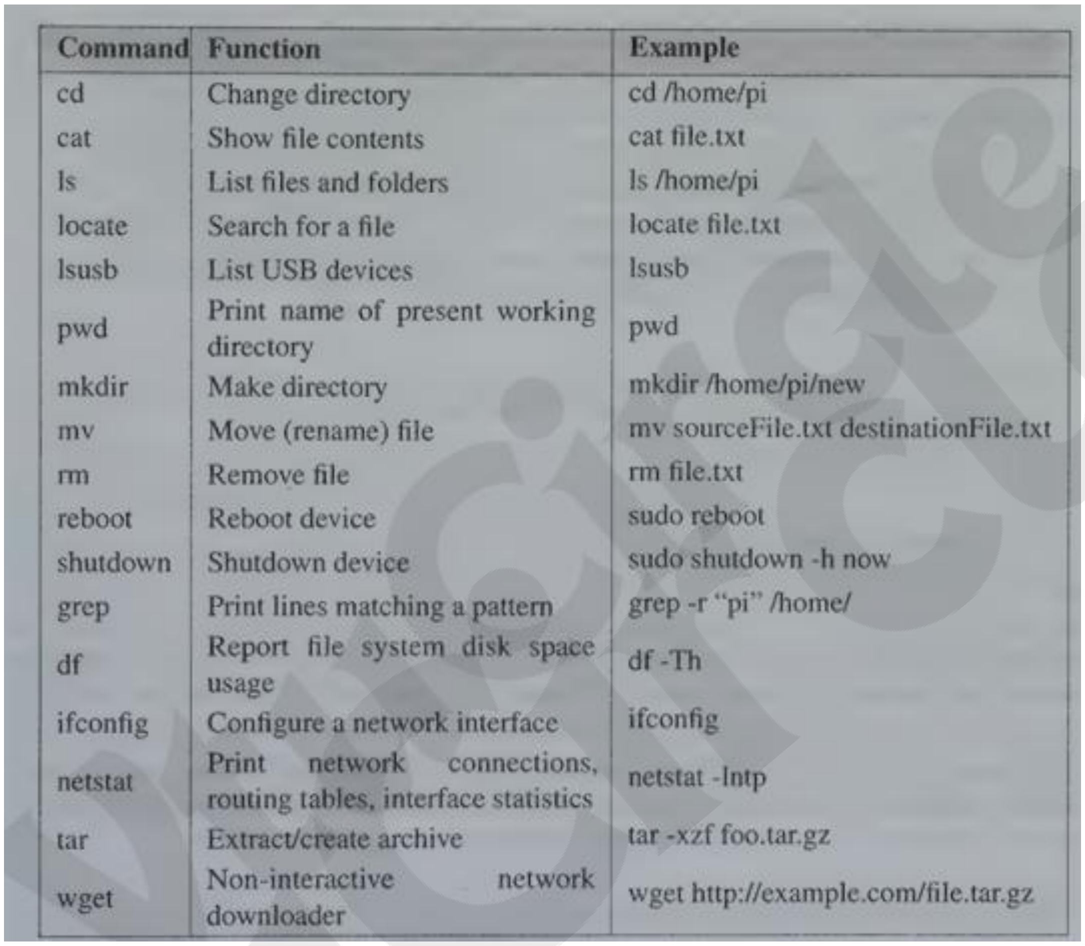


```
Raspberry Pi Software Configuration Tool (raspi-config)
Setup Options

1 Expand Filesystem              Ensures that all of the SD card s
2 Change User Password           Change password for the default u
3 Enable Boot to Desktop/Scratch Choose whether to boot into a des
4 Internationalisation Options   Set up language and regional sett
5 Enable Camera                  Enable this Pi to work with the R
6 Add to Rastrack                Add this Pi to the online Raspber
7 Overclock                      Configure overclocking for your P
8 Advanced Options               Configure advanced settings
9 About raspi-config             Information about this configurat

              <Select>                      <Finish>
```

Figure 4.8:Rasberry Pi Configuration Tool

Basic Linux Commands on Raspberry Pi
The following table lists common Linux commands used on Raspberry Pi, along with their functions and examples:

Table 4.2 Rasberry Pi frequently used commands



Command	Function	Example
cd	Change directory	cd /home/pi
cat	Show file contents	cat file.txt
ls	List files and folders	ls /home/pi
locate	Search for a file	locate file.txt
lsusb	List USB devices	lsusb
pwd	Print name of present working directory	pwd
mkdir	Make directory	mkdir /home/pi/new
mv	Move (rename) file	mv sourceFile.txt destinationFile.txt
rm	Remove file	rm file.txt
reboot	Reboot device	sudo reboot
shutdown	Shutdown device	sudo shutdown -h now
grep	Print lines matching a pattern	grep -r "pi" /home/
df	Report file system disk space usage	df -Th
ifconfig	Configure a network interface	ifconfig
netstat	Print network connections, routing tables, interface statistics	netstat -lntp
tar	Extract/create archive	tar -xzf foo.tar.gz
wget	Non-interactive network downloader	wget http://example.com/file.tar.gz

4.5 Rasberry Pi Interfaces

Raspberry Pi has serial, SPI and I2C interfaces for data transfer as shown in Figure 4.3.

4.5.1 Serial

The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.

4.5.2 SPI

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices. In an SPI connection, there is one master device and one or more peripheral devices. There are five pins on Raspberry Pi for SPI interface:

- MISO (Master In Slave Out) : Master line for sending data to the peripherals.

- MOSI (Master Out Slave In) : Slave line for sending data to the master.
- SCK (Serial Clock) : Clock generated by master to synchronize data transmission
- CEO (Chip Enable 0) : To enable or disable devices.
- CEO (Chip Enable 1) : To enable or disable devices.

4.5.3 12C

The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

4.6 Programming Raspberry Pi with Python

In this section you will learn how to get started with developing Python programs on Raspberry Pi. Raspberry Pi runs Linux and supports Python out of the box. Therefore, you can run any Python program that runs on a normal computer. However, it is the general purpose input/output capability provided by the GPIO pins on Raspberry Pi that makes it useful device for Internet of Things. You can interface a wide variety of sensor and actuators with Raspberry Pi using the GPIO pins and the SPI, I2C and serial interfaces. Input from the sensors connected to Raspberry Pi can be processed and various actions can be taken, for instance, sending data to a server, sending an email, triggering a relay switch.

4.6.1 Controlling LED with Rasberry Pi

Let us start with a basic example of controlling an LED from Raspberry Pi. Figure 7.9 shows the schematic diagram of connecting an LED to Raspberry Pi. Box 7.1 shows how to turn the LED on/off from command line. In this example the LED is connected to GPIO pin 18. You can connect the LED to any other GPIO pin as well. Box 7.2 shows a Python program for blinking an LED connected to Raspberry Pi every second. The program uses the RPi. GPIO module to control the GPIO on Raspberry Pi. In this program we set pin 18 direction to output and then write True/False alternatively after a delay of one second.

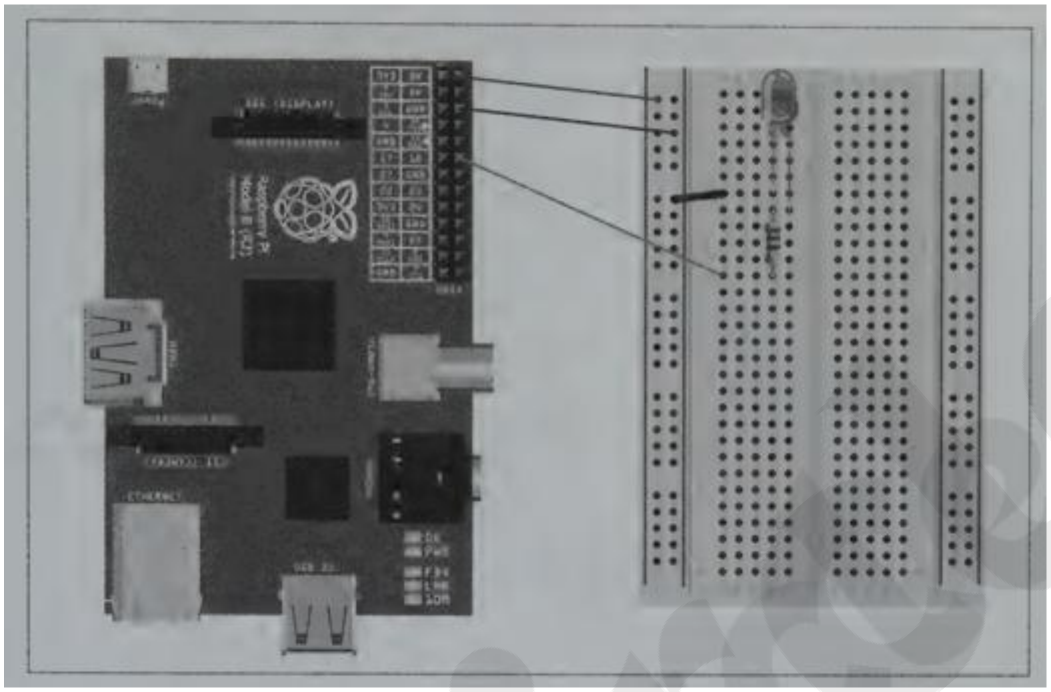


Figure 4.9: Controlling LED with Rasberry Pi

Box 4.1: Switching LED on/off from Raspberry Pi console

```
$ echo 18 > /sys/class/gpio/export
$ cd /sys/class/gpio/gpio18

# Set pin 18 direction to out
$ echo out > direction

# Turn LED on
$ echo 1 > value
# Turn LED off
$ echo 0 > value
```

Box 4.2: Python program for blinking LED

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.OUT)
```

```
while True:

    GPIO.output(18, True)

    time.sleep(1)

    GPIO.output(18, False)

    time.sleep(1)
```

4.6.2 Interfacing an LED and Switch with Raspberry Pi

Now let us look at a more detailed example involving an LED and a switch that is used to control the LED.

In this example the LED is connected to GPIO pin 18 and the switch is connected to pin 25. In the infinite while loop the state of GPIO 25 is checked and the state of LED is toggled if the switch is pressed. This example shows how to get input from GPIO pins and process the input and take some action. You can modify this example to log the state of an LED. Let us look at another example, in which the action is sending an email alert. This program uses the Python SMTP library for sending an email when the switch connected to Raspberry Pi is pressed.

Box 4.3: Python program for controlling an LED with a switch

```
from time import sleep

import RPi.GPIO as GPIO

GPIO.setmode (GPIO.BCM)

#Switch Pin

GPIO.setup(25, GPIO.IN)

#LED Pin

GPLO.setup (18, GPIO.OUT)

state=false

def  toggleLED (pin):
```

```
state =-not state

GPIO.output (pin, state)

while True:

try:

if (GPIO. input.(25) ==True):

toggleLED (pin)

sleep (.01)

except KeyboardInterrupt:

exit ()
```

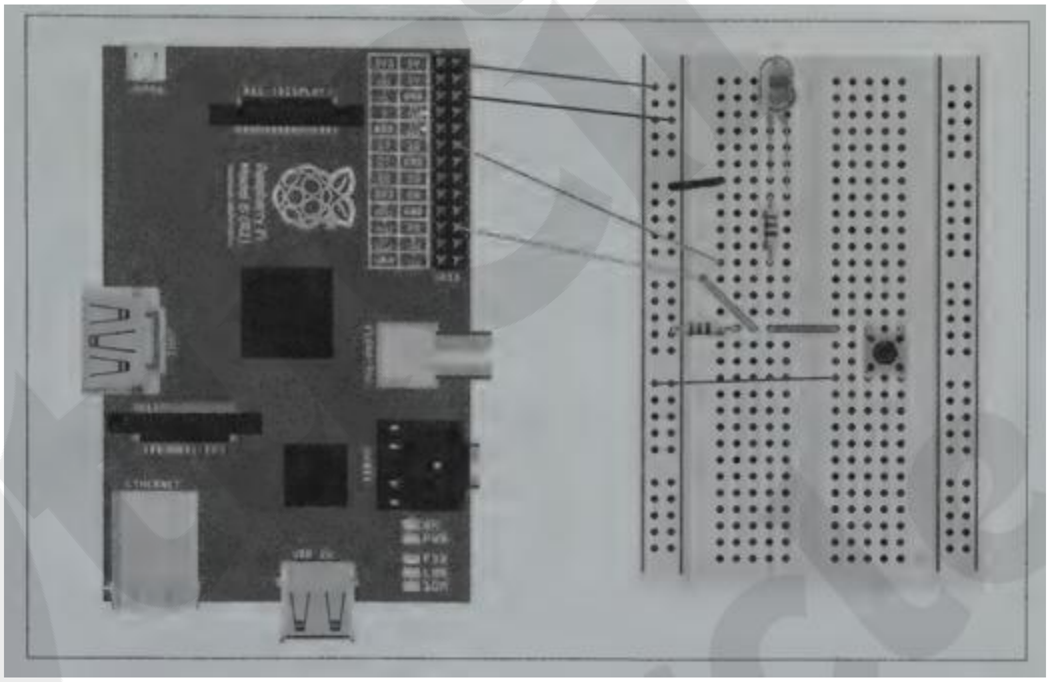


Figure 4.10:Interfacing LED and switch with Rasberry Pi

Box 4.4: Python program for sending an email on switch press

```
import smtplib

from time import sleep
```

```
import RPi.GPIO as GPIO

from sys import exit

from_email = ‘<my-email>’

receipients_list = [’<receipient-email>’ ]

cc_list =[]

subject = ‘Hello’

message = ‘Switch pressed on Raspberry Pi’

username = ‘<Gmail-username>’

password = ‘<password>’

server = 'smtp.gmail.com:587’

GPIO.setmode (GPIO.BCM)

GPLO. setup (25, GPLO. IN)

def  sendemail (from_addr, to_addr_list, cc_addr_list,

subject, message,

login, password,

smtpserver) :

header = ‘From: %s \n’ % from_addr

header += ‘To:= %s  \n’ %  ‘ , ‘ .join(to_addr_list)

header += ‘Cc:= %s  \n’ %  ‘ , ‘ .join(cc_addr_list)

header += ‘Subject:= %s  \n \n’ %  subject

message = header + message

server = smtplib.SMTP (smtpserver)
```

```
server.starttls ()

server.login (login, password)

problems = server.sendmail(from_addr, to_addr_list, server.quit ()

while True:

try:

if (GPIO.input(25) == True):

sendemail(from_email, receipients_list,

cc_ list, subject, message, username, password, server)

sleep (.01)

except KeyboardiInterrupt:

exit ()
```

4.6.3 Interfacing a Light Sensor (LDR) with Raspberry Pi

So far you have learned how to interface LED and switch with Raspberry Pi. Now let us look at an example of interfacing a Light Dependent Resistor (LDR) with Raspberry Pi and turning an LED on/off based on the light-level sensed.

Figure 4.11 shows the schematic diagram of connecting an LDR to Raspberry Pi. Connect one side of LDR to 3.3V and other side to a 1 {uF capacitor and also to a GPIO pin (pin 18 in this example). An LED is connected to pin 18 which is controlled based on the light-level sensed. Box 4.5 shows the Python program for the LDR example. The readLDR() function returns a count which is proportional to the light level. In this function the LDR pin is set to output and low and then to input. At this point the capacitor starts charging through the resistor (and a counter is started) until the input pin reads high (this happens when capacitor voltage becomes greater than 1.4V). The counter is stopped when the input reads high. The final count is proportional to the light level as greater the amount of light, smaller is the LDR resistance and greater is the time taken to charge the capacitor.

Box 4.5: Python program for switching LED/Light based on reading LDR reading

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
```

```
LDR_PIN = 4
LED_PIN = 18
ldr_count = 0

def readLDR(pin):
    count = 0
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, False)
    time.sleep(0.1)
    GPIO.setup(pin, GPIO.IN)
    while (GPIO.input(pin) == GPIO.LOW):
        count += 1
    return count

while True:
    ldr_value = readLDR(LDR_PIN)
    print("LDR Value:", ldr_value)
    if ldr_value < 1000:
        GPIO.output(LED_PIN, GPIO.HIGH)
    else:
        GPIO.output(LED_PIN, GPIO.LOW)
    time.sleep(1)
```

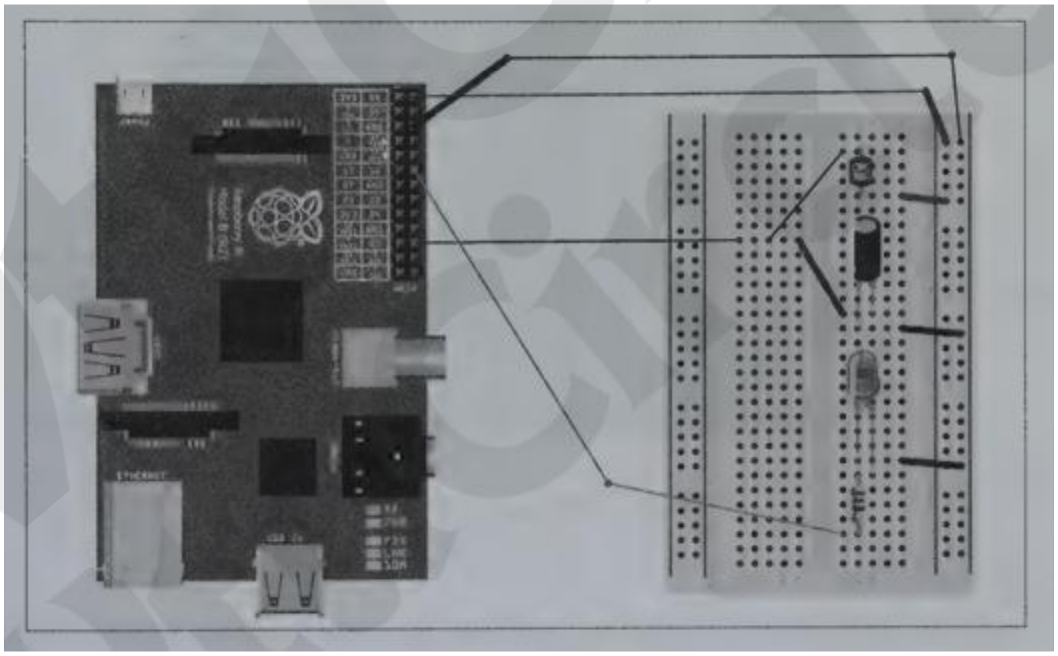


Figure 4.11:Interfacing LDR with Rasberry Pi

4.7 Home Automation

4.7.1 Smart Lighting

A design of a smart home automation system was described using the IoT design methodology. A concrete implementation of the system based on Django framework is described in this section. The purpose of the home automation system is to control the lights in a typical home remotely using a web application. The system includes auto and manual modes. In auto mode, the system measures the light in a room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light. Figure 7.12 shows the deployment design of the home automation system. As explained, the system has two REST services (mode and state) and a controller native service. Figures 7.13 and 7.14 show specifications of the mode and state REST services of the home automation system. The Mode service is a RESTful web service that sets mode to auto or manual (PUT request), or retrieves the current mode (GET request) . The mode is updated to/retrieved from the database. The State service is a RESTful web service that sets the light appliance state to on/off (PUT request), or retrieves the current light state (GET request). The state is updated to/retrieved from the status database.

Box 4.6: Django model for mode and state REST services - models.py

```
from django.db import models

class Mode (models.Model):

name = models.CharField(max_length=50)

class State (models.Model):

name = models.CharField(max_length=50)
```

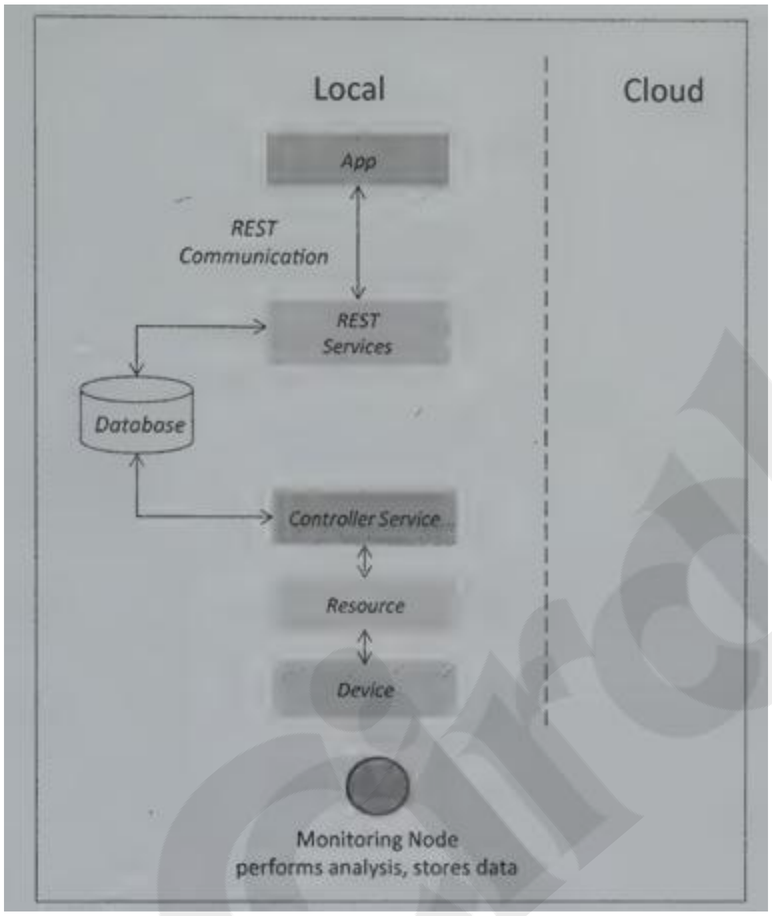


Figure 4.12: Deployment design of the home automation IoT system

Box 4.7: Serializers for mode and state REST services - serializers.py

```
from myapp.models import Mode, State

from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):

class Meta:

model = Mode

fields = ('url', 'name' )

class StateSerializer (serializers.HyperlinkedModelSerializer) :
```

class Meta:

model = State

fields = ('url', 'name')

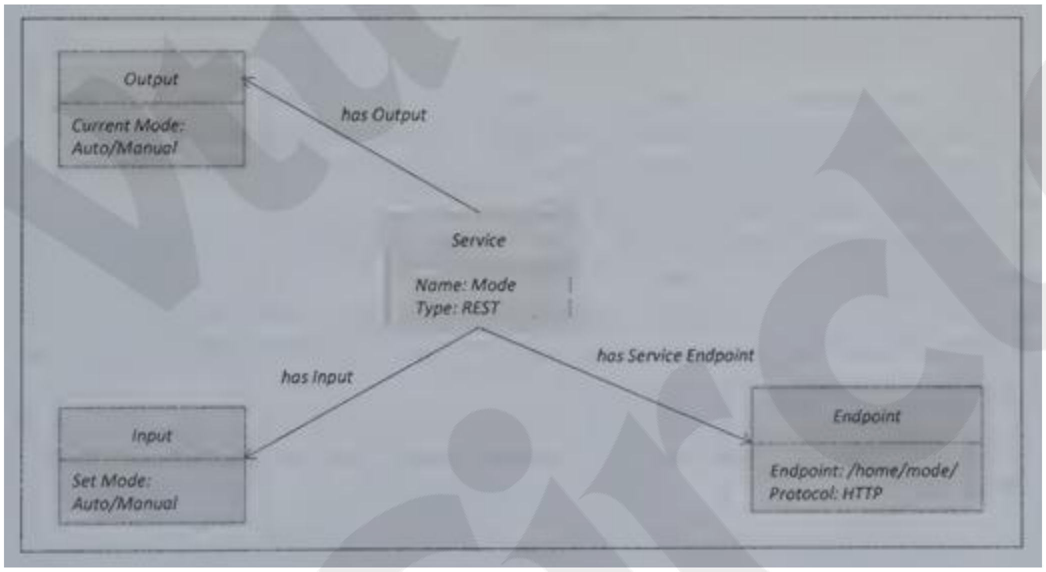


Figure 4.13: Service specification for home automation IoT system - mode service

After implementing the Django model, we implement the model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Box 4.2 shows the serializers for mode and state REST services. After implementing the serializers, we write ViewSets for the Django models. ViewSets combine the logic for a set of related views in a single class. Box 4.8 shows the Django views for REST services and home automation application. The ViewSets for the models (ModeViewSet and State ViewSet) are included in the views file. The application view (home) is described later in this section.

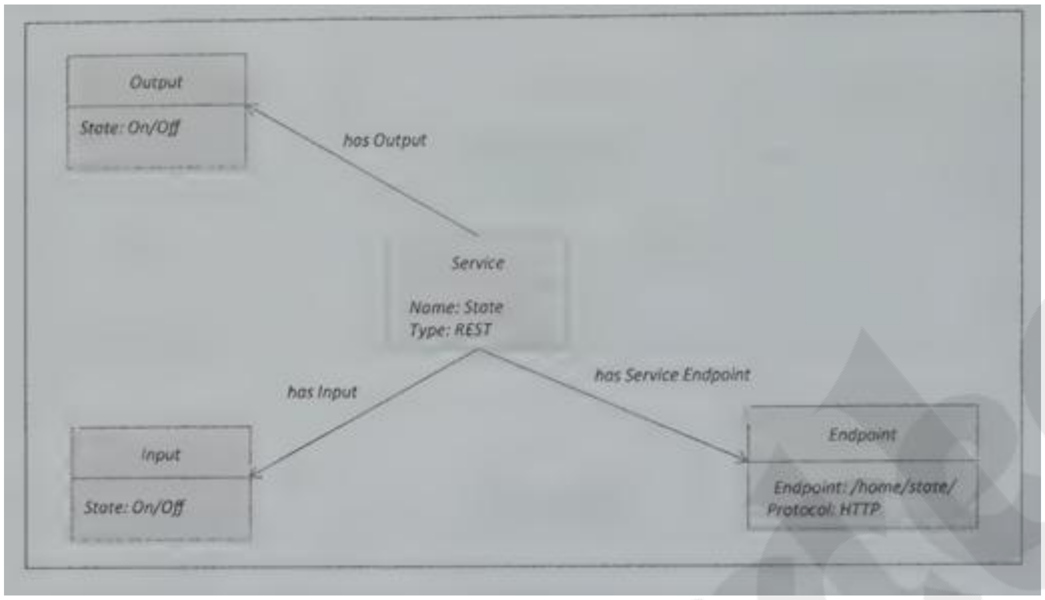


Figure 4.14: Service specification for home automation IoT system - state service

Box 4.8: Django views for REST services and home automation application - views.py

```
from myapp.models import Mode, State

from rest_framework import viewsets

from django.shortcuts import render_to_response

from django.template import RequestContext

from myapp.serializers import ModeSerializer, StateSerializer

import requests

import json

class ModeViewSet (viewsets.ModelViewSet) :

queryset = Mode.objects.all()

serializer class = ModeSerializer

class StateViewSet (viewsets.ModelViewSet) :

queryset = State.objects.all()
```

```
serializer_class = StateSerializer

def home (request):

outs=''

currentmode= ‘auto’

currentstate= ‘off’

if  ‘on’ in request.POST;:

values = "name":>- "on"

r=requests.put (’http://127.0.0.1:8000/state/1/’,

data=values, auth=(’myuser’, 'password’) )

result=r.text

output = json.loads (result)

out=output [’ name’ ]

Li Voth’

“tn. request .ROST:: values = "name": "off"

r=requests.put (‘http://127.0.0.1:8000/state/1/’,

data=values, auth=(’myuser’, 'password’ ) )

result=r.text

output = json.loads (result)

out=output [’name’ ]

Lia? if auto’ in,

request..POST ; values = "name": "auto"

r=requests.put (’http://127.0.0.1:8000/mode/1/’,
data=values, auth=('myuser', 'password' ) )

result=r.text

output = json.loads (result)

out=output [ 'name' ]

'manual' in

request.POST:

values = "name":

"manual"

r=requests.put ( http://127.0.0.1:8000/mode/1/',

data=values, auth=('myuser', 'password') )

result=r.text

output = json.loads (result)

out=output [ ' name' ]

r=requests.get ('http://127.0.0.1:8000/mode/1/",

auth=('myuser', ''password' ) )

result=r.text

output = json.loads (result)

currentmode=output [' name' ]

r=requests.get

(http: //127.0.0.1:8000/state/1/',

auth=('myuser', 'password' ))

result=r.text
```

output = json.loads (result)

currentstate=output [‘name’]

return render _to_response(’lights.html’,’r’:out,

'currentmode’ :currentmode, ’currentstate’ :currentstate,

context_instance=RequestContext(request)

Box 4.9 shows the URL patterns for the REST services and home automation application. Since ViewSets are used instead of views for the REST services, we can automatically generate the URL configuration by simply registering the viewsets with a router class. Routers automatically determine how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

Box 4.9: Django URL patterns for REST services and home automation application - urls.py

from django.conf.urls import patterns, include, url

from django.contrib import admin

from rest_framework import routers

from myapp import views

admin.autodiscover ()

router = routers.DefaultRouter ()

router.register(r’mode’, views.ModeViewSet)

router.register(r’state’, views.StateViewSet)

urlpatterns = patterns ('',

Senet Ganon

include (router.urls)),

url (r'‘‘api—auth/'', include (’rest_framework.urls’,

```
namespace=' rest_framework')),

url(r'admin/', include (admin.site.urls)),

url(r'*home/'',

'myapp.views.home'),

}
```

Box 4.10 shows the code for the Django template for the home automation application.

Box 4.10: Django template for home automation application - index.html

```
<!DOCTYPE html>

<html>

<head>

<body>

<p> {ix} }<p>

<h3>State</h3>

<form action="" method="post">{% csrf_token %}

<input type="submit" name="0n" value="on" />

<input type="submit" name="off" value="off" />

<br>

<h3>Mode</h3>

<form action=""method="post">{% csrf_token %}

<input type="submit" name="auto" value="auto" />

<input type="submit" name="manual" value="manual" />

</form>
```

```
</body>

</html>
```

Figure 4.15 shows a screenshot of the home automation web application.

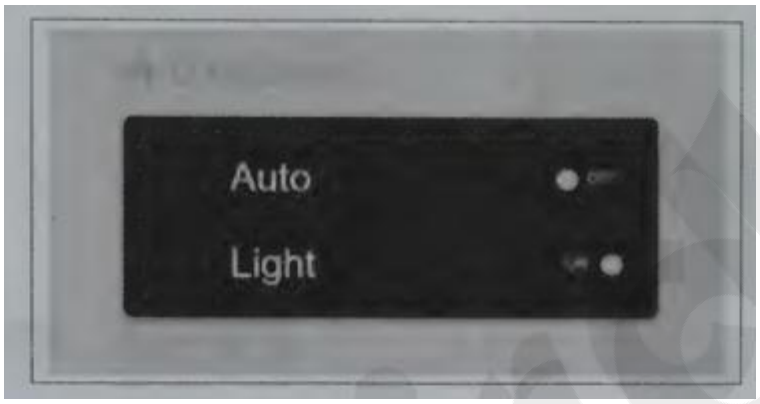


Figure 4.15: Home automation web application screenshot

Figure 4.16 shows a schematic diagram of the home automation loT system. The devices and components used in this example are Raspberry Pi mini computer, LDR sensor and relay switch actuator. Figure 4.17 shows the specification of the controller native service that runs on Raspberry Pi. When in auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. When in manual mode, the controller service, retrieves the current state from the database and switches the light on/off. A Python implementation of the controller service is shown in Box 4.11.

Box 4.11 Python code for controller native service - controller.py

```
import time

import datetime

import sqlite3

import spidev

import RPi.GPIO as GPIO

#Initialize SQLite

con = sqlite3.connect ('database.sqlite' )
```

```
cur = con.cursor()

#LDR channel on MCP3008

LIGHT_CHANNEL = 0

#GPIO Setup

GPIO.setmode (GPIO.BCM)

LIGHT_PIN = 25

# Open SPI bus

spi = spidev.SpiDev ()

spil.open (0,0)

#Light Level Threshold

threshold=200

readLDR():

#Function to read LDR connected to MCP3008

def readLDR():

light_level = ReadChannel(LIGHT_CHANNEL)

lux = ConvertLux(light_level, 2)

return lux

#Function to convert LDR reading to Lux

def  ConvertLux(data,places) :

R=10 #10k-ohm resistor connected to LDR

volts= (data *3.3)/1023

volts = round(volts,places)
```

```
lux=500« (3.3-volts) / (R*volts)

return lux

Function to read SPI data from MCP3008 chip

def  ReadChannel (channel):

adc =spi.xfer2([1, (8t+channel) <<4,0])

data = ((adc [1]&3)<<8+adc[2]

return data

# get current state from DB

getCurrentMode():

cur.execute('SELECT * FROM myapp_mode' )

data = cur.fetchone()

return data[l]

#Get current state from DB

def getCurrentState():

cur.execute('SELECT * FROM myapp_state' )

data = cur.fetchone ()

return data[1]

#Store current state in DB

def setCurrentState(val):

query=' UPDATE myapp_state set name=""'+valt'"'

cur.execute (query)

def switchOnLight (PIN) :
```

```
GPIO.setup (PIN, GPIO.OUT)

GPIO.output (PIN, True)

def switchOffLight (PIN):

GPIO.setup (PIN, GPIO.OUT)

GPIO.output (PIN, False)

def runManualMode () :

current State=getCurrentState ()

if currentState== ‘on’:

switchOnLight (LIGHT_PIN)

elif currentState== ‘off’:

switchOffLight (LIGHT_PIN)

def

runAutoMode():

#Read LDR

light level=readLDR ()

if

lightlevel < ldr_threshold:

switchOnLight (LIGHT_PIN)

else:

switchOffLight (LIGHT_PIN)

print “Manual” +” — “+getCurrentState()

#Controller main function
```

```
def runController():

currentMode=get CurrentMode ()

if currentMode==' auto’:

runAutoMode ()

elif currentMode==’manual’:

runManualMode ()

return true

while True:

runController ()

time.sleep
```

4.7.2 Home Intrusion System

A concrete implementation of a home intrusion detection system is described in this section. The purpose of the home intrusion detection system is to detect intrusions using sensors (such as PIR sensors and door sensors) and raise alerts, if necessary. Figure 4.16 shows the process diagram for the home intrusion detection system. Each room in the home has a PIR motion sensor and each door has a door sensor. These sensors can detect motion or opening of doors. Each sensor is read at regular intervals and the motion detection or door opening events are stored and alerts are sent. Figure 4.17 shows the domain model for the home intrusion detection system. The domain model includes physical entities for room and door and the corresponding virtual entities. The device in this example is a single-board mini computer which has PIR and door sensors attached to it. The domain model also includes the services involved in the system. Figure 4.18 shows the information model for the home intrusion detection system. The information model defines the attributes of room and door virtual entities and their possible values. The room virtual entity has an attribute ’motion’ and the door virtual entity has an attribute ’state’. The next step is to define the service specifications for the system. The services are derived from the process specification and the information model. The system has three services - (1) a RESTful web service that retrieves the current state of a door from the database or sets the current state of a door to open/closed, (2) A RESTful web service that retrieves the current motion in a room or sets the motion of a room to yes/no, (3) a native controller service that runs on the device and reads the PIR and door sensors and calls the REST services for updating the state of rooms and doors in the database. Figures 4.19, Figures 4.20 and 4.21 show specifications of the web services and the controller service.

Case Study 2: IoT in Smart Cities

Title: *Smart Waste Management in Pune, India*

Overview:
The Pune Municipal Corporation implemented an IoT-based waste management system to monitor garbage levels and optimize collection routes.

Design Highlights:

- **Sensors Used:** Ultrasonic fill-level sensors in bins

- **Connectivity:** LoRaWAN for low-power, long-range communication
- **Platform:** Central dashboard showing bin status and GPS tracking of collection vehicles
- **Analytics:** Predictive analysis of waste generation trends

Outcome:

- Improved garbage collection efficiency by 40%
- Reduced fuel usage and pollution from waste trucks
- Increased citizen satisfaction and hygiene levels

Case Study 3: IoT in Agriculture

Title: *Smart Irrigation System for Precision Farming*

Overview:
Farmers in Karnataka adopted an IoT-based system to monitor soil moisture and automate irrigation, especially during drought-prone seasons.

Design Highlights:

- **Sensors Used:** Soil moisture sensors, humidity and temperature sensors
- **Connectivity:** GSM/GPRS modules for remote areas

- **Actuators:** Automated water pumps and valves
- **Platform:** Cloud-based data monitoring with mobile alerts

Outcome:

- Saved up to 30% water through optimized irrigation
- Improved crop yield and reduced manual labor
- Enabled data-driven decisions for fertilizer and pest control

For all the above case studies you need to understand the design, circuitry and code.